AppendFilesExcept Method

Declaration (protected method)
procedure AppendFilesExcept(const destfile, fromfile: String; notfiles: TStringList);

Description
This method assumes that both **destfile** and **fromfile** are multi-file archives. It appends all the compressed files in **fromfile** to **destfile**, except those in the **notfiles** list.

If **destfile** does not exist, it is created and will end up containing either a full copy or a subset of **fromfile**, depending on the contents of **notfiles**.

This method is called by CompressFiles and DeleteFiles.

Blobstream Property

Applies to
TCBlobstream object.

Declaration
property Blobstream: TBlobstream;

Description
Run time and read only. This is a regular TBlobstream which can be used to directly access the underlying (and probably compressed) data in a TCBlobfield.

If you use this property for any data access, you should **not** also use the data access methods and properties native to the TCBlobstream, because these maintain and manipulate independent in-memory buffers until you free the TCBlobstream.

See the TCBlobstream object for more information.

C++ Builder Information

Like the Visual Component Library provided with C++ Builder, TCompress is written in Pascal. However, Borland have made it exceptionally easy to mix C++ and Pascal units in the C++ Builder environment. TCompress includes all the files needed to support C++ development, including .MAK (project) files for each of the example projects.

Key points to be aware of when using TCompress with C++ Builder:

- C++ Builder's VCL is based on Delphi 2.0, not Delphi 3.0. In practise, this simply means that the TCDBRichText component is not available for the current version of C++ Builder. Everything else should function as documented.
- The OBJ (binary) and HPP (include) files provided are for projects you create using C++ units. If you intend to use TCompress with code from any Pascal units (including any of the example projects supplied with TCompress), you must make sure that the Pascal DCU files are *also* available in your library path. This will be the case if you follow the installation instructions below.
- This help file and the example projects provided for TCompress all use Pascal syntax. To obtain full C++ typing and syntax for each of the methods, constants, types and properties in TCompress, print the **compress.hpp** and **compctrl.hpp** include files. Then simply call the desired method with the appropriate syntax, e.g. Compress1.CompressFile("c:\\myfiles.arc",filename,coLZH5);

Installation Instructions for C++ Builder:

- Create a directory for TCompress under the C++ Builder root directory, such as **CBuilder\lib\tcomp**
- Move the contents of Compress.zip into this directory, then unzip **OBJs.zip**
- See the Installing Help notes below for context-sensitive help at design time
- BACKUP your **bin/cmplib32.dcl** file, and backup and remove any earlier TCompress components
- Run C++ Builder and using its Component|Install dialog, add **;$(BCB)\lib\tcomp** to the Search path, then click **Add** and type **compress**. Click **Add** again and type **compctrl**. Click **OK**.
- All being well, a library rebuild will take place and a new Compress tab will appear on your component palette. This contains the Compress, CDBMemo (Compressed Database Memo) and CDBImage components.
- BACKUP the **examples\data\BIOLIFE.*** files, as this database is used by COMPDEMO to show off blob compression
- Ensure the **DBDEMOS** alias is pointing to the directory containing the **BIOLIFE** database
- Load the COMPDEMO project (**lib\tcomp\Compdemo.mak**) and run it.

Notes:

- TCompress and its companions provide very comprehensive data compression capabilities. Don't let this intimidate you -- most developers might only need to drop TCompress on a form and call its CompressFile/ExpandFile methods and no more... or maybe drop CDBMemo/CDBImage components on as well, set their properties and simply interact with them -- no programming required at all.
- Don't miss the many additional source code projects and examples in **BLOBDEMO.PAS (BLOB.MAK), SELFEXTF.PAS (SELFEXTR.MAK), BMTESTF.PAS (BMTEST.MAK), ARC2BLOB.PAS, ARC2MEM.PAS** and **RLECOMP.PAS.** There is a description of each one in README.TXT and many comments in each source file.

OPENHELP: Installing Context Sensitive Help

You need to use the OpenHelp utility to do this. In the initial release of C++ Builder, it was not automatically installed, but you can find the files on the CD in **\images\common\borland\openhelp**. Copy them to a **\program files\borland\common files\openhelp** directory and run the Openhelp program. Then:

- Select the IDE Files entry in the Search Ranges panel
- Click the Add button in the Available Help Files list
- Browse until you find the Compress.hlp file (should be in CBuilder\lib\tcomp)

- Find the new entry in the Available Help Files list, select it and click the '>' button to make it part of the IDE Files search range
- Exit OpenHelp. Next time you run C++ Builder, you'll be able to use context sensitive help with TCompress components and properties at design time.

CField Property

Applies to
TCDBMemo, TCDBImage components.

Declaration
property CField: TCBlobfield;

Description
Run time and read-only. This is just a public read-only version of the protected CompressedField property.

COMPDEMO Application

This application demonstrates the main features of TCompress, using a simple drag-and-drop user interface.

See installation to set your system up to properly compile and run COMPDEMO (especially if you are using Delphi 1.0).

Once it is running, select the desired compression method using the radio buttons, then drag files, images or memo text to and from the various components on the screen. In general, everything can be dragged everywhere, and compression or expansion is automatic.

You can change the archive name and path (top right), and you can change directories (center left) to expand or compress files to/from different directories. The entire archive can be expanded to the current directory by dragging the gray panel which holds the archive list.

Images and text from the compressed BIOLIFE database fields at the bottom of the screen can be dragged into and out of the directory or archive lists, and images can be dragged to and from the picture panel (a simple TImage) in the center of the screen.

You can cause the current database image or memo to be rewritten with the current compression method by right-clicking on it (this is useful for reverting the database blobs to None after you've finished testing!).

Once you have a feel for how COMPDEMO works, have a look at the code in **compmain.pas** -- the TCompress interface routines are near the top; the rest is there to provide for the drag-and-drop interface. Just scan the code for references to the Compress1, CDBImage1 and CDBMemo1 components.

For a simple database application, you'd just need to drop a TCompress and one or more TCDBMemo/TCDBImage components on, set them up, and away you go -- no coding required...

There is more example code which many comments in the **BLOB.DPR** project (read the **BLOBDEMO.PAS** header), and in the files **RLECOMP.PAS**, **ARC2MEM.PAS** and **ARC2BLOB.PAS**.

CheckHeader Method

Declaration (protected method) v Changed in V3.5
procedure CheckHeader(compressedFile: TStream;var hdr: TCompressHeader; var CID: string);

Description
Used to verify that **compressedFile** is a valid multi-file archive. One of several possible exceptions is raised if not. If its size is not enough to hold a header, the **hdr** data is set to sensible default values (including the passed-in value of cID), and no further processing is done.

Otherwise **hdr** is read from the stream (and **cID** set to its ComMethodID). The stream position is left at the end of the header, and checks are made on the **hdr**'s CompressedFileHeaderID, archiveType (must be caMulti) and ComMethodID.

V3.5 change: The **compressedFile** stream was formerly rewound to the start before the check. Now it is checked from the current position. This change facilitates the creation of self-extracting EXEs (see ExpandFilesFromStream).

CheckSpaceBeforeExpand Property

Applies to
TCompress component.

Declaration New in V3.05
property CheckSpaceBeforeExpand: Boolean; default is False

Description
In versions of TCompress prior to v3.05, a checks was always made before expanding a file to ensure that there was enough free disk space for it to be written.

Since the introduction of drives >2GB, the drive space information returned by Delphi's **diskfree** function for such drives has proven to be inconsistent and unreliable, especially in networked environments. As a consequence, this new property has been introduced to allow you to disable the checks made by TCompress. Note the default setting of False.

Compress Method

Declaration
procedure Compress(compressedStream, uncompressedStream: TStream; compressionMethod: TCompressionMethod);

Description
This compression routine is for single-item archives rather than the multi-file archives supported by CompressFiles et al.

For example, this method is used for compression of database memos and images by TCDBMemo and TCDBImage.

You can use it with any stream type, such as TFileStream, TMemoryStream or TBlobStream.

The compressed data is stored in **compressedStream** with a small header indicating the compression method. If the compressed data is **larger** than the uncompressed original, the header is eliminated and the original is stored instead.

Expand is used to expand data which has been compressed using this method.

CompressFile Method

Declaration
procedure CompressFile(const arcFile: String; var fromFile: String; compressionMode: TCompressionMethod);

Description
A simple single-file interface to the functions provided by CompressFiles. The **fromfile** parameter may be amended or set to an empty string if an OnCheckFile event handler is defined.

See also Filename Handling in Compressed Archives, TargetPath and ExceptionOnFileError.

CompressFiles Method

Declaration
procedure CompressFiles(const arcFile: String; whichfiles: TStringList; compressionMode: TCompressionMethod);

Description
Compresses one or more files from the **whichfiles** list into the **arcfile** multi-file archive.

If **arcfile** already exists, the files are updated or appended as appropriate.

If an OnCheckFile event handler is defined, it will be called with a process mode of **cmCompress** before each file is compressed.

When CompressFiles returns, **whichfiles** contains only those files which were actually compressed (with the paths set by OnCheckFile).

CompressFile provides a simpler single-file interface to this method.

See COMPDEMO for an example of the use of this method.

See also Filename Handling in Compressed Archives, TargetPath and ExceptionOnFileError.

CompressFilesToStream Method

Declaration v Changed in V3.5
procedure CompressFilesToStream(dest: TStream; whichfiles: TStringList; compressionMode: TCompressionMethod);

Description
This method is called by the CompressFiles method and is very similar.

The main differences are:
- The files are compressed into the **dest** stream, which can be of any stream type, e.g. TFileStream, TMemoryStream or TBlobStream. This means that you can, for example, compress a number of files into a database blob using multi-file archive format.
- Unlike CompressFiles, this method will not handle updating or appending of files in the archive. If there is already data in the **dest** stream, it is overwritten (from the current position) with a new multi-file archive containing the files in **whichfiles**.
- While there are stream-based methods for expanding files (ExpandFilesFromStream) and scanning the archive (ScanCompressedStream), there is no method similar to DeleteFiles.

V3.5 changes:
- This procedure used to be a function, with special truncation handing required if it returned non-zero. This is no longer required. However, the routine now assumes that the original size of the **dest** stream is less than or equal to its final size when all the files are compressed into it.

- Prior to V3.5, the writing always commenced from the *start* of **dest**. From V3.5 on, writing starts from the *current position* in dest, making CompressFilesToStream very handy for making self-extracting archives (archives which are appended to an EXE). See ExpandFilesFromStream for more information on this.

See also Filename Handling in Compressed Archives, TargetPath and ExceptionOnFileError.

CompressSource Property

Applies to
TCDBMemo, TCDBImage components, TCBlobfield object.

Declaration
property CompressSource: TCompress;

Description
This links the database component to a TCompress object which will handle data compression and expansion for it.

CompressStreamToArchive

v New in V3.0

Declaration

procedure CompressStreamToArchive(const arcFile: string; uncompressedStream: TStream;const filename: string; compressionMode: TCompressionMethod);

Description

This method is analogous to the CompressFile method but it can be used to independently save *any* stream to an archive, not just a specific file. It is most useful if, for example, you want to use an archive as a standalone data management system, working directly to and from it via streams *without* using files as intermediaries. Thus, for example, you might:

- Save data directly into an archive with this method (using a unique, dummy value for **filename**)
- Extract it directly from the archive using ExpandStreamFromArchive

Notes:

- No checks are made as to whether **filename** is already used in the archive -- the new information is simply appended in compressed form to the end of the archive. If you are in any doubt, call DeleteFiles first
- **UncompressedStream** is compressed from its current position i.e. there is no seek to its start first, and the data from the remainder of the stream is compressed
- If you want to store miscellaneous stream data in *non-file* multi-file archives, see CompressStreamToArchiveStream

CompressStreamToArchiveStream

Declaration v Changed in V3.5
procedure CompressStreamToArchiveStream(dest, source: TStream; const filename: string; fHdr: TCompressedFileHeader; var cID: string; compressionMode: TCompressionMethod);

Description
This method is called by CompressStreamToArchive and CompressFilesToStream to compress a stream's contents to the current archive position. You are unlikely to need to call it directly unless you are planning to compress *non-file* streams into a multi-file archive which is itself *not* stored in a file. All other cases are covered by the two methods mentioned above.

The rest of this section is therefore provided in case you do want to handle the exceptional case.

Because this method is operating at a fairly low level, a number of important pre-processing and post-processing steps should be taken to ensure that the integrity of the archive is maintained. Should you omit any, the archive might still be usable for your purposes, but may not be valid for access by other TCompress methods such as ExpandFilesFromStream etc.

Before calling CompressStreamToArchive:
- Set **cID** to an appropriate Compression method ID (e.g. 'LZH')
- Open the **source** stream, set to the desired start point for compression
- Open the **dest** (archive) stream and seek to its end
- Set fHdr's DateTime and Attribute fields to suitable dummy values
- Set **filename** to a suitable and *unique* value

During the call, the method itself will:
- Write the **fHdr** and **filename** information to the archive
- Compress the source stream into the archive using DoCompress
- If compressing makes the data *larger* (unlikely, but possible), rewrite the data in uncompressed form, and set **cID** to 'NON' (no compression)
- Set fHdr's Fullsize, CompressedSize, CompressedMode, FilenameLength and Locked fields
- Alter **cID** if DoCompress calls a custom compression method which changes it

After calling CompressStreamToArchiveStream, if you want a 100% valid archive, you should update the archive header:
- Read the TCompressHeader information from the beginning of the archive
- Increment the its Fullsize field by the size of the **source** stream from the position at which compression was requested
- If **cID** has changed, alter archive header's ComMethodID field as well
- Rewrite the TCompressHeader at the start of the archive

V3.5 change: This method previously had a **truncatepos** parameter which required special handling. This is no longer the case, but the method assumes that the **dest** stream starts out less than or equal to its eventual size when **source** has been compressed into it.

CompressString Method

Declaration v New in V3.5
function CompressString(const unCompressedString; compressionMethod: TCompressionMethod): string;

Description
Compresses **unCompressedString** using **compressionMethod** -- leaves it unchanged if it won't get any smaller. Use ExpandString to restore the compressed string to its original value. All checksum, OnShowProgress, encryption and other processing is in force.

Because each compressed string has a regular TCompress header at its start, string compression has a fixed overhead of 21 bytes and thus is likely to be ineffective on strings less than 30 characters long.

Example:
```
setLength(s,100000);                { make a looong string full of x's }
fillchar(s[1],Length(s),'x');
s := Compress1.CompressString(s,coLZH5); { now make it a whole lot smaller }
showmessage('Compressed length is '+IntToStr(Length(s))+' ('+
          IntToStr(Compress1.CompressedPercentage)+'% compressed)');
s := Compress1.ExpandString(s);          { back to original content }
showmessage('Expanded (original) length is '+IntToStr(Length(s))+#13#10+
             copy(s,1,40)+' (etc)');
```

CompressedField Property

Applies to
TCDBMemo, TCDBImage components.

Declaration
property CompressedField: TCBlobfield;

Description
Run time only. The TCBlobfield provides a handle for all access to the field data, handling compression and expansion as required.

CompressedPercentage Property

Applies to
TCompress component.

Declaration
property CompressedPercentage: Integer;

Description
This run time property notes the percentage effectiveness of the last compression operation.

For example, **0%** means it has failed to compress the data at all, and the original data has been saved instead; **95%** means the compressed data is only **5%** the size of the original data.

CompressionMethod Property

Applies to
TCDBMemo, TCDBImage components, TCBlobfield object.

Declaration
property CompressionMethod: TCompressionMethod;

Description
This determines what compression method will be used when this field is saved.

The method used will be auto-detected on expansion. Thus, there is no danger in varying the compression method between compressions if you so wish.

It is always possible that the data cannot be compressed, in which case it will be saved in its original form, with no archive header.

CompressionTime Property

Applies to
TCompress component.

Declaration
property CompressionTime: Longint;

Description
This run time property notes how long the last compression or expansion operation took, expressed in 1ms clock ticks.

The timing is limited to the compression/expansion operation itself, not any associated overheads such as file or database opening.

The compression time for TCDBMemo/TCDBImage fields cannot be reliably judged. This is because the Borland Database Engine usually follows a record write by one or more reads. By the time you get a chance to check CompressionTime, it holds the time of the expansion that occurred during the last read operation.

DataField Property

Applies to
TCDBMemo, TCDBImage components.

Declaration
property DataField: TField;

Description
This is just the same as the regular DataField property.

However, you should be aware of an important design-time difference which is detailed in the DataSource property.

DataSource Property

Applies to
TCDBMemo, TCDBImage components.

Declaration
property DataSource: TDataSource;

Description
This is exactly as it would be for a regular TDBMemo or TDBImage, with one important difference:

Special Note:
The dataset (TTable or TQuery) to which the DataSource is linked **must not** initially contain the same DataField as the one specified in the TCDBMemo or TCDBImage.

Thus, if your TCDBImage component has a DataField of **Graphic**, and its DataSource is linked to Table1, then Table1 should not (initially) have a **Graphic** field in its field list.

If it does, you will see the following warning:

TCompress warning: Table1 already has a non-compressed field called 'Graphic'.

Double click on Table1, remove the existing field, and try again.

The reason for the message is that Table1's regular TBlobfield cannot handle compression and must be supplanted by a TCBlobfield which **can**. Unfortunately, Delphi's implementation does not make it possible for TCDBImage or TCDBMemo to make this kind of replacement automatically.

Once you have removed the old field, specify your TCDBImage's DataField property again to create its TCBlobfield replacement. The **Graphic** field may subsequently reappear in Table1's field list -- that's ok, as it will be the one created by TCDBImage.

DeleteFiles Method

Declaration
procedure DeleteFiles(const arcFile: String; whichfiles: TStringList);

Description
Deletes one or more files in **whichfiles** from the **arcfile** multi-file archive.

If an OnCheckFile event handler is defined, it will be called with a process mode of **cmDelete** for each file before any file is deleted.

When the DeleteFiles returns, **whichfiles** contains only those files which were actually deleted.

See COMPDEMO for an example of the use of this method.

Differences -- Delphi 1.0, 2.0 and 3.0

TCompress V3.0 includes units for Delphi 1.0 (D1DCUs.ZIP), 2.0 (D2DCUs.ZIP) and 3.0 (D3DCUs.ZIP).

Please do *not* mix these units and projects made with them in the same directories.

If you write projects for both Delphi 1.0 and later versions with common source and forms, you'll want to be aware of the following differences when using TCompress:

1. Operationally, there is virtually no difference. Both versions compress to compatible file or blob formats.
2. 16 bit projects which decompress archives created with 32-bit projects may require special handling in the OnCheckFile event to deal with long filenames. They are not modified by TCompress itself.
3. The parameter list for OnCheckFile in Delphi 1.0 uses an **OpenString** for the filename parameter -- in later versions of Delphi this is simply a **string**.
4. If you load a form created with Delphi 2.0 or later into Delphi 1.0, you'll see warnings about the Blobtype property for any CDBImage or CDBMemo objects you have. This is a new property added to blob fields in Delphi 2.0. The warning can be ignored if you are moving software back to Delphi 1.0.
5. The LoadCompressedResource and LoadExpandedResource methods have been successfully tested with Delphi 2.0 and 3.0. On *one* our Delphi 1.02 development machines, we encountered intractable GPF problems with programs **compiled** using these functions. When compiled on another very similar machine, the same program (SELFEXTR.DPR) was fine, no matter which machine it executed on. Your own experiences may differ.
6. The TCDBRichText component is only supported under Delphi 3.0.

The demonstration files and projects which come with TCompress are interchangeable between all versions of Delphi so long as you keep the above points in mind.

DoCompress Method

Declaration
function DoCompress(compressedStream, uncompressedStream: TStream; compressionMethod: TCompressionMethod; var compressID: String; var checksum: Longint): Longint;

Description
This is the lowest level public function offered by TCompress. It compresses the data from the current position in **uncompressedStream**, writing it to the current position in **compressedStream**.

CompressionMethod determines the compression format to be used, and a **checksum** of the uncompressed data is returned (0 for coNone). The function return value is the compressed data size in bytes.

If **compressionMethod** is coCustom, **compressID** may be altered by the OnCompress event handler.

Generally speaking, you will not need to use this function directly.

DoExpand Method

Declaration
procedure DoExpand(expandedStream, unexpandedStream: TStream; compressedSize, fullSize, checksum: Longint; compressionMethod: TCompressionMethod; compressID : String; Locked: Boolean);

Description
This low-level method is used to expand bytes directly from one stream to another. It expands exactly **compressedSize** bytes from the current position in **unexpandedStream**, writing the result to the current position in **expandedStream**. The **fullSize** parameter is the original uncompressed file size, which some algorithms (e.g. LZH) require.

CompressionMethod specifies the compression format which is expected, and the **checksum** is compared against a newly-calculated checksum of the expanded data (except in the case of compression method coNone).

CompressID should be specified when **compressionMethod** is coCustom, but may be an empty string for the other modes.

The **Locked** parameter should be True if the locked flag for this particular item was set (see TCompressHeader or TCompressedFileHeader/TCompressedFileInfo).

One useful application of this method is to expand a file directly from the current location in a multi-file archive to a memory or blob stream -- see **ARC2MEM.PAS** for an example of this.

Events

OnCheckFile
OnCompress
OnExpand
OnRecognize
OnShowProgress

ExceptionOnFileError Property

Applies to
TCompress component.

Declaration
property ExceptionOnFileError: Boolean; Default is False

Description
In versions prior to V2.5, there were two occasions when TCompress would display a message box rather than generate an exception. These were:
a) If a file could not be opened for reading during compression to a multi-file archive
b) If a file could not be created or opened for writing during expansion from a multi-file archive

The interactive message is undesirable in unsupervised programs, so in V2.5 it was been removed. The default behaviour is now to *ignore* the error and continue with the next file. In such cases, the file(s) which could not be processed will be missing from the modified **whichfiles** list which is passed back by the compression or expansion routine, so after-the-event detection is possible. The advantage of this approach is that those files which *could* be processed will be.

However, you may prefer to have the routine generate an exception and end processing as soon as an error occurs. If so, set this property to True.

Exceptions

EUnrecognizedCompressionMethod
ComMethodID in the archive header is not recognized.

EInvalidHeaderArchiveType
The Archive type stored in the archive header is not valid for this operation. For example, an individual file which has been saved with the single-item Compress method is being expanded with the multi-file ExpandFile method instead of the Expand method.

EInvalidHeader
Unable to find a valid comID at the start of an archive header.

EBadChecksum
Data which has just been expanded does not have the same checksum as the original data before it was compressed. Data corruption or data access problems are the most likely causes.

EInsufficientDiskSpace
Not enough disk space when expanding or compressing a file. This is most likely to occur during updates to large archives.

EInvalidKey
This compressed data or file was compressed with a Key which differs from the one currently set.

Expand Method

Declaration
procedure Expand(expandedStream,unexpandedStream: TStream);

Description
This expands single-item archives created with Compress, rather than the multi-file archives supported by CompressFiles et al.

For example, this method is used for expansion of database memos and images by TCDBMemo and TCDBImage.

You can use it with any stream type, such as TFileStream, TMemoryStream or TBlobStream.

The compressed data is read from **unexpandedstream** and written to **expandedStream.** If the data in **unexpandedstream** has no archive header, it is assumed to have been stored in its original (uncompressed) form.

ExpandFile Method

Declaration
procedure ExpandFile(var toFile: String; const arcFile: String);

Description
A simple single-file interface to the functions provided by ExpandFiles. The **tofile** parameter may be amended or set to an empty string if an OnCheckFile event handler is defined.

See also:
v Filename Handling in Compressed Archives.
v TargetPath, to control the destination of the expanded file
v ExceptionOnFileError, to control whether a file creation error should generate an exception
v MakeDirectories, to control whether the target directory should be automatically created

ExpandFiles Method

Declaration
procedure ExpandFiles(const arcFile: String; whichfiles: TStringList);

Description
Expands one or more files in **whichfiles** from the **arcfile** multi-file archive.

If an OnCheckFile event handler is defined, it will be called with a process mode of **cmExpand** before each file is expanded.

When the ExpandFiles returns, **whichfiles** contains only those files which were actually expanded (with the actual paths set by OnCheckFile).

If **whichfiles** is nil, all files in the archive are expanded.

ExpandFile provides a simpler single-file interface to this method.

See also:
v COMPDEMO for an example of the use of this method.
v Filename Handling in Compressed Archives.
v TargetPath, to control the destination of the expanded files
v ExceptionOnFileError, to control whether file creation errors should generate an exception
v MakeDirectories, to control whether target directories should be automatically created

ExpandFilesFromStream Method

Declaration Changed in V3.5
procedure ExpandFilesFromStream(compressedStream: TStream; whichfiles: TStringList);

Description
This method is an exact analogy of the ExpandFiles method except that the expand is performed from **compressedStream**.

V3.5 changes:

- Prior to V3.5, this method would seek to the *start* of **compressedStream** before starting expansion. It now expands from the *current position*. This feature can be exploited to easily make self-extracting EXEs of any size (unlike the complex approach based on LoadCompressedResource which is shown in the **selfxsml.dpr/doinstall.pas** example code). To see an example of the new approach, download our TCompLHA component set and examine the **sfx.dpr** and **makeexe.dpr** projects. If references in that code to "CompLHA" are all changed to "Compress", the same code will work perfectly with TCompress instead of TCompLHA.

See CompressFilesToStream for more information about how streams may be used for flexible multi-file archive storage.

ExpandStreamFromArchive

v New in V3.0

Declaration

procedure ExpandStreamFromArchive(const arcFile: string; uncompressedStream: TStream; const filename: string);

Description

This method is analogous to the ExpandFile method but it can be used to uncompress data directly into a stream instead of to a disk file. It is most useful if, for example, you want to use an archive as a standalone data management system, working directly to and from it via streams *without* using files as intermediaries. Thus, for example, you might:

- Save data directly into an archive with CompressStreamToArchive (using a unique, dummy value for **filename**)
- Extract it directly from the archive using this method

Notes:

- No advance checks are made on whether there is sufficient disk space for the expansion operation
- From TCompress 3.03, **filename** comparisons are no longer case sensitive

ExpandString Method

Declaration v New in V3.5
function ExpandString(const CompressedString): string;

Description
If **CompressedString** contains a string which was compressed using the CompressString method, returns the expanded result, otherwise returns the string as-is. See the example in CompressString.

Filename Handling In Compressed Archives

v behaviour changed in V3.0

When a file is compressed into a multi-file archive with CompressFile, CompressFiles or CompressFilesToStream, the following things happen to its name:

- v If TargetPath is non-blank and is found at the left of the filename (e.g. **c:\targetpath**\Filename.txt), it is stripped from the filename (e.g. Filename.txt). The check is *not* case sensitive in TCompress 3.0.

- v If an OnCheckFile event is defined, this is called with the modified filename as a parameter -- and the stored name may be modified by the event handler (see the COMPDEMO source for an example of this)

- v After this call is made, any remaining drive information in the filename is stripped (thus **C:\ANOTHER_PATH\ThisFile.TXT** is only stored as **\ANOTHER_PATH\ThisFile.TXT**)

- v No case change is made to the stored filename in TCompress 3.0, and case is not important which specifying which files to extract from an archive.

After the above processing, the filename is stored in the archive. This has a bearing on what filename (and TargetPath) you should use to expand the file later. Ideally, the correct name should be obtained from a list generated by the ScanCompressedFile method. Failing that, be sure to omit any drive reference, and use correct (as stored) paths in the target name, e.g.

tofile := '\ANOTHER_PATH\thisfile.txt'; { case doesn't matter, but a complete match with the stored path *does* }
ExpandFile(tofile, 'c:\myfiles.arc'); { this will expand to \ANOTHER_PATH\ThisFile.txt }
tofile := 'Filename.txt'; { we have no absolute path here so... }
ExpandFile(tofile, 'c:\myfiles.arc'); { it will go to whereever TargetPath is set, or to the "current" directory if it TargetPath is blank }

See also:
v TargetPath, to control the destination of expanded files and the store paths of compressed files
v ExceptionOnFileError, to control whether file creation errors should generate an exception
v MakeDirectories, to control whether target directories should be automatically created

Please note: While the above process affects the *name* of the file stored, the actual *origin* of the file stored will not change as a result. Thus, even though the name **D:\WINDOWS\MYFILE.INI** is converted to **\windows\myfile.ini** for filename storage, the file which is compressed is still the file found on drive D.

FreeFileList Method

v behaviour changed in V3.0

Declaration
procedure FreeFileList(FInfo: TStringlist);

Description
If you use ScanCompressedFile or ScanCompressedStream to obtain a list of the files in a multi-file archive, it will populate a TStringList with a set of filenames *and* objects containing file information. To quickly clear this information when you no longer need it, and free the memory used by the TCompressedFileInfo objects, just call this method.

Note: In TCompress V3.0 this method will *not* free the Finfo TStringList itself, only clear it. See the source of the Compdemo application for an example of its use.

GetAllFilesInDirectory Method

v New in V3.0

Declaration
procedure GetAllFilesInDir(List: TStringList; Dirname: string; const Match: string; Anything: Boolean);

Description
Utility method to help with file and directory management when using TCompress. List will be filled with a list of files in the directory Dirname which match the wild card string given in Match (empty if none). If Dirname is empty, the Windows current directory is assumed.

For example, a Dirname of 'c:\delphi\source' with a Match value of '*.*' might return a List containing 'c:\delphi\source\Lisezvcl.txt' and 'c:\delphi\source\VCLinfo.txt'.

Filenames are always returned in **original** case, with full paths as shown above.

If Anything is True, the returned list includes System, Readonly, Hidden and Directory files (otherwise these will be omitted).

See also GetMatchingFiles, which provides a simpler interface.

GetFileHeader Method

Declaration (protected method)
function GetFileHeader(Stream: Tstream; var Fhdr: TCompressedFileHeader; var Filename: String): Boolean;

Description
This method tries to read **Fhdr** and **Filename** from the current position in **Stream**. Returns True if the reads are successful (data is not validated).

GetMatchingFiles Method

v New in V3.0

Declaration
procedure GetMatchingFiles(List: TStringList; const Matchname: string);

Description
Utility method to help with file and directory management when using TCompress. List will be filled with a list of files which match the directory-path-and-wildcard information in Matchname (empty if none). If no path is specified, the Windows current directory is assumed. A wild card **must** be specified (i.e. the Matchname cannot be empty or simply a directory path).

For example, a Matchname of 'c:\delphi\source*.rc' might return a List containing 'c:\delphi\source\Toconsts.rc'.

Filenames are always returned in **original** case, with full paths as shown above.

The returned list always excludes System, Readonly, Hidden and Directory files. If you want to find these files also, use GetAllFilesInDir instead.

Installation

For C++ Builder Installation instructions, see C++ Builder Information.

Installation Instructions:

***Delphi 1.0/2.0:**

- Unzip the correct DCUs file for your version of Delphi (e.g. D2DCUs.ZIP for Delphi 2.0)
- BACKUP your **complib.dcl** file, and backup and remove any earlier TCompress components
- Put **compress.dcu/.dcr** and **compctrl.dcu/.dcr** in a directory on Delphi's Component Library Search path
- In the install components dialog, click **Add** and type **compress**. Click **Add** again and type **compctrl**. Click **OK**.

***Delphi 3.0:**

- Unzip the D3DCUs.ZIP file into a suitable component directory
- From the Tools|Options|Library screen, add the above directory to your Library Path
- *If you have a TCompress 3.0 package installed,* remove it with Component|Install Packages, select and **Remove**
- File|Open the **compdb35.dpk** package source, click its **Compile** and **Install** buttons to create and install the package, then exit (note: do **not** use the Component menu's Install Packages option for this unless you've *previously* created the package with this step!)

***All versions:**

- All being well, a rebuild will take place and a new Compress tab will appear on your component palette. This contains the Compress, CDBMemo (Compressed Database Memo), CDBImage (Compressed Database Image) and, in Delphi 3.0, the CDBRichText (Compressed Database Rich Text) components.
- BACKUP the **demos\data\BIOLIFE.*** files, as this database is used by COMPDEMO to show off blob compression
- Ensure the **DBDEMOS** alias is pointing to the directory containing the **BIOLIFE** database
- Load the COMPDEMO project, compile and run it (if using Delphi 1.0, see the notes at the top of COMPMAIN.PAS). Delphi 3.0 users who are not familiar with runtime packages should turn OFF the Project|Options|Packages|Build With Runtime Packages option before testing each project they load (see Troubleshooting)

Notes:

- **Delphi 1.0:** All the demonstration projects were saved with Delphi 2. When you load them with Delphi 1.0, you will usually see warnings which arise from Delphi version differences. It is generally safe to ignore these warnings, but DO check the comments at the head of the main unit in case modifications are required for Delphi 1.0 (particularly for the COMPDEMO application).
- **Delphi 3.0:** If you don't intend to use Blob (memo/image/rich text) database compression, use the **comp35.dpk** package instead of the **compdb35.dpk** one. This will mean that projects you create will have no BDE dependencies.
- TCompress and its companions provide very comprehensive data compression capabilities. Don't let this intimidate you -- most developers might only need to drop TCompress on a form and call its ExpandFile/CompressFile methods and no more... or maybe drop CDBMemo/CDBImage components on as well and simply interact with them -- no programming required at all.
- Don't miss the many additional source code projects and examples in **BLOBDEMO.PAS (BLOB.DPR), SELFEXTF.PAS (SELFEXTR.DPR), BMTESTF.PAS (BMTEST.DPR), ARC2BLOB.PAS, ARC2MEM.PAS** and **RLECOMP.PAS,.** There is a description of each one in README.TXT and many comments in each source file.

Installing Help (for design-time help):

*****Delphi 1.0/2.0:**

- Put this help file in Delphi's **bin** directory
- Put the **compress.kwf** file in Delphi's **help** directory
- Run the **helpinst** utility, and select File|Open|**bin\delphi.hdx**
- Click the **+** icon and select **compress.kwf**
- Select File|Save to write the updated **.hdx** file, then exit
- You should now be able to access this help from within Delphi, e.g. by pressing F1 while the cursor is in the CompressSource property

*****Delphi 3.0:**

- Put this help file in Delphi's **help** directory
- In the same directory, use Notepad to add the following line to the Third Party Help section at the end of DELPHI3.CFG:

:Link compress.hlp

- Delete the **hidden** DELPHI3.GID file from that directory if there is one. This will be rebuilt next time you use the Delphi 3 help.

Introduction To Data Compression

Data compression encodes data so that it uses fewer bytes than usual. The amount of compression depends on the nature (content) of the data and the algorithm used.

Here are some typical examples -- your compression may vary:

Algorithm:	**RLE**	**LZH**	**LZW**
BMP images	50-80%	90-99%	90-95%
Text files	0-10%	30-60%	25-40%
DB files	30-50%	55-95%	50-80%

Data compression takes time. However, with a fast CPU, the time lost in data compression (done once) is usually balanced by a gain at expansion time (done frequently) due to a reduction in disk or network access.

One point about data compression: occasionally, a file **cannot** be compressed (try compressing a ".zip" file, for example). When that happens, TCompress will store the original data as-is, rather than a "compressed" copy.

Data compression is often used to "archive" files -- store multiple files in a compressed archive. TCompress includes easy-to-use multi-file archive functions.

TCompress's three built-in algorithms:

RLE: Run-Length-Encoding
This simple, quick compression replaces repeated characters (e.g. aaaaaaaaaaaa) with a "counter" code (e.g. 12a). It is most effective when you have numerous character repeats, such as in BMP files. It is least effective where such sequences are rare, such as in most text.

LZH1 and LZH5 : Lempel-Ziv-Huffman
The compression method is popular in compression utilities such as LHArc and others. Although its compression time is fairly slow, expansion is quick and it can achieve remarkable compression performance. **Note:** Even though TCompress supports LZH encoding, its archive formats are not compatible with those of LHArc and LHA. If you want compatibility with those programs, or if your projects mainly involve multi-file archiving and backup applications rather than general-purpose compression, see our TCompLHA component at http://www.spis.co.nz/complha.htm

v New in V3.5*:* Prior to V3.5, TCompress only supported LZH1 compression. It now also supports LZH5, which is a significantly faster, better process, so you should use it in preference. If you want to *disable* certain compression or expansion options to save a few KB of memory in your projects, this can be done very easily (with $DEFINEs) -- you should order the Compress unit source when you register the component.

About the third algorithm mentioned above:

LZW: Lempel-Ziv-Welch
The LZW algorithm is one of the better general-purpose compression algorithms. Many hardware and software products (such as high speed modems or .GIF files) use this algorithm because of its efficient compression and processing speed. However, its use is covered by US Patent Number 4,558,302, which requires a license from Unisys (a typical royalty is 0.95% of your application's retail price). As a consequence, very few developers now make use of LZW, and it is not provided in TCompress..

Key Property

Applies to
TCompress component.

Declaration Changed in V3.5
property Key: Longint; Default =0, meaning no protection

Description
This property allows compressed items or files to be "locked" in such a way that your program will not be able to decompressed them unless the key is set to the correct value first.

During compression, set this key to the desired value -- as it is a Longint, very high values can be used if need be. Note that in the case of a multi-file archive, it is possible to have some files protected with one key, some with a different key, and some not protected at all.

During expansion, if the key is not the correct value, an EInvalidKey exception will be generated.

Notes:
- V3.5 change: Earlier "bicycle lock" protection has been replaced by file encryption (*unless* a file is stored with a CompressionMethod of **coNone**). V3.5 will still unlock files protected with earlier versions.
- Be careful when experimenting with Key, particularly if you are using DoCompress, CompressString, compressing blobs or using CDBMemo/CDBImage. If Key is non-zero, protection will be applied in **all** these cases so make sure that is your intention!

For more information, see:
- TCompressHeader
- TCompressedFileHeader
- TCompressedFileInfo

LoadCompressedResource Method

Declaration
function LoadCompressedResource(ResourceName, DLLName: string): TStream;

Description
Use this method to retrieve a compressed item or archive from your EXE's resources (or those of a related EXE or DLL). The value returned is a handle of a newly-created stream which holds the **compressed** data which was stored under the name **ResourceName**. It is up to you to then expand the data to another stream using either Expand (if it was a single item) or ExpandFilesFromStream (if it was a multi-file archive).

If **DLLName** is blank, the resource is assumed to be in the current EXE. If non-blank, it must contain a full path to the target DLL or EXE file. All resources must be created (using the Borland Resource Compiler or Resource Workship) with a **type ID** of TCOMPRESS. Full instructions and examples of making compressed resources and incorporating them into projects can be found in the SELFEXTR.DPR and BMTEST.DPR sample projects. Example of decompressing a multi-file archive which was saved as a compressed resource:

```
    { code extract from the DOINSTAL.PAS unit of the SELFEXTR.DPR project }
    TempStream := LoadCompressedResource(RESOURCE_NAME,''); { get as COMPRESSED data in a
stream }
    try
        MakeDirectories := True;
        Targetpath := TargetDir;
        if TempStream<>nil then
          ExpandFilesFromStream(TempStream,nil); { expand the lot }
    finally
        TempStream.free;
    end;
 end;
```

See also: LoadExpandedResource, which is for dealing with individual resources such as bitmaps, rather than for multi-file archives.

Note: The **SFX.DPR** and **MAKEEXE.DPR** projects supplied with our TCompLHA component set show an easy alternative way of creating self-extracting archives, one which does *not* involve using resources. It works for any archive size and any version of Delphi. The same process would operate equally well with TCompress *provided* the TCompress source was amended so that the CheckHeader method did not seek to the start of a stream before compression/expansion.

Special note for Delphi 1.0 users:
Under Delphi 1.02 on *one* of our test machines (P160/32MB Win95+Service Pack #1+SP1 patches) we encountered major and inexplicable GPF problems which we were unable to resolve. The same application (SELFEXTR.DPR) was fine if **compiled** on a second similar machine. With that in mind, please consider this function as "unsupported" for 16-bit Delphi and use it at your own risk.

LoadExpandedResource Method

Declaration
function LoadExpandedResource(ResourceName, DLLName: string): TStream;

Description
This function is exactly analogous to the LoadCompressedResource function except that the stream it creates and returns will contain **already-expanded** data from the resource, not the original compressed data. This is most useful for loading common resources such as bitmaps, e.g.

```
{ code extract from the BMTEST example project }
ResourceStream := Compress.LoadExpandedResource(Resname,'');
try
  MyBitMap.LoadFromStream(ResourceStream);
finally
  ResourceStream.free;   { MUST make sure it gets freed }
end;
```

Note: The original resource data must have been compressed with the Compress method, not one of the multi-file-archive compression methods. See LoadCompressedResource for more information.

Special note for Delphi 1.0 users:
Under Delphi 1.02 on *one* of our test machines (P160/32MB Win95+Service Pack #1+SP1 patches) we encountered major and inexplicable GPF problems with LoadCompressResource, which this method calls, which we were unable to resolve. The same application (SELFEXTR.DPR) was fine if **compiled** on a second similar machine. With that in mind, please consider this function as "unsupported" for 16-bit Delphi and use it at your own risk.

MakeDirectories Property

Applies to
TCompress component.

Declaration
property MakeDirectories: Boolean; default is False

Description
When expanding files from a multi-file archive with ExpandFile, ExpandFiles or ExpandFilesFromStream, set this to True if you want the destination directory(ies) to be automatically created if required.

See TargetPath and Filename Handling In Compressed Archives for more information.

Methods

For multi-file archive files:

CompressFile CompressFiles
CompressStreamToArchive CompressString v New in V3.5
DeleteFiles ExpandFile ExpandFiles
ExpandStreamFromArchive ExpandString v New in V3.5
FreeFileList
GetAllFilesInDir
GetMatchingFiles
ScanCompressedFile

For multi-file archive streams:

CompressFilesToStream v Changed in V3.5
CompressStreamToArchiveStream v Changed in V3.5
ExpandFilesFromStream v Changed in V3.5
ScanCompressedStream

For single-item stream to stream:

Compress Expand

For stream to stream without header:

DoCompress DoExpand Recognize

For loading compressed resources:

LoadCompressedResource LoadExpandedResource

Protected methods:

AppendFilesExcept CheckHeader v Changed in V3.5
GetFileHeader PutFileHeader SetHeader

OnCheckFile Event

Declaration
procedure CheckFile(var filepath: String; mode: TCProcessMode);

Description
This event is called just **before** the multi-file action referred to by **mode** takes place for the current **filepath**.

This gives you the opportunity to seek user confirmation and:
v Skip the operation for the file in question by setting **filepath** to **CompressSkipFlag**.
v Skip this and all subsequent files by setting **filepath** to **CompressNoMoreFlag**
v Amend **filepath** so that it is either compressed with or expanded to a different path

Note: The TargetPath property lessens the need to make path amendments using OnCheckFile, and the MakeDirectories property means you don't need to worry about directory creation either.

When **mode** is **cmCompress**, **filepath** is the full path of the original file as passed to CompressFiles -- you can leave it as-is if you want path information stored in your archive, or shorten it to filename.ext. Note that any drive designation is automatically removed after this call (see Filename Handling in Compressed Archives).

When **mode** is **cmExpand**, **filepath** is the one that has been stored in the archive. If you don't amend it, the file will be expanded into that path (or into the current directory if there is no path). Alternatively, you can alter **filepath** to have it expanded into some other directory, or with some other filename.

See the **Compress1CheckFile** method in COMPDEMO for an example of a handler for this event.

Also note the minor parameter difference between Delphi 1.0, 2.0 and 3.0.

OnCompress Event

Declaration
procedure OnCompress(dest, source: TStream; var compressID: OpenString; var outputsize: Longint; var checksum: Longint);

Description
This event is called when custom compression has been requested by specifying a compression method of **coCustom**.

Your handler must:
- Set a three-letter compression ID in **compressID** (preset default is 'CUS')
- Compress the data from the **source** stream into the **dest** stream, using only methods common to all stream types, including write, read and seek.
- Leave both stream positions at the end of the read/written areas
- Count the precise number of bytes output to **dest** and set the **outputsize** variable accordingly
- Optionally calculate a **checksum** of the **source** data for validation on expansion (preset default is 0)

Notes
- No header management is required by your routine -- a header has already been placed in the **dest** stream prior to this call, and will be updated after the call if required.
- If your compression method **increases** the size of the data, the calling method will take care of storing a literal copy of the original rather than your compressed version (isn't that kind?).
- See the comments in OnRecognize on version management using **compressID**.
- See the source code example of an OnCompress handler for RLE compression in **RLECOMP.PAS**.

OnExpand Event

Declaration
procedure OnExpand(dest, source: TStream; sourcesize, destsize: Longint; compressID: String; var checksum: Longint);

Description
This event is called when expansion has been requested by using **coCustom** compression method.

Your handler must:
- Decompress from **source** to **dest**, using only the methods common to all stream types including write, read and seek.
- Respect the **sourcesize** setting, reading exactly that many bytes and no more. The **destsize** parameter is the original uncompressed file size, which some algorithms (e.g. LZH) require.
- Leave the stream positions at the end of the read/written areas.
- Calculate and set a **checksum** value (preset default 0) which the calling routine will verify against the checksum made when the data was compressed.

Notes
- The **compressID** string is passed to you in case you need it to decode version information. See OnRecognize.
- No header processing is required from your handler -- the **source** stream starts out positioned at the first byte of compressed data.
- See the source code example of an OnExpand handler for RLE compression in **RLECOMP.PAS**.

OnRecognize Event

Declaration
procedure OnRecognize(compressID: String; var recognized: Boolean);

Description
This event is called when a custom **compressID** has been found in the archive header.

Your handler must:
- "Recognize" **compressID** by setting **recognized** to **true** if it is valid (preset default is false).

Notes
- ID values **NON**, **RLE** and **LZH** are reserved.
- By requiring you to recognize the three-character compression ID, this event is allowing for the possibility that a TCompress archive has been compressed with one custom handler, and may now be attempting decompression with another.
- You may choose to recognize multiple IDs, reflecting either differing compression methods or (more likely) different versions of your original method.
- Depending on your application, it is **very desirable** to continue to support an old compression format, even if you later supersede it with a new version. Thus, you may end up recognizing ids such as **FRA**, **FRB** and **FRC**.

OnShowProgress Event

Declaration
procedure OnShowProgress(var PercentageDone: Longint);

Description
This event is called every time 8 kilobytes of data is read during a compression or expansion event. Use it to display a progress meter or provide for other user interaction during a long compression operation. See the example in COMPDEMO.

The **var** parameter allows an "abort" signal to be sent back to the compression routine. If you set **PercentageDone** to -1, the compression will stop as if the file had ended at that moment. Note that any work already done will be retained, e.g. a truncated compressed file will have been created and will require cleaning up.

Note: The effect of an abort when compressing blobs (CDBMemo/CDBIMage) and during expansion is unpredictable.

TCompress versions prior to V2.5 used to call Application.ProcessMessages just after this event was called. This has been removed in order to lessen the unit dependencies of the Compress unit. As a result, if you want to make your application more responsive and "Windows-friendly" during compression, you should place an Application.ProcessMessages statement in your handler for this event.

Other Products

We recommend the following quality components for Delphi/C++ Builder developers:

 TSegCompress by Ken McClain, http://www.mindspring.com/~kcmcclain/index.htm

TSegCompress is a "wrapper" component for TCompress. It lets you split TCompress archives into multiple chunks. For example, use TSegCompress to break an archive into 1.44MB chunks for backing up to floppy disk, and then to splice them all back together.

TCompLHA LHarc/LHA Archive Manager, http://www.spis.co.nz/compLHA.htm

Whereas TCompress is crafted to handle a wide range of general-purpose compression tasks, TCompLHA is specifically tailored to create and manage archives compatible with the popular freeware LHArc and LHA utilities (these archives are also compatible with other utilities like WinZip). TCompLHA can also read TCompress multi-file archives. An exceptionally easy interface gives you maximum results with almost no development effort, but powerful event hooks are provided to give you full control should you need it. Support is also included for encryption, segmented (disk-spanning) archives and easy self-extracting archives.

Webhub Dynamic Web Automation Framework, http://www.href.com/

If you're creating any kind of Web site CGI, database access, automation or Web page generation application for Win95 or WinNT, this extensive, proven, high-performance, superbly-supported component set should be your starting point. Don't waste a moment with anything else!

 DBExtender, http://www.gfi-mbh.com/index.htm

Forget Infopower or Orpheus -- this high-function navigator bar replacement gives you easy access to incremental searches, filtering, find/find-next and table view and edit for *any* table in your project -- all with one remarkably easy-to-use component.

THtmlViewer, http://www.pbear.com/

Drop this on your project and you'll be able to load and view HTML files with ease -- supports HTML 2.0

and most HTML 3.x features (including client-side image maps). As used in our SPIS Webview Offline Browser (below).

Other SPIS Products to check out:

Sorry, these aren't components, but we thought you might like a very brief run-down:

v SPIS Webview Offline Browser, http://www.spis.co.nz/webcentr/webview.htm

Webview is a small (under 800K) low-cost offline Web browser for Windows 3.x, Win95 or WinNT. Ideal for distributing sample web sites, product catalogues and demonstrations. Supports forms, external programs, scripted demos, editor, compressed pages, tables, inline images (GIF,JPG,PNG,BMP), image-maps, and image cache.

v SPIS Webwatch Site Monitor, http://www.spis.co.nz/webcentr/webwatch.htm

WinNT/Win95 automatic Web site monitor and problem notifier. Configurable to regularly check for connection, server, CGI or page problems on any number of sites. If problems are found, WebWatch can sound alarms, send email/faxmail or run a pager program, even run CGIs on the target server to fix the problem. Also features as an external link tester, connection timer and CGI endurance tester.

Properties

TCompress:
CheckSpaceBeforeExpand New in 3.05
CompressedPercentage (run time)
CompressionTime (run time)
ExceptionOnFileError
Key
MakeDirectories
Regname
RegNumber
TargetPath

TCDBMemo/TCDBImage/TCDBRichText
CField
CompressedField (protected)
CompressionMethod
CompressSource
DataField
DataSource

TCBlobfield:
CompressionMethod
CompressSource

TCBlobstream:
Blobstream

PutFileHeader Method

Declaration (protected method)
procedure PutFileHeader(Stream: Tstream; var Fhdr: TCompressedFileHeader; const Filename: String);

Description
This method sets **Fhdr.FileNameLength** and then writes **FHdr** and **Filename** to the current position in **Stream**.

Recognize Method

Declaration
function Recognize(const cID: String): TCompressionMethod

Description
This function simply returns the correct compression method based on the 3-character string in **cID**. An exception will occur if it is not recognized either as a built-in compression method, or a supported custom format (as recognized by an OnRecognize event handler).

Use this method if you are directly manipulating an archive and need to obtain the right mode to call DoExpand after reading the **ComMethodID** from the archive header.

Note
You must convert **ComMethodID** from a **array** of char into a string before calling this method.

RegName Property

Applies to
TCompress component.

Declaration
property RegName:String;

Description
Once you have registered your copy of TCompress, this property should be set to your registered name, e.g. **Peter Hyde**.

In conjunction with the RegNumber property, this will eliminate the occasional reminder dialog.

RegNumber Property

Applies to
TCompress component.

Declaration
property RegNumber: Longint;

Description
Once you have registered your copy of TCompress, this property should be set to your personal registration number, e.g. **12345**.

In conjunction with the RegName property, this will eliminate the occasional reminder dialog.

Registration

TCompress 3.5 Registration and License fee (one developer): $NZ90 (approx. $US65).
Please inquire about multi-developer or site licenses.
No additional payment or royalty is required when you distribute applications made with TCompress.

Full source code of the Compress unit (TCompress component) is also available for an additional $NZ59 (approx. $US40). The Compctrl unit source (TCDBImage, TCDBMemo and TCDBRich components) is $NZ49 (approx. $US34).

On registering, you will receive:
- Your personal Registration Number, which will eliminate the reminder dialog
- The latest version, if updated
- Rapid email assistance with any conundrums you encounter when using TCompress
- Our thanks for supporting quality products for Delphi & C++ Builder developers

We accept registrations via Mastercard, Visa or bank check in your currency to equivalent value at prevailing exchange rates.

Upgrade prices for registered users of earlier versions:

Upgrade from V3.0 or later: no charge

Registration upgrade from V2.5 or earlier: $NZ49 (approx. $US34)
If you bought source units for earlier versions, the upgrade fee above includes a source upgrade for each unit you own.

For proof of registration, please supply your existing RegName and RegNumber. For proof of source code ownership, please supply the first four source lines of each unit you own.

Crossgrade price for TCompLHA users: $NZ59 (approx. $US40)
(for proof of registration, please supply your TCompLHA RegName and RegNumber)

To register or upgrade to TCompress 3.5, print this form or copy it into an email message:

Full name: _____

Company name (if applicable): _____

Postal Address: _____

_____

_____

Email address: _____

Fax: _____ Tel: _____

How and where did you first find out about TCompress?

___ Web search engine: _____

___ Web site: _____

___ Power Tools Catalog (in the box with your compiler software)

___ Magazine advertisement: _____

___ Other: _____

With which product(s) will you use TCompress?

___ Delphi 1.0 ___ Delphi 2.0 ___ Delphi 3.0 ___ C++ Builder

Order for (tick):

____ TCompress v3.5 registered version @ $NZ90

____ Compress unit source @ $NZ59

____ Compctrl unit source @ $NZ49

____ Registration upgrade (from v2.5 or earlier to v3.5) @ $NZ49

____ Registration crossgrade for TCompLHA users @ $NZ59

Payment method (tick) ____ check (enclosed)

____ credit card (details below)

Card type: ____ Mastercard ____ Visa

Expiry Date: _____

Card number: _____

Cardholder's name (if different from above): _____

Send via: _____ Airmail ($NZ10 shipping charge applies)

_____ Email (Basic MIME)

_____ Email (UUENCODED)

NB: Email is not 100% secure! If you include credit card details with your order, put the order in a zip file (or TCompress archive) attached to your message, not within the main message. Better yet, send by fax or mail.

Send your order to:
South Pacific Information Services Ltd
PO Box 19-760, Christchurch, **New Zealand**
Fax: +64-3-384-5138
Email: software@spis.co.nz
Web page: http://www.spis.co.nz/compress.htm

Also see: http://www.mindspring.com/~kmcclain/index.htm to obtain and download Ken McClain's

TSegCompress. This add-on component to TCompress allows you to create segmented archives (e.g. for saving one archive on several floppies).

ScanCompressedFile Method

Declaration
procedure ScanCompressedFile(const arcfile: String; var finfo: TStringList);

Description
This method scans the **arcfile** multi-file archive and returns a list of the archive's files in **finfo**.

Also sets each **object** element in **finfo** to an instance of the TCompressedFileInfo object -- use the FreeFileList method to free all these when you've finished using the list.

See COMPDEMO for an example of the use of this method.

ScanCompressedStream Method

Declaration
procedure ScanCompressedStream(compressedStream: TStream; var finfo: TStringList);

Description
This method is an exact analogy of the ScanCompressedFile method except that the scan is performed from the **dest** stream. 1Don't overlook the notes in ScanCompressedFile about the file information objects allocated during the scan which must subsequently be freed with a FreeFileList call.

See CompressFilesToStream for more information about how streams may be used for flexible multi-file archive storage.

SetHeader Method

Declaration (protected method)
procedure SetHeader(var hdr: TCompressHeader; const cID: String; aType: TCompressArchiveType; size: Longint);

Description
This method initializes the archive header in **hdr**, based on the data in cID (a 3-character compression ID), **aType** and **size**.

It is called by CompressFilesToStream and Compress.

TCBlobStream Object

Unit
Compctrls

Description
This component is equivalent to Delphi's TBlobstream object, with a full range of matching properties and methods.

However, it has several subtle operational differences:

The **Create** method requires a TCBlobfield or derivative as its first parameter, rather than the TBlobfield required by TBlobstream's **Create**. The second parameter is the mode (**bmRead** or **bmWrite**, as required).

When created for reading a TCBlobfield, a TCBlobstream will immediately read into memory the entire contents of the field, automatically decompressing it if required. All subsequent read/write operations access the decompressed data in memory.

When created for writing a field, the TCBlobstream stores all data written to it in memory until it is freed, at which point the data is compressed (if required) and written to the database.

In **bmRead** mode, the Size property returns the actual (decompressed) size of the field data. In **bmWrite** mode, it will be the larger of the field's **DataSize** property, or however many bytes have already been written.

If you need to directly access the underlying (possibly compressed) data in the database, use the read-only Blobstream property which provides a regular TBlobstream handle to the field data. If you do this, you should **not** use the regular TCBlobstream methods and properties at the same time. Refer to the source example in **ARC2BLOB.PAS**.

TCBlobfield Class

TCMemofield **See also:** TMemofield
TCGraphicfield **See also:** TGraphicfield

Unit
Compctrls

Description
This class and its two derivatives are themselves derived from Delphi's TBlobfield component.

In effect, TCBlobfield instances supplant regular TBlobfields when you are performing database access with compression.

Normally, an appropriate TCBlobfield will be created for you automatically if you are using a TCDBMemo or TCDBImage component. However, you can create a TCBlobfield in code to access and compress any blob field. For example, you might do this for a field containing sound data. Refer to the source example in **BLOBDEMO.PAS (BLOB.DPR)**.

To support compression, TCBlobfield and its derivatives have two additional properties: CompressionMethod and CompressSource.

Database access (and compression/expansion) operations on TCBlobfields are performed by creating and using a TCBlobstream object. See **BLOBDEMO.PAS** for an example of this.

TCDBImage Component

Properties Tasks **See also:** TDBImage, TImage

Unit
Compctrl

Description
A TCDBImage component provides the same capabilities as Delphi's TDBImage, except that it can automatically read and write compressed images from the database.

The data compression features will not be enabled unless you link the TCDBImage to a TCompress component by setting the CompressSource property.

You should also set the CompressionMethod property, as the default compression is none (coNone).

The DataField and DataSource properties are the same as those for a TDBImage, apart from one important design-time difference which is detailed in the DataSource property information.

The CField run-time property provides access to the underlying TCGraphicField which provides the database access and compression.

See Using TCDBImage and the COMPDEMO application for more information and examples of use.

TCDBMemo Component

Properties Tasks **See also:** TDBMemo, TMemo

Unit
Compctrl

Description
A TCDBMemo component provides the same capabilities as Delphi's TDBMemo, except that it can automatically read and write compressed text from the database.

The data compression features will not be enabled unless you link the TCDBMemo to a TCompress component by setting the CompressSource property.

You should also set the CompressionMethod property, as the default compression is none (coNone).

The DataField and DataSource properties are the same as those for a TDBMemo, apart from one important design-time difference which is detailed in the DataSource property information.

The CField run-time property provides access to the underlying TCMemoField which provides the database access and compression.

See Using TCDBMemo and the COMPDEMO application for more information and examples of use.

TCDBRichText Component

Properties Tasks **See also:** TDBRichText, TRichText

v New in V3.0 (available in Delphi 3.0 only)

Unit
Compctrl

Description
A TCDBRichText component provides the same capabilities as Delphi's TDBRichText, except that it can automatically read and write compressed text from the database.

Its TCompress-specific properties are identical in name and effect to those of TCDBMemo. To avoid needless repetition, the Task topic above is that of TCDBMemo.

TCGraphicfield Class

See TCBlobfield class.

TCMemofield Class

See TCBlobfield class.

TCProcessMode Type

Unit
Compress

Declaration
```
TCProcessMode = ( cmCompress,          compress operation
                  cmExpand,            expand operation
                  cmDelete);           file delete operation
```

Description
These constants are used in the OnCheckFile event to tell your handler what operation is in progress so you can amend your user interface or filename handling as required.

TCompress V3.5

Delphi Choice

File and Database Compression for Delphi & C++ Builder

Fax: +64-3-384-5138 Email: software@spis.co.nz
Web: http://www.spis.co.nz/compress.htm

Key features:
v Delphi 1, 2, 3 and C++ Builder all supported
v Multi-file compressed archives
v Database BLOB (memo, image, binary) compression
v In-memory compression using streams
v Self-extracting EXEs and file encryption
v Easy CompressString and ExpandString
v Event hooks for customizable user interaction
v Built-in RLE (Run-Length Encoding) and LZH (Lempel-Ziv-Huffman) compression
v Add new custom compression routines at any time

Contents:

We encourage you to report any difficulties or limitations you may encounter so that we can consider improvements for everyone.

TCompress Component

Properties Methods Events Types Exceptions Tasks

Unit
Compress

Description
A TCompress component lets you manage a wide range of data compression tasks. As well as managing multi-file archives and providing a range of file and stream compression capabilities, it is essential for using the TCDBMemo and TCDBImage components.

While it has two forms of built-in compression and default handling for most situations, the events provided by this component mean you can provide your own user interface, and even your own specialist data compression routines (e.g. for fractal compression).

The RegName and RegNumber properties provide for registration of the component so that you will no longer see the occasional reminder dialog.

Two run time properties, CompressedPercentage and CompressionTime, both provide information on the performance of the **last** compression or expansion operation.

The TargetPath, MakeDirectories and ExceptionOnFileError properties increase the control you have over multi-file archives, and the Key property allows you to protect compressed items or individual files in an archive.

See Using TCompress for more information on TCompress tasks. Also refer to the numerous examples in the COMPDEMO application source and other example files provided.

TCompressArchiveType

Unit
Compress

Declaration
```
TCompressArchiveType =
                        ( caSingle,          single item archive (no file headers)
                          caMulti );         multi-file archive (file header for each)
```

Description
These constants are used in the archive header of each archive to note what kind of archive it is.

TCompressHeader Type

Unit
Compress

Declaration
```
TCompressHeader =
  Record
    ComId: array[0..4] of char;                    Always SPIS + chr(26)
    ComMethodId: array[0..2] of char;              RLE, LZH, NON, CUS/user-defined
    Fullsize: Longint;                             excl. header, not used for caMulti
    ArchiveType: TCompressArchiveType;             caSingle, caMulti
    Checksum: Longint;                             checksum value, not used caMulti
    Locked: Longint;                               1 if file has been locked (V3.05 or earlier), 2 if
encrypted (V3.5 or later), otherwise 0
 end;
```

Description
Every compressed archive (whether single-item or multi-file) starts with a copy of this header.

If the header is missing, data is read and treated as if it was not compressed. For example, the Compress method will store the original data with no header if the compressed data size was not smaller than the original.

If it is a multi-file archive, this header is immediately followed by the first file header, followed in turn by the compressed copy of that file. Note that the **Locked** flag is unimportant for multi-file archives, but may be required if you plan to call DoExpand directly.

Note: In common with TCompressedFileHeader, this structure is compiled with {$ALIGN OFF} under Delphi 2.0 to ensure data compatibility with the Delphi 1.0 version.

TCompressedFileHeader Type

Unit
Compress

Declaration
```
TCompressedFileHeader =
  Record
    FilenameLength: Smallint;          filename itself immediately follows this header
    Datetime: Longint;                 original file date and time
    Attributes: Smallint;              original file attributes
    Fullsize: Longint;                 original file size
    CompressedSize: Longint;           size of compressed copy of the file
    CompressedMode: TCompressionMethod;    how compressed
    Checksum: Longint;                 checksum value of original file, 0 if coNone
    Locked: Longint;                   1 if file has been locked (V3.05 or earlier), 2 if encrypted (V3.5
or later), otherwise 0
 end;
```

Description
In a multi-file archive, the archive header is followed by 0 or more repeats of this header, each followed by its filename and then its compressed file data.

The filename is max 255 characters long, and is always stored with **no** drive letter (although a path may be stored if desired).

Note that **compressedMode** can vary from file to file in a single archive.

Refer to source code examples which use this structure in **ARC2MEM.PAS** or **ARC2BLOB.PAS**.

Note: In common with TCompressHeader, this structure is compiled with {$ALIGN OFF} under Delphi 2.0 to ensure data compatibility with the Delphi 1.0 version.

TCompressedFileInfo Type

Unit
Compress

Declaration

```
TCompressedFileInfo = Class(TObject)
 public
    Datetime: Longint;                          original file date and time
    Fullsize: Longint;                          original file size
    Attributes: Smallint;                       original file attributes
    CompressedSize: Longint;                    compressed file size
    CompressedMode:    TCompressionMethod;      how compressed
    Checksum: Longint;                          original file data checksum
    Position: Longint;                          start of File header in the archive
    Locked: Boolean;                            True if this file was locked or encrypted, otherwise False
end;
```

Description
An instance of this object is created and assigned to each element of the TStringList which is filled by the ScanCompressedFile or ScanCompressedStream method.

You can use it to determine and/or display information such as the original file's creation date and attributes, the compression percentage, or whether or not it was locked (which is important if you plan to call the DoExpand method directly).

Warning: While instances of this object are created automatically, it is **up to you** to free them when you have finished with the TStringList. The easiest way to do this is to call the FreeFileList method. For an example of this, see the code in the **FormDestroy** method of COMPDEMO.

TCompressionMethod Type

Unit
Compress

Declaration
TCompressionMethod = (coNone, no data compression -- just store
coRLE, compress with RLE
coLZH, compress with LZH1
coCustom compress with Custom method
coLZH5); compress with LZH5 (30-50% faster, 10-20% better than LZH1)

Description
These constants are used in calling various TCompress methods, and also to set the CompressionMethod property of the TCDBMemo and TCDBImage components.

Only use coCustom if you have provided suitable handlers for the OnCompress, OnExpand and OnRecognize events.

TargetPath Property

Applies to
TCompress component.

Declaration
property TargetPath: String;

Description
This property provides you with fine control of the filepath stored in a multi-file archive and of the destination directory when the archive is expanded.

Note: if you have an OnCheckFile event, that will be called *after* any TargetPath-related modifications have been applied, so you have a chance to amend them further if need be.

Examples:
Assume TargetPath is **c:\mydir** in all cases.

Compress1.CompressFile('MyArchive.arc',Filepath,coLZH);

Filepath parameter	**Actual Filepath stored in the archive**
c:\mydir\Myfile.txt	Myfile.txt -- the TargetPath portion is trimmed from the stored path
c:\mydir\subdir\myfile.txt	subdir\myfile.txt
c:\otherfile.txt	\otherfile.txt (a non-matching path is kept intact, except for the drive letter)
c:\MYDIR\this.txt	this.txt (the TargetPath comparison is *not* case sensitive in TCompress 3.0)

Compress1.ExpandFile(FilePath,'MyArchive.arc');

Filepath parameter	**Where the file ends up**
MYFILE.TXT	c:\mydir\myfile.txt (note that case is not important in TCompress 3.0)
subdir\myfile.txt	c:\mydir\subdir\myfile.txt
\otherfile.txt	\otherfile.txt (on the current drive -- absolute paths override TargetPath handling)
otherfile.txt	none -- "otherfile.txt" will not be found in the archive, resulting in the request being silently ignored

Note: on expansion, directories and subdirectories will be created if necessary, provided MakeDirectories is True.

See Filename Handling in Compressed Archives for more information.

Troubleshooting TCompress 3.5

Checklist of common problems (also see the entries in FAQ.TXT):

- If you get a run time exception, see exceptions.
- If you are compiling a Delphi 3.0 project and get "DLL not found" message, either turn OFF the Project|Options|Packages|Build With Runtime Packages option, or remember to put all necessary DLLs (including COMP*.DPL) into the Windows Path
- If you see a **Duplicate field name** error when using TCDBMemo, TCDBImage or TCDBRichText, see the special note in the DataSource property information.
- If you have trouble expanding files you stored in an archive, see Filename Handling In Compressed Archives.
- If you get GPFs when running Delphi 1.0 programs which use LoadCompressedResource or LoadExpandedResource methods, see the notes in Differences -- Delphi 1.0, 2.0 and 3.0
- If you are getting a "Unit version mismatch error", there are several possibilities to check:

a) Are you using the correct (Delphi 1.0 vs. later version) units?
b) If you have the Compress unit source, but not the Compctrl source, have you amended the Interface section of the unit? If so, you'll need to buy and compile the Compctrl source as well.
c) Under Delphi 1.02, are the Compress & Compctrl units (source and/or DCUs) in the same directory as your project? If so, try moving them to another directory.
d) Have you installed or made any alterations to basic VCL units such as DB, DBTABLES or DBCTRLS? If so, you'll need the Compctrl source.

- Database blob compression is known or reported to work with BDE-supported databases, Interbase and MS SQL server (for these latter cases, set the TCBlobfield's Size property to 1). Please advise of your experiences with other databases.
- If you are experiencing problems with blob compression in projects using data modules, try placing the Compress component on the data module containing the table whose field(s) are being compressed, rather than on the main form.

Don't forget to check FAQ.TXT!

Any major updates to TCompress will be issued to BBS, Web and ftp sites worldwide -- you should check the date and time of their COMPRESS.ZIP against your own (all file times in this version are 3.50). You can also check these sites:
Web pages: http://www.spis.co.nz/compress.htm
Ftp site: ftp.sonic.net in /pub/users/ann/dcustore/compress.zip

The Web site will always contain the latest version information, new components, FAQs, tips and tricks and anything else we and our users have discovered in our perennial search for better, easier compression.

If you suspect there is a problem with any part of TCompress please contact us with the following information:

- Your version of Delphi or C++ Builder (and any patches applied)
- Your version of Windows
- Your version of these components (see the help window title above, and file dates and times)
- What you were doing, and what went wrong (be as precise as possible, and **do** check whether the error can be replicated after a reboot).
- If possible, a copy of the data being compressed/expanded, and enough source code from your application to allow us to independently repeat the problem.

Send the above information to South Pacific Information Services Ltd via any of the methods listed at the very end of the registration section.

Types

TCompress Types:
TCompressionMethod
TCProcessMode
TCompressedFileInfo
TCompressArchiveType
TCompressHeader
TCompressedFileHeader
'TCOMPRESS' resource type

TCDBImage/TCDBMemo Classes:
TCBlobField
TCBlobStream

Using TCDBImage

TCDBImage Reference TDBImage Reference

Purpose
Use the TCDBImage component to compress the pictures stored in Image fields in your database.

CDBImage offers exactly the same features as Delphi's TDBImage, except that you can link it to a TCompress component to automatically compress/expand the data as it is accessed.

Until the CompressSource and CompressionMethod properties are specified, TCDBImage will behave just like a regular TDBImage. (Even after they are specified, you will still be able to access **uncompressed** as well as compressed data with this component).

Setting Up Your Component
- Start by dropping a TCompress component on your form
- Select the TCDBImage component and set its CompressSource property to the new TCompress component
- Select the desired CompressionMethod
- Specify a DataSource and DataField for your component

Special Note
- The field you specify should **not** already be in the fields list of the TTable or TQuery to which the Datasource is linked. If it is, remove it.

Using Your Component
- Once your component is set up, it will **automatically** compress data written to its field. Existing records will be untouched unless the field is updated (e.g. amended or cut-and-pasted over itself).
- If data compression does not make a field smaller, the uncompressed contents of the field are kept rather than the "compressed" form

Compatibility Limitations
- Fields compressed with TCDBImage will naturally **not** be directly understandable to external applications (including ReportSmith).
- Operations which work at the Borland Database Engine level may not find meaningful data if they expect to find an uncompressed image.
- Thus, the "compressed BLOB" solution is strictly for Delphi apps until such time that Borland rewrite the BDE to provide true, in-built compression. This is unlikely.

Using TCDBMemo

TCDBMemo Reference TDBMemo Reference

Purpose
Use the TCDBMemo component to compress the text data stored in database memo fields.

TCDBMemo offers exactly the same features as Delphi's TDBMemo, except that you can link it to a TCompress component to automatically compress/expand the data as it is accessed.

Until the CompressSource and CompressionMethod properties are specified, the TCDBMemo will behave just like a regular TDBMemo. (Even after they are specified, you will still be able to access **uncompressed** as well as compressed data with this component).

Setting Up Your Component
- Start by dropping a TCompress component on your form
- Select the TCDBMemo component and set its CompressSource property to the new TCompress component
- Select the desired CompressionMethod
- Specify a DataSource and DataField for your component

Special Note
- The field you specify should **not** already be in the fields list of the TTable or TQuery to which the Datasource is linked. If it is, remove it.

Using Your Component
- Once your component is set up, it will **automatically** compress data written to its field. Existing records will be untouched unless the field is updated (e.g. amended or cut-and-pasted over itself).
- If data compression does not make a field smaller, the uncompressed contents of the field are kept rather than the "compressed" form

Compatibility Limitations
- Fields compressed with TCDBMemo will naturally **not** be directly understandable to external applications (including ReportSmith).
- Operations which work at the Borland Database Engine level (e.g. queries looking for keywords) may not find meaningful data.
- Thus, the "compressed BLOB" solution is strictly for Delphi apps until such time that Borland rewrite the BDE to provide true, in-built compression. This is unlikely.

Using TCompress

TCompress Reference

Purpose

Use the TCompress component to compress or expand data from files, resources, memory or database blobs into other files, memory or database blobs.

As supplied, TCompress supports LZH and RLE compression. You can also add custom compression routines (such as LZW) using the OnCompress, OnExpand and OnRecognize Events.

The OnCheckFile Event allows you to manage the user interaction involved in working with multi-file archives, although the TargetPath and MakeDirectories properties may lessen the need to program your own handling.

Tasks

- To compress, view and expand files in a multi-file archive, use the CompressFiles, ExpandFiles, DeleteFiles and ScanCompressedFile methods.
- To compress files to a stream (e.g. TMemoryStream or TBlobStream), use the CompressFilesToStream, ExpandFilesFromStream and ScanCompressedStream methods.
- To compress streamed (non-file) data to a multi-file archive, use CompressStreamToArchive and ExpandStreamFromArchive
- To compress a single item from one stream to another with an archive header, use Compress and Expand.
- To protect (or to read protected) compressed data or files, set the Key property
- To compress from one stream to another directly (no header) use DoCompress and DoExpand.
- To provide your own user interface for multi-file archive handling, or to vary the filenames/paths stored in an archive, write a handler for the OnCheckFile event and/or use the TargetPath and MakeDirectories properties.
- To compress and expand strings in memory, use CompressString and ExpandString
- To provide custom compression routines, write handlers for the OnCompress, OnExpand and OnRecognize events.
- To create and manage self-extracting EXEs or compressed resources, use LoadCompressedResource or LoadExpandedResource **or** see the notes in ExpandFilesFromStream which discuss an even easier approach which doesn't involve resources.

What's New in TCompress 3.5?

Hint: Keep an eye out for the v New in V3.5 notice throughout the help file.

New in TCompress 3.5:

v LZH5 compression support has been added -- significantly faster and better than LZH1
v Easy in-memory CompressString and ExpandString utility methods
v Changes to certain methods to permit easy creation of self-extracting EXEs
v Encryption has replaced the earlier "bicycle lock" protection provided by the Key property
v BDE dependencies in the Compress unit have been entirely removed (i.e. no more **CompOnly** unit)
v Easy $DEFINE enabling/disabling of compression and expansion methods to save memory (if you order the Compress unit source when you register)
v TruncateFile and TruncateStream protected methods have been abolished -- internal changes mean they are no longer required

New in TCompress 3.01 to 3.05:

v **const** added to almost all string parameters to methods
v GetAllFilesInDir and GetMatchingFiles now clear the file list before filling it, and no longer lowercase their results
v New CheckSpaceBeforeExpand property (for disabling space checking on >2GB networked drives)
v Percentage calculation improved for file sizes > 20MB
v Compression error for files > 20MB when OnShowProgress event was hooked is eliminated
v ExpandStreamFromArchive **filename** comparisons are no longer case sensitive

New in TCompress 3.0:

v GetAllFilesInDirectory and GetMatchingFiles methods make directory and file management easier
v TCDBRichText component added in Delphi 3.0 version
v Three new methods: CompressStreamTo Archive, ExpandStreamFromArchive and CompressStreamToArchiveStream make it very easy to compress arbitrary data directly to/from an archive *without* using files
v C++ Builder and Delphi 3.0 support

Upward Compatibility Issues with TCompress 2.5:

Three changes *might* have an impact on your existing programs:

1. TCompress 3.0 no longer forces stored filenames to lower case, but instead does case-free comparisons for all filenames (and for the Targetpath setting). See Filename Handling in Compressed Archives for more information.
2. The FreeFileList method no longer *frees* the TStringList passed to it -- it simply frees any TCompressedFileInfo objects it points to, and then *clears* the list. This implementation makes it easier to clear the list between calls to ScanCompressedFile/ScanCompressedStream, which is desirable (see the CompDemo source).
3. Earlier versions of TCDBImage placed the object's Dataset in edit state any time its Datasource's edit state changed. This resulted in unnecessary locks being set and maintained. V3.0 does not do this -- the main effect being that if you are using LoadFromFile, you should put the Dataset in edit mode first (see CompDemo for examples of this).