Zip Plugin 2.0

Liest und schreibt Zip-Archive mit Hollywood

Andreas Falkenhahn

Inhaltsverzeichnis

1 Allgemeine Informationen

1.1 Einführung

Mit diesem Plugin können Hollywood-Skripte Zip-Archive lesen und schreiben. Es verwendet die neuen Plug-In-Schnittstellen für Datei- und Verzeichnisadapter von Hollywood, mit denen Sie Zip-Archive durchlaufen können, als wären sie normale Verzeichnisse. Auf Dateien in Zip-Archiven kann auch zugegriffen werden, als wäre das Zip-Archiv ein normales Verzeichnis. Es ist nicht notwendig, eine in einem Zip-Archiv gespeicherte Datei in eine temporäre Datei zu entpacken, bevor sie geöffnet werden kann. Die Adapter-Plugin-Schnittstelle von Hollywood ermöglicht das direkte lesen und schreiben vom Zip-Archiv in den jeweiligen Datei-Handler.

Darüber hinaus bietet zip.hwp eine Reihe von Befehlen zum Lesen, Ändern und Schreiben von Zip-Archiven. Neue Zip-Archive können erstellt werden, vorhandene Zip-Archive können geöffnet und geändert werden. Es gibt eine Vielzahl von Befehlen, mit denen Sie Attribute von Dateien, die in Zip-Archiven gespeichert sind, lesen, ändern und schreiben können. Darüber hinaus unterstützt zip.hwp das Lesen und Speichern passwortgeschützter Dateien mit starker AES-128-, AES-192- und AES-256-Verschlüsselung.

Ab Version 2.0 unterstützt zip.hwp auch einige neue Funktionen von Hollywood 10, wie die Möglichkeit, Zip-Archive wie Dateisysteme zu behandeln, was bedeutet, dass Sie Hollywood-Befehle wie DeleteFile() oder Rename() auch direkt für Dateien oder Verzeichnisse in Zip-Archive verwenden können.

1.2 Lizenzvereinbarung

GEGENTEILIGES GESCHRIEBEN STEHT STELLEN DER AUTOR UND/ODER DRITTE DAS PROGRAMM "SO WIE ES IST" ZUR VERFÜGUNG, OHNE IRGENDEINE GARANTIE, WEDER DIREKT NOCH INDIREKT. DIES BEINHALTET, IST ABER NICHT DARAUF BESCHRÄNKT, VERKÄUFLICHKEIT UND EIGNUNG FÜR EINEN BESTIMMTEN VERWENDUNGSZWECK. DAS VOLLSTÄNDIGE RISIKO DER QUALITÄT UND AUSFÜHRBARKEIT DES PROGRAMMS LIEGT BEIM ANWENDER. SOLLTE SICH DAS PROGRAMM ALS DEFEKT HERAUSSTELLEN, LIEGEN ALLE KOSTEN FÜR SERVICE, INSTANDSETZUNG ODER NACHBESSERUNG BEIM ANWENDER.

KEIN COPYRIGHT-INHABER ODER DRITTER, DER DAS PROGRAMM WIE OBEN ERLAUBT WEITERVERKAUFT, KANN FÜR SCHÄDEN IRGENDWELCHER ART HAFTBAR GEMACHT WERDEN (DIES BEINHALTET, IST ABER NICHT BESCHRÄNKT AUF, DATENVERLUST INFOLGE UNFÄHIGKEIT DES PROGRAMMS, MIT ANDEREN PROGRAMMEN ZUSAMMENZUARBEITEN), SELBST WENN EIN SOLCHER INHABER ODER DRITTER AUF DIE MÖGLICHKEIT EINES SOLCHEN SCHADENS HINGEWIESEN WURDE, AUSSER ES BESTEHT EINE SCHRIFTLICHE EINWILLIGUNG ODER WIRD VOM GESETZ VERLANGT.

1.3 Voraussetzung

- Minimum: Hollywood 6.0 oder besser
- Unicode: Hollywood 7.0 oder besser wird empfohlen, weil zip.hwp intern UTF-8 für alle Zeichenketten verwendet und Hollywood-Versionen vor 7.0 sind nicht Unicodefähig item optimum: Hollywood 10.0 oder höher, da mehrere Funktionen von zip.hwp nur auf Hollywood 10 oder höher verfügbar sind

1.4 Installation

Die Installation von zip.hwp ist unkompliziert und einfach: Kopieren Sie einfach die Datei `zip.hwp` für die Plattform Ihrer Wahl in das Hollywood Plugin-Verzeichnis. Auf allen Systemen außer auf AmigaOS und kompatiblen Systemen müssen Plugins in einem Verzeichnis mit dem Namen `Plugins` gespeichert werden, das sich im selben Verzeichnis wie das Hauptprogramm von Hollywood befindet. Auf AmigaOS und kompatiblen Systemen müssen Plugins stattdessen in `LIBS: Hollywood` installiert werden. Unter macOS muss sich das Verzeichnis `Plugins` im Verzeichnis `Resources` des Programmpakets befinden, d.h. im Verzeichnis `HollywoodInterpreter.app/Contents/Resources`. Beachten Sie, dass `HollywoodInterpreter.app` im Programmpaket `Hollywood.app` selbst gespeichert ist, nämlich in `Hollywood.app/Contents/Resources`.

Unter Windows sollten Sie auch die Datei `zip.chm` in das Verzeichnis `Docs` Ihrer Hollywood-Installation kopieren. Dann können Sie die Online-Hilfe aufrufen, indem Sie F1 drücken, wenn sich der Cursor in der Hollywood-IDE über einem zip.hwp-Befehl befindet.

Kopieren Sie unter Linux und macOS das Verzeichnis `zip`, welches sich im Verzeichnis `Docs` des zip.hwp-Distributionsarchivs befindet, in das Verzeichnis `Docs` Ihrer Hollywood-Installation. Beachten Sie, dass sich das Verzeichnis `Docs` unter macOS im Programmpaket `Hollywood.app` befindet, d.h. in `Hollywood.app/Contents/Resources/Docs`.

2 Über zip.hwp

2.1 Danksagungen

zip.hwp wurde von Andreas Falkenhahn geschrieben. Die Arbeit an diesem Projekt wurde Anfang 2014 als Demonstration der leistungsfähigen neuen Datei- und Verzeichnisadapter-Plugin-Schnittstelle von Hollywood 6.0 gestartet, die es Plugins ermöglichen, sich in Hollywoods Datei- und Verzeichnishandler einzuklinken. zip.hwp nutzt diese Funktion, indem Hollywood denkt, dass Zip-Archive nur Verzeichnisse sind. So ist es möglich, sie mit Hollywoods normalen Verzeichnisfunktionen zu durchlaufen oder Dateien in Zip-Archiven zu öffnen, ohne sie zuerst zu entpacken.

Ein besonderer Dank geht an Helmut Haake und Dominic Widmer für die Übersetzung des Handbuchs ins Deutsche. Fehler oder Verbesserungsvorschläge bzgl. des deutschen Handbuchs bitte an das Übersetzungsteam richten, welches unter handbuch@gmx.ch erreicht werden kann.

Wenn Sie mich kontaktieren möchten, können Sie entweder eine E-Mail an andreas@airsoftsoftwair.de senden oder das Kontaktformular auf http://www.hollywood-mal.com verwenden.

2.2 Häufig gestellte Fragen

In diesem Abschnitt werden einige häufig gestellte Fragen behandelt. Bitte lesen Sie sie zuerst, bevor Sie in einem Forum nachfragen, da Ihr Problem hier möglicherweise behandelt wurde.

F: Gibt es ein Hollywood-Forum, in dem ich mit anderen Benutzern in Kontakt treten kann?

A: Ja, bitte besuchen Sie die "Community" oder "Forum"-Sektion des offiziellen Hollywood-Portals unter http://www.hollywood-mal.com.

F: Wo kann ich um Hilfe bitten?

A: Es gibt ein lebhaftes englischsprachiges Forum auf http://forums.hollywood-mal.com. Sie können gerne Ihre Frage dort stellen. Ausserdem ist ein deutschsprachiges Forum vorhanden, welches Sie unter https://www.amiga-resistance.info/ erreichen können.

F: Ich habe einen Fehler gefunden.

A: Bitte informiere mich darüber in den speziellen Bereichen des Forums oder der Mailingliste.

2.3 Bekannte Probleme

Hier ist eine Liste von Dingen, die zip.hwp noch nicht unterstützt oder die auf irgendeine Weise verwirrend sein können:

- Ist noch offen (tpd)

2.4 Zukunft

Hier sind einige Dinge, die auf meiner Liste stehen:

– Ist noch offen (tpd)

Zögern Sie nicht, mich zu kontaktieren, wenn in zip.hwp eine bestimmte Funktion fehlt, die für Ihr Projekt wichtig ist.

2.5 Geschichte

Bitte schauen Sie in die auf englisch verfasste Datei `history.txt`. Hier finden Sie ein vollständiges Änderungsprotokoll von zip.hwp.

3 Benutzung

3.1 Verwendung des Plugins

Es gibt mehrere Möglichkeiten, dieses Plugin zu verwenden: Es gibt eine Low-Level-Bibliotheksschnittstelle, die spezielle Funktionen für den Umgang mit Zip-Archiven bereitstellt, z.B. zip.OpenArchive() zum Öffnen eines Archivs und zip.ExtractFile() zum Extrahieren einer Datei. Die Low-Level-Schnittstelle ist etwas schwieriger zu verwenden als die High-Level-Schnittstelle, bietet aber das größte Maß an Kontrolle über Zip-Archive und auch die beste Leistung.

Darüber hinaus verfügt zip.hwp über eine High-Level-Schnittstelle, die es Ihrem Skript ermöglicht, mit Zip-Archiven umzugehen, ohne neue APIs erlernen zu müssen. Stattdessen können Sie einfach die normalen Hollywood-Befehle verwenden, um auf Zip-Archive zuzugreifen. Es gibt zwei Möglichkeiten, die High-Level-Schnittstelle zu verwenden: Sie können das Plugin entweder global aktivieren, indem Sie den Tag `InstallAdapter` beim Aufruf von `@REQUIRE` auf `True` setzen. Fügen Sie dazu einfach die Präprozessor-Anweisung von oben in Ihr Skript ein:

```
@REQUIRE "zip", {InstallAdapter = True}
```

Wenn Sie die High-Level-Schnittstelle über `@REQUIRE` aktivieren, wird es global verfügbar und alle folgenden Befehle, die sich mit Dateien befassen, unterstützen das Öffnen von Dateien aus Zip-Archivquellen. Zum Beispiel könnten Sie dann folgende Zeile schreiben:

```
LoadBrush(1, "test.zip/testpicture.jpg")
```

Wenn Sie nur sehr wenige Dateien aus ZIP-Archivquellen öffnen müssen, können Sie die High-Level-Schnittstelle auch nicht global aktivieren, indem Sie den Tag `InstallAdapter` bei `@REQUIRE` weglassen und einfach den von den meisten Hollywood-Befehlen angebotenen Tag `Adapter` verwenden, um die Datei mit dem zip.hwp-Plugin zu öffnen. Hier ist ein Beispiel:

```
LoadBrush(1, "test.zip/testpicture.jpg", {Adapter = "zip"})
```

Mit dem Tag `Adapter` wird `LoadBrush()` angewiesen, die angegebene Datei mit dem angegebenen Adapter zu öffnen, der in unserem Fall `"zip"` ist. Mit dem Tag `Adapter` können Sie dieses Plugin also auch verwenden, ohne zuvor einen globalen Dateiadapter installiert zu haben.

Dasselbe gilt für Hollywood-Befehle, die mit Verzeichnissen arbeiten. Sobald die High-Level-Schnittstelle aktiviert wurde, indem `InstallAdapter` auf `True` gesetzt wird, ist es möglich, folgendes zu verwenden:

```
OpenDirectory(1, "test.zip")
```

Sie könnten dann alle Dateien und Verzeichnisse in `test.zip` durchlaufen. Wenn Sie den globalen Adapter für zip.hwp nicht aktiviert haben, verwenden Sie einfach den lokalen `Adapter` wie oben, z.B:

```
OpenDirectory(1, "test.zip", {Adapter = "zip"})
```

Weitere Informationen über die Arbeit mit Zip-Archiven als Verzeichnisse finden Sie im nächsten Kapitel.

3.2 Zip Archive als Verzeichnisse

Wenn Sie den Tag `InstallAdapter` auf `True` setzen, hängt sich das Zip-Plugin in Hollywoods Datei und Verzeichnis-Handler ein, damit Hollywood glaubt, dass Zip-Archive normale Verzeichnisse sind. Auf diese Weise können Sie alle Dateien und Verzeichnisse in einem Zip-Archiv mithilfe normaler Befehle aus der Dateisystembibliothek (DOS) von Hollywood durchlaufen.

Um beispielsweise alle Dateien und Verzeichnisse in einer Datei mit dem Namen `test.zip` zu durchlaufen, können Sie den folgenden Code verwenden:

```
OpenDirectory(1, "test.zip")
Local e = NextDirectoryEntry(1)
While e <> Nil
    DebugPrint(e.name)
    e = NextDirectoryEntry(1)
Wend
CloseDirectory(1)
```

Wenn Sie nicht vom Stammverzeichnis innerhalb von `test.zip` aus starten möchten, können Sie auch bequem von einem Unterverzeichnis aus starten, indem Sie einfach vorgeben, dass `test.zip` ein Verzeichnis ist. Z.B. um auf ein Unterverzeichnis mit dem Namen `files` in `test.zip` zuzugreifen, gehen Sie wie folgt vor:

```
OpenDirectory(1, "test.zip/files")
```

Schließlich ist es auch möglich, rekursiv alle Dateien und Verzeichnisse in einem Zip-Archiv zu durchlaufen. Hier ist eine Funktion, die das tut:

```
Function p_DumpZip(d$, idt)
    Local id = OpenDirectory(Nil, d$)
    Local e = NextDirectoryEntry(id)
    While e <> Nil
        If e.Type = #DOSTYPE_DIRECTORY
            DebugPrint(RepeatStr(" ", idt) .. "+", e.name)
            p_DumpZip(FullPath(d$, e.name), idt + 4)
        Else
            DebugPrint(RepeatStr(" ", idt) .. "",e.name,e.size,e.time)
        EndIf
        e = NextDirectoryEntry(id)
    Wend
    CloseDirectory(id)
EndFunction
```

Um den Inhalt eines Zip-Archivs abzulegen, rufen Sie einfach folgende Funktion auf:

```
p_DumpZip("test.zip", 0)
```

Es wird dann einen schönen Baum des Inhalts des Zip-Archivs ausgeben.

3.3 Dateien entpacken

Da sich zip.hwp in den Datei-Handler von Hollywood einklinkt, wenn Sie die High-Level-Schnittstelle verwenden, ist das Entpacken von Dateien nur eine Frage der Verwendung des Hollywood-Befehls `CopyFile()` für die Datei, die Sie entpacken möchten. Um beispielsweise

eine Datei mit dem Namen `testpicture.jpg` aus `test.zip` zu entpacken, verwenden Sie einfach die folgende Zeile:

```
CopyFile("test.zip/testpicture.jpg", "outputdir")
```

Wenn das Zip-Archiv passwortgeschützt ist, können Sie das Passwort mithilfe der neuen Benutzer-Tags, die in Hollywood 10 eingeführt wurden, an zip.hwp übergeben. Wenn beispielsweise `test.zip` das Passwort ?123456? verwendet, können Sie dieses Passwort für zip.hwp so übergeben:

```
CopyFile("test.zip/pic.jpg", "out", {UserTags = {Password = "123456")})
```

Da `CopyFile()` auch ganze Verzeichnisse einschließlich aller Unterverzeichnisse kopieren kann und weil sich zip.hwp auch in den Datei-Handler von Hollywood einhakt, ist es sogar möglich, ein ganzes Archiv mit `CopyFile()` so zu entpacken:

```
CopyFile("test.zip", "outputdir")
```

Sie können Dateien auch direkt aus Zip-Archiven öffnen, da sich zip.hwp in den Datei-Handler von Hollywood einklinkt. Sie könnten dies beispielsweise tun, um alle Zeilen von `test.txt` auszugeben, die in `test.zip` gespeichert sind:

```
OpenFile(1, "test.zip/test.txt")
While Not Eof(1) Do DebugPrint(ReadLine(1))
CloseFile(1)
```

Alle Hollywood-Befehle, die sich mit Dateien befassen, unterstützen das Öffnen von Dateien aus Zip-Archiven, wenn der Dateiadapter von zip.hwp aktiviert wurde. Sie könnten also auch Bilder und andere Datendateien direkt aus einem solchen Zip-Archiv laden:

```
LoadBrush(1, "test.zip/test.jpg")
```

3.4 Archive ändern

Sie können Zip-Archive auch ändern, indem Sie einfach Befehle aus der Dateisystembibliothek (DOS) von Hollywood verwenden. Um beispielsweise die Datei `test.jpg` aus dem ZIP-Archiv `test.zip` zu löschen, können Sie einfach wie folgt vorgehen:

```
DeleteFile("test.zip/test.jpg", {Adapter = "zip"})
```

Beachten Sie, dass es zwingend erforderlich ist, den Tag `Adapter` an `DeleteFile()` zu übergeben, da zip.hwp keinen Dateisystemadapter installiert, selbst wenn der Tag `InstallAdapter` auf `True` gesetzt ist (siehe oben). Auf den Dateisystemadapter von Zip.hwp kann nur zugegriffen werden, indem er direkt an einen Hollywood-Befehl im Tag `Adapter` übergeben wird.

Es ist auch möglich, Dateien in Zip-Archiven mit dem Befehl `Rename()` von Hollywood umzubenennen. Das kann man so machen:

```
Rename("test.zip/oldname.txt", "newname.txt", {Adapter = "zip"})
```

Wie oben müssen Sie den Tag `Adapter` an `Rename()` übergeben, damit dies funktioniert.

Auf diese Weise können Sie Verzeichnisse in Zip-Archiven erstellen:

```
MakeDirectory("test.zip/a_new_dir", {Adapter = "zip"})
```

Vergessen Sie nicht, auch hier "zip" im Tag `Adapter` zu übergeben.

Auf diese Weise können neue Dateien in Zip-Archiven erstellt werden:

```
OpenFile(1, "test.zip/new_file", #MODE_WRITE)
```

```
WriteLine("Hello World!")
CloseFile(1)
```

Oder noch kürzer:

```
StringToFile("Hello World!", "test.zip/new_file")
```

Sie können Benutzer-Tags auch verwenden, um ein Passwort und eine Verschlüsselungsstufe anzugeben:

```
StringToFile("Hello World!", "test.zip/new_file", {UserTags =
      {Password = "123456", Encryption = #ZIP_EM_AES_128}})
```

Beachten Sie, dass beim Schreiben von Dateien in ZIP-Archive vorhandene Dateien im ZIP-Archiv nicht gelöscht werden, sondern die neuen Dateien an das ZIP-Archiv angehängt werden. Beachten Sie jedoch, dass wenn die Datei, die in ein Zip-Archiv geschrieben werden soll, vorhanden ist, diese automatisch überschrieben wird. Seien Sie also vorsichtig.

Mit `CopyFile()` ist es sogar möglich, Dateien in Zip-Archive zu kopieren. Sie könnten zum Beispiel auch so etwas tun:

```
CopyFile("testfile", "test.zip", {Adapter = "zip"})
```

Der obige Code speichert die Datei `testfile` im ZIP-Archiv `test.zip`.

Schließlich können Sie die Attribute von Dateien in ZIP-Archiven mit dem Befehl `SetFileAttributes()` von Hollywood ändern oder Dateien mit dem Befehl `MoveFile()` von Hollywood in ZIP-Archive hinein und aus diesen verschieben. Übergeben Sie einfach den Namen der Datei, die Sie ändern möchten, sowie `"zip"` im Tag `Adapter` und es wird funktionieren. Es ist wirklich praktisch und leistungsstark!

3.5 Zip-Archiv Grundlagen

Wenn Sie die Low-Level-Schnittstelle von zip.hwp verwenden, müssen Sie zunächst einige Grundlagen über die Struktur von Zip-Archiven erlernen.

Zip-Archive sind nur eine Sammlung von Dateien, die in Indizes von 0 bis zur Anzahl der Einträge im Zip-Archiv minus 1 gespeichert werden. Es ist nicht notwendig, Verzeichnisse als einzelne Einträge zu speichern. Stattdessen können sie auch als Teil eines Dateinamens gespeichert werden. Z.B. wenn eine Datei im Zip-Archiv als `a/b/c/test.txt` gespeichert ist, werden die Verzeichnisse a, b und c implizit als existierend deklariert, obwohl sie keine eigenen Einträge in der Datei im Zip-Archiv haben, sondern nur als Teil einer Datei existieren.

Natürlich können Verzeichnisse auch als einzelne Einträge anstatt als Teil eines Dateinamens gespeichert werden. In diesem Fall werden sie einfach als Dateien mit einer Größe von 0 Bytes gespeichert, wobei der Dateiname mit einem Schrägstrich endet, der anzeigt, dass der Eintrag ein Verzeichnis ist. Da es in Zip-Archiven keinen eindeutigen Verzeichniseintragstyp gibt, können alle Befehle in diesem Plugin, die sich mit Dateien befassen, auch auf Verzeichnisse innerhalb des Zip-Archivs zugreifen. Beachten Sie also, dass ein Befehl wie zip.RenameFile() auch zum Umbenennen von Verzeichnissen verwendet werden und zip.DeleteFile() auch Verzeichnisse löschen kann. Innerhalb eines Zip-Archivs sind Verzeichnisse und Dateien ziemlich identisch, nur dass bei Verzeichnissen ihr Dateiname in einem Schrägstrich endet, um anzuzeigen, dass es sich nicht um eine Datei handelt.

3.6 Zip-Archive erstellen

Mit den Befehlen zip.OpenArchive(), zip.AddFile() und zip.CloseArchive() können Sie neue Zip-Archive erstellen. Der folgende Code zeigt, wie ein neues Zip-Archiv mit dem Namen `test.zip` erstellt wird, das die Datei `testpicture.jpg` enthält:

```
zip.OpenArchive(1, "test.zip", #MODE_WRITE)
zip.AddFile(1, "testpicture.jpg")
zip.CloseArchive(1)
```

Beachten Sie, dass zip.AddFile() die Datei nicht sofort komprimiert und in das Archiv schreibt. Stattdessen werden Dateien zuerst gesammelt und erst komprimiert und in das Archiv geschrieben, wenn Sie zip.CloseArchive() aufrufen. Aus diesem Grund kann das Schließen eines Archivs einige Zeit in Anspruch nehmen. Darum gibt es auch die Möglichkeit, eine Callback-Funktion zu übergeben, die von zip.CloseArchive() von Zeit zu Zeit aufgerufen wird, so dass Sie eine Statusleiste oder etwas anderes aktualisieren können.

3.7 Dateien verknüpfen

Beachten Sie, dass alle in den Präprozessor-Anweisungen deklarierten Dateien automatisch mit Ihrem Applet oder Ihrem Programm verknüpft werden, wenn sich Hollywood im Kompilierungsmodus befindet. Wenn Sie also etwas wie die folgende Zeile verwenden, wird nicht nur `testpicture.jpg` sondern das gesamte Zip-Archiv `test.zip` mit Ihrem Applet oder Programm verknüpft:

```
@BRUSH 1, "test.zip/testpicture.jpg"
```

Wenn Sie das nicht möchten, können Sie den optionalen Tag `Link` auf `False` setzen. Wenn `Link` auf `False` gesetzt ist, wird Hollywood die angegebene Datei nicht mit Ihrem Applet oder Programm verknüpfen. Dies bedeutet jedoch, dass Sie `test.zip` mit Ihrem Applet oder Programm verteilen müssen, damit die Daten daraus geladen werden können. So deaktivieren Sie die Verknüpfung:

```
@BRUSH 1, "test.zip/testpicture.jpg", {Link = False}
```

Wenn Sie so vorgehen, wird Hollywood die Datei niemals mit Ihrem Applet oder Programm verknüpfen. Stattdessen wird es immer aus der angegebenen Datei geladen.

4 Befehlsreferenz

4.1 zip.AddDirectory

BEZEICHNUNG

zip.AddDirectory – fügt ein Verzeichnis zum Zip-Archiv hinzu

ÜBERSICHT

```
idx = zip.AddDirectory(id, d$[, t])
```

BESCHREIBUNG

Dieser Befehl erstellt ein neues Verzeichnis mit dem Namen `d$` im Zip-Archiv mit der ID `id` und gibt den Index in `idx` zurück. Das Verzeichnis ist leer und Sie können Dateien mit dem Befehl zip.AddFile() hinzufügen.

Die folgenden Tags werden derzeit durch das optionale Tabellenargument erkannt:

`Encoding:`

Mit diesem Tag kann die Zeichensatzcodierung eingestellt werden, die beim Speichern des Verzeichnisnamens verwendet werden soll. Dies kann eine der folgenden speziellen Konstanten sein:

`#ZIP_FL_ENC_UTF_8:`

Verwendet die UTF-8-Codierung. Dies ist die Standardeinstellung.

`#ZIP_FL_ENC_CP437:`

Verwendet die Codepage 437-Codierung. Da dies die Standardcodierung unter MS-DOS war, war es auch die Standardcodierung des ursprünglichen Zip-Formats. Wenn Sie also maximale Kompatibilität benötigen, können Sie diese Codierung verwenden. Aber denken Sie daran, dass sie nur westliche Zeichen speichern kann.

(V1.2)

EINGABEN

`id`	Identifikator des zu verwendenden Zip-Archivs
`d$`	Name des Verzeichnisses, das im ZIP-Archiv erstellt werden soll
`table`	optional: Tabelle mit weiteren Optionen (siehe oben) (V1.2)

RÜCKGABEWERTE

`idx`	Index des neu hinzugefügten Verzeichnisses im ZIP-Archiv

4.2 zip.AddFile

BEZEICHNUNG

zip.AddFile – fügt eine Datei zum Zip-Archiv hinzu

ÜBERSICHT

```
idx = zip.AddFile(id, f$[, table])
```

BESCHREIBUNG

Dieser Befehl fügt die durch `f$` angegebene Datei dem in `id` angegebenen Zip-Archiv hinzu und gibt den Index der neu hinzugefügten Datei in `idx` zurück. Mit dem optionalen Tabellenargument `table` können Sie weitere Optionen angeben.

Die folgenden Tags werden derzeit vom optionalen Argument `table` erkannt:

`NewName:` Mit diesem Tag können Sie diese Datei unter einem neuen Namen im Zip-Archiv speichern. Wenn Sie die Datei in einem Unterverzeichnis im ZIP-Archiv speichern möchten, müssen Sie auch diesen Tag verwenden und den Namen des/der Unterverzeichnisse(s) in `NewName` angeben. Wenn `NewName` weggelassen wird, wird die Datei immer im Stammverzeichnis des Zip-Archivs gespeichert.

`Encryption:`

Mit diesem Tag können Sie die gewünschte Verschlüsselungsmethode für die Datei festlegen. Es kann auf eine der folgenden speziellen Konstanten eingestellt werden:

`#ZIP_EM_NONE:`
: Keine Verschlüsselung. Dies ist der Standardwert.

`#ZIP_EM_AES_128:`
: Winzip AES-128-Verschlüsselung.

`#ZIP_EM_AES_192:`
: Winzip AES-192-Verschlüsselung.

`#ZIP_EM_AES_256:`
: Winzip AES-256-Verschlüsselung.

Wenn Sie den Tag `Encryption` angeben, müssen Sie auch ein Passwort im Tag `Password` angeben (siehe unten), das zum Entschlüsseln der Datei benötigt wird. Wenn Sie den Tag `Password` nicht verwenden, wird das mit zip.SetDefaultPassword() festgelegte Standard-Passwort verwendet.

`Password:`

Wenn der Tag `Encryption` auf einen anderen Wert als `#ZIP_EM_NONE` gesetzt wurde (siehe oben), kann dieser Tag auf ein Passwort gesetzt werden, welches zum Schutz der Datei verwendet werden soll. Wenn Sie diesen Tag nicht angeben, wird das mit zip.SetDefaultPassword() festgelegte Standard-Passwort verwendet.

`Compression:`

Mit diesem Tag kann die gewünschte Komprimierungsmethode für die Datei festgelegt werden. Die folgenden Komprimierungsmethoden werden derzeit unterstützt:

`#ZIP_CM_DEFAULT:`
: Dies ist die Standardeinstellung. Derzeit identisch mit `#ZIP_CM_DEFLATE`.

`#ZIP_CM_STORE:`
Die Datei unkomprimiert speichern.

`#ZIP_CM_BZIP2:`
Die Datei mit dem bzip2-Algorithmus komprimieren.

`#ZIP_CM_DEFLATE:`
Die Datei mit dem zlib-Algorithmus und den Standardoptionen komprimieren.

Beachten Sie, dass nur `#ZIP_CM_DEFLATE` und `#ZIP_CM_STORE` als universell unterstützt gelten.

Wenn Sie diesen Tag angeben, können Sie auch den Tag `CompressionFlags` übergeben, um die Komprimierungsstufe festzulegen (siehe unten).

`CompressionFlags:`
Mit diesem Tag kann die Komprimierungsstufe definiert werden. Sie geht von 1 für die schnellste Komprimierung bis 9 für die höchste Komprimierung. Sie können auch 0 übergeben, um die Standardeinstellungen des Komprimierers zu verwenden. Der Standardwert ist 0.

`Comment:` Mit diesem Tag kann die Datei dem Zip-Archiv mit einem angehängten Kommentar hinzugefügt werden.

`Time:` Mit diesem Tag kann der Datumsstempel der Datei geändert werden. Standardmäßig wird der Datumsstempel aus der in `f$` angegebenen Datei genommen. Wenn Sie der Datei einen anderen Datumsstempel zuweisen möchten, müssen Sie diesem Tag eine Zeichenkette im Standard-Hollywood-Datumsformat von `dd-mmm-yyyy hh:mm:ss` übergeben.

`Encoding:`
Mit diesem Tag kann die Zeichensatzcodierung eingestellt werden, die beim Speichern des Verzeichnisnamens verwendet werden soll. Dies kann eine der folgenden speziellen Konstanten sein:

`#ZIP_FL_ENC_UTF_8:`
Verwendet die UTF-8-Codierung. Dies ist die Standardeinstellung.

`#ZIP_FL_ENC_CP437:`
Verwendet die Codepage 437-Codierung. Da dies die Standardcodierung unter MS-DOS war, war es auch die Standardcodierung des ursprünglichen Zip-Formats. Wenn Sie also maximale Kompatibilität benötigen, können Sie diese Codierung verwenden. Aber denken Sie daran, dass sie nur westliche Zeichen speichern kann.

(V1.2)

Beachten Sie, dass dieser Befehl nicht sofort beginnt, die Datei zu komprimieren und in das Zip-Archiv zu schreiben. Stattdessen wird die Datei nur in eine interne Liste eingefügt und das Komprimieren und Schreiben erfolgt, sobald Sie zip.CloseArchive()

aufrufen. Das bedeutet, dass Sie sicherstellen müssen, dass die in `f$` angegebene Datei noch verfügbar ist, wenn Sie zip.CloseArchive() aufrufen. Beachten Sie deshalb, dass wenn Sie den Namen einer temporären Datei an `f$` übergeben, Sie diese temporäre Datei nicht löschen, bevor Sie zip.CloseArchive() aufrufen.

EINGABEN

`id`	Identifikator des zu verwendenden Zip-Archivs
`f$`	Pfad zu einer Datei, die dem ZIP-Archiv hinzugefügt werden soll
`table`	Optional: Tabelle mit weiteren Optionen (siehe oben)

RÜCKGABEWERTE

`idx`	Index der neu hinzugefügten Datei im ZIP-Archiv

4.3 zip.CloseArchive

BEZEICHNUNG

zip.CloseArchive – schliesst das Zip-Archiv

ÜBERSICHT

```
zip.CloseArchive(id[, discard, callback, userdata])
```

BESCHREIBUNG

Dieser Befehl schließt das in `id` angegebene Zip-Archiv. Beachten Sie, dass `zip.CloseArchive()` den Punkt angibt, an dem das Komprimieren und Schreiben der Daten tatsächlich erfolgt, wenn das Zip-Archiv zum Schreiben geöffnet wurde. Aus diesem Grund kann es einige Zeit dauern, bis dieser Befehl seine Arbeit beendet hat.

Wenn Sie alle Änderungen verwerfen möchten, die am ZIP-Archiv vorgenommen wurden, müssen Sie `True` im Parameter `discard` übergeben. In diesem Fall wird das ursprüngliche Zip-Archiv nicht geändert und alle Änderungen werden verworfen. Dies wird auch mit allen Zip-Archiven geschehen, die Sie mit zip.OpenArchive() öffnen, aber vergessen mit `zip.CloseArchive()` zu schließen. Änderungen werden nur dann in das Zip-Archiv geschrieben, wenn Sie explizit zip.CloseArchive() aufrufen, wobei `discard` `False` ist.

Wenn Sie den Fortschritt beim Komprimieren von Daten und beim Schreiben in das Zip-Archiv überwachen möchten, können Sie eine Callback-Funktion im dritten Parameter `callback` übergeben. Optional ist es auch möglich, Benutzerdaten anzugeben, die in ihrem vierten Argument `userdata` an die Callback-Funktion übergeben werden. Der Parameter `userdata` kann Werte beliebigen Typs annehmen: Zahlen, Zeichenketten, Tabellen und sogar Funktionen können als Benutzerdaten übergeben werden.

Die Status-Callback-Funktion empfängt ein einzelnes Tabellenelement, das die folgenden Felder enthält:

`Action:`	Enthält die Zeichenkette "CloseArchive".
`ID:`	Enthält den Identifikator des Zip-Archivs, an dem gerade gearbeitet wird.
`Progress:`	Enthält einen Wert zwischen 0 und 100, der angibt, wie viel Arbeit bereits erledigt wurde.

`UserData:`
Enthält den Wert, den Sie im Argument `userdata` übergeben haben.

Offensichtlich wird die Callback-Funktion niemals aufgerufen, wenn `discard` auf `True` gesetzt ist.

EINGABEN

`id`	Identifikator des zu schließenden Zip-Archivs
`discard`	Optional: `True` um alle Änderungen zu verwerfen, `False`, um alle Änderungen in das Zip-Archiv zu schreiben (voreingestellt ist `False`)
`callback`	Optional: Funktion, die von Zeit zu Zeit aufgefrufen wird
`userdata`	Optional: Benutzerspezifische Daten, die an die Callback-Funktion übergeben werden

4.4 zip.DeleteFile

BEZEICHNUNG
zip.DeleteFile – löscht eine Datei aus dem Zip-Archiv

ÜBERSICHT
`zip.DeleteFile(id, idx)`

BESCHREIBUNG
Dieser Befehl löscht die Datei im Index `idx` in dem von `id` angegebenen Zip-Archiv.

Beachten Sie, dass die Änderung am Zip-Archiv nicht sofort erfolgt, sondern bis zum Aufruf von zip.CloseArchive() verschoben wird.

Dieser Befehl kann auch für Verzeichnisse verwendet werden. Siehe Abschnitt 3.5 [Zip Archiv-Grundlagen], Seite 8, für Details.

EINGABEN

`id`	Identifikator des zu verwendenden Zip-Archivs
`idx`	Zu löschende Datei

4.5 zip.ExtractFile

BEZEICHNUNG
zip.ExtractFile – entpackt eine Datei aus dem Zip-Archiv

ÜBERSICHT
`zip.ExtractFile(id, idx, dst$[, table])`

BESCHREIBUNG
Dieser Befehl kann verwendet werden, um die Datei im Index `idx` innerhalb des von `id` angegebenen Zip-Archivs zu der von `dst$` angegebenen externen Datei zu entpacken.

Wenn `dst$` bereits existiert, wird sie überschrieben. Mit dem optionalen Tabellenargument `table` können Sie folgende Optionen für die Operation angeben:

`Password:`
: Wenn die zu entpackende Datei durch ein Passwort geschützt ist, müssen Sie dieses Passwort hier angeben. Wenn Sie diesen Tag nicht angeben, wird das Standard-Passwort verwendet, das mit dem zip.SetDefaultPassword() festgelegt wurde.

`Callback:`
: Mit diesem Tag können Sie eine Funktion übergeben, die von Zeit zu Zeit aufgerufen werden soll. Dies kann nützlich sein, wenn Sie eine Statusleiste oder etwas anzeigen möchten, während die Zip-Datei entpackt wird. Die Funktion erhält eine Tabelle als einziges Argument. In der Tabelle werden folgende Felder initialisiert:

  `Action:` Enthält die Zeichenkette `"ExtractFile"`.

  `ID:` Enthält den Identifikator des Zip-Archivs.

  `Progress:`
  : Enthält einen Wert zwischen 0 und 100, der angibt, wie viel Arbeit bereits erledigt wurde.

  `UserData:`
  : Enthält den Wert, den Sie im Argument `UserData` übergeben haben (siehe unten).

  Sie können auch Benutzerdaten, die an Ihren Callback-Funktion weitergeleitet werden, mit dem folgenden Tag übergeben.

`UserData:`
: Dieser Tag kann auf beliebige Daten gesetzt werden, die an die Callback-Funktion übergeben werden sollen, den Sie im Tag `Callback` übergeben haben. Wenn Sie diesen Tag ohne den Tag `Callback` angeben, wird er einfach ignoriert.

EINGABEN

`id`	Identifikator des zu verwendenden Zip-Archivs
`idx`	Index der zu entpackenden Datei
`dst$`	Gewünschte Zieldatei
`table`	Optional: Tabelle mit weiteren Parametern

4.6 zip.GetFileAtIndex

BEZEICHNUNG

zip.GetFileAtIndex – ermittelt den Namen der Datei im Index

ÜBERSICHT

```
name$ = zip.GetFileAtIndex(id, idx)
```

BESCHREIBUNG

Dieser Befehl gibt den Namen der Datei im Index `idx` in dem von `id` angegebenen Zip-Archiv zurück. Wenn der von diesem Befehl zurückgegebene Name mit einem Schrägstrich endet, handelt es sich um ein Verzeichnis, andernfalls um eine Datei. Siehe Abschnitt 3.5 [Zip-Archiv Grundlagen], Seite 8, für Details.

Um die Anzahl der Dateien in einem Zip-Archiv herauszufinden, können Sie `#ZIPATTRNUMENTRIES` mit dem Befehl `GetAttribute()` von Hollywood abfragen. Siehe Abschnitt 4.9 [zip.GetObjectType], Seite 19, für Details.

EINGABEN

`id` Identifikator des zu verwendenden Zip-Archivs

`idx` Index der Rückfrage (im Bereich von 0 bis Anzahl der Einträge minus 1)

RÜCKGABEWERTE

`name$` Name des Eintrags im Index

4.7 zip.GetFileAttributes

BEZEICHNUNG

zip.GetFileAttributes – ermittelt die Dateiattribute

ÜBERSICHT

```
t = zip.GetFileAttributes(id, idx)
```

BESCHREIBUNG

Dieser Befehl gibt Attribute der Datei an dem in `idx` angegebenen Index innerhalb des von `id` angegebenen Zip-Archivs zurück. `zip.GetFileAttributes()` gibt eine Tabelle mit folgenden Informationen über die Datei zurück:

`Size:` Die Größe der Datei in Byte oder 0 für Verzeichnisse.

`CompressedSize:`
Die komprimierte Größe der Datei in Byte oder 0 für Verzeichnisse.

`CRC32:` Die CRC32-Prüfsumme der Datei oder 0 für Verzeichnisse.

`Compression:`
Die für die Datei verwendete Komprimierungsmethode. Dies wird eine der folgenden speziellen Konstanten sein:

`#ZIP_CM_DEFAULT:`
Standardkomprimierung, derzeit identisch mit `#ZIP_CM_DEFLATE`.

`#ZIP_CM_STORE:`
Die Datei ist unkomprimiert.

`#ZIP_CM_BZIP2:`
Die Datei ist mit dem bzip2-Algorithmus komprimiert.

`#ZIP_CM_DEFLATE:`
Die Datei ist mit dem ZLIB-Algorithmus und den Standardoptionen kompromiert.

`Encryption:`
: Die für die Datei verwendete Verschlüsselungsmethode. Dies wird eine der folgenden speziellen Konstanten sein:

  `#ZIP_EM_NONE:`
  : Keine Verschlüsselung.

  `#ZIP_EM_AES_128:`
  : Winzip AES-128 Verschlüsselung.

  `#ZIP_EM_AES_192:`
  : Winzip AES-192 Verschlüsselung.

  `#ZIP_EM_AES_256:`
  : Winzip AES-256 Verschlüsselung.

`Time:`
: Der Datumsstempel für die Datei. Dies wird im Standard-Hollywood-Datumsformat von `dd-mmm-yyyy hh:mm:ss` sein.

EINGABEN

`id`
: Identifikator des zu verwendenden Zip-Archivs

`idx`
: Index der abzufragenden Datei

RÜCKGABEWERTE

`t`
: Tabelle mit Dateiattributen

4.8 zip.GetFileComment

BEZEICHNUNG

zip.GetFileComment – ermittelt den Dateikommentar

ÜBERSICHT

```
c$ = zip.GetFileComment(id[, idx])
```

BESCHREIBUNG

Dieser Befehl kann verwendet werden, um den Kommentar einer Datei innerhalb des von `id` angegebenen Zip-Archivs oder des gesamten Zip-Archivs zu ermitteln. Wenn der optionale Parameter `idx` angegeben wird, ruft `zip.GetFileComment()` den Kommentar der Datei aus diesem Index ab. Wenn der Parameter `idx` nicht angegeben oder auf -1 gesetzt ist, wird der Kommentar des Zip-Archivs selbst zurückgegeben.

Dieser Befehl kann auch auf Verzeichnisse angewendet werden. Siehe Abschnitt 3.5 [Zip-Archiv Grundlagen], Seite 8, für Details.

EINGABEN

`id`
: Identifikator des zu verwendenden Zip-Archivs

`idx`
: Optional: Index der Datei, deren Kommentar ermittelt werden soll; Wenn dies weggelassen oder auf -1 gesetzt wird, wird der Kommentar des gesamten Archivs zurückgegeben (voreingestellt ist -1).

RÜCKGABEWERTE

`c$` Dateikommentar oder leere Zeichenkette, wenn kein Kommentar vorhanden ist

4.9 zip.GetObjectType

BEZEICHNUNG

zip.GetObjectType – ruft den Zip-Archiv-Objekttyp ab

ÜBERSICHT

```
type = zip.GetObjectType()
```

BESCHREIBUNG

Dieser Befehl gibt den Objekttyp zurück, der von Zip-Archiven verwendet wird, die mit dem Befehl zip.OpenArchive() geöffnet wurden. Sie können diesen Objekttyp dann mit Befehlen aus der Objektbibliothek von Hollywood wie `GetAttribute()`, `SetObjectData()`, `GetObjectData()` usw. verwenden.

Insbesondere kann der Befehl `GetAttribute()` von Hollywood verwendet werden, um bestimmte Eigenschaften von mit zip.OpenArchive() geöffneten Zip-Archiven abzufragen. Die folgenden Attribute werden derzeit von `GetAttribute()` für Zip-Archive unterstützt:

`#ZIPATTRNUMENTRIES:`
Gibt die Anzahl der Einträge im ZIP-Archiv zurück.

EINGABEN

Keine

RÜCKGABEWERTE

`type` Interner Zip-Archivtyp zur Verwendung mit der Objektbibliothek von Hollywood

BEISPIEL

```
zip.OpenArchive(1, "test.zip")
ZIP_ARCHIVE = zip.GetObjectType()
numentries = GetAttribute(ZIP_ARCHIVE, 1, #ZIPATTRNUMENTRIES)
```

Der obige Code öffnet `test.zip` und fragt die Anzahl der Einträge im Archiv über `GetAttribute()` ab.

4.10 zip.LocateFile

BEZEICHNUNG

zip.LocateFile – sucht die Datei im ZIP-Archiv

ÜBERSICHT

```
idx = zip.LocateFile(id, name$[, table])
```

BESCHREIBUNG

Dieser Befehl sucht nach der in `name$` angegebene Datei innerhalb des von `id` angegebenen Zip-Archivs und gibt ihren Index zurück, wenn sie gefunden wird, andernfalls wird -1 zurückgegeben.

Mit dem optionalen Argument `table` können weitere Optionen angegeben werden. Folgende Tabellentags werden derzeit erkannt:

`NoCase:` Wenn dieser Tag auf `True` festgelegt ist, unterscheidet `zip.LocateFile()` nicht zwischen Groß- und Kleinbuchstaben. Dies macht die Suche langsamer. Der Standardwert ist `False`.

`NoDir:` Wenn dieser Tag auf `True` gesetzt ist, stimmt `zip.LocateFile()` nur mit dem Dateinamen überein. Ausserdem wird es auch auf `True` gesetzt, wenn sich die Datei in einem Unterverzeichnis des Archivs befindet. Der Standardwert ist `False`.

Dieser Befehl kann auch auf Verzeichnisse angewendet werden. Siehe Abschnitt 3.5 [Zip-Archiv Grundlagen], Seite 8, für Details.

EINGABEN

`id` Identifikator des zu verwendenden Zip-Archivs

`name$` Name der zu suchenden Datei

`table` Optional: Tabellenargument mit weiteren Optionen (siehe oben)

RÜCKGABEWERTE

`idx` Index der Datei innerhalb des Zip-Archivs oder -1, wenn es nicht gefunden wurde

4.11 zip.OpenArchive

BEZEICHNUNG

zip.OpenArchive – öffnet ein Zip-Archiv zum Lesen oder Schreiben

ÜBERSICHT

```
[id] = zip.OpenArchive(id, filename$[, mode])
```

BESCHREIBUNG

Dieser Befehl versucht, das durch `filename$` angegebene Zip-Archiv zu öffnen und weist ihr den Identifikator `id` zu. Wenn Sie `Nil` in `id` übergeben, wählt `zip.OpenArchive()` automatisch eine freie ID aus und gibt sie zurück. Wenn die Datei nicht existiert, schlägt dieser Befehl fehl, es sei denn, Sie verwenden das Argument `mode`, um ein Zip-Archiv zum Schreiben zu öffnen. In diesem Fall erstellt `zip.OpenArchive()` die Datei für Sie.

Die folgenden Modi werden derzeit unterstützt:

`#MODE_READ:`
Öffnet das Zip-Archiv zum Lesen. Dies ist der Standardmodus.

`#MODE_READWRITE:`
Öffnet das Zip-Archiv zum Lesen und Schreiben. Wenn das angegebene Zip-Archiv nicht existiert, wird es automatisch erstellt.

`#MODE_WRITE:`
Öffnet das Zip-Archiv zum Schreiben. Wenn das angegebene Zip-Archiv bereits existiert, wird es überschrieben.

Obwohl zip.hwp beim Beenden des Programms alle geöffneten Zip-Archive automatisch schließt, wird dringend empfohlen, ein geöffnetes Zip-Archiv nach Abschluss der Arbeit mit dem Befehl zip.CloseArchive() zu schliessen. Andernfalls verschwenden Sie Ressourcen. Falls Sie ein Zip-Archiv schreiben oder ändern, ist zip.CloseArchive() der Moment, bei dem die eigentliche Arbeit erledigt wird.

Beachten Sie, dass `zip.OpenArchive()` ein Standard-Hollywood-Objekt erstellt, das auch mit Befehlen aus Hollywoods Objektbibliothek wie `GetAttribute()`, `SetObjectData()`, `GetObjectData()` usw. verwendet werden kann. Siehe Abschnitt 4.9 [zip.GetObjectType], Seite 19, für Details.

EINGABEN

`id` Identifikator der Datei oder `Nil` für die automatische ID-Auswahl

`filename$`
Name der zu öffnenden Datei

`mode` Modus, um die Datei zu öffnen; kann `#MODE_READ`, `#MODE_WRITE` oder `#MODE_READWRITE` sein (der Standardwert ist `#MODE_READ`)

RÜCKGABEWERTE

`id` Optional: Identifikator der Datei; wird nur zurückgegeben, wenn Sie `Nil` als Argument in `id` übergeben (siehe oben)

4.12 zip.RenameFile

BEZEICHNUNG

zip.RenameFile – benennt die Datei im Zip-Archiv um

ÜBERSICHT

```
zip.RenameFile(id, idx, newname$[, t])
```

BESCHREIBUNG

Dieser Befehl benennt die Datei beim Index `idx` im von `id` angegebenen Zip-Archiv in den Namen um, der im Argument `newname$` übergeben wurde.

Wenn Sie einen Verzeichniseintrag umbenennen müssen, muss `neuname$` mit einem Schrägstrich enden. Siehe Abschnitt 3.5 [Zip-Archiv Grundlagen], Seite 8, für Details.

Beachten Sie, dass die Änderung am Zip-Archiv nicht sofort erfolgt, sondern verschoben wird, bis Sie zip.CloseArchive() aufrufen.

Die folgenden Tags werden derzeit durch das optionale Tabellenargument erkannt:

`Encoding:`
Mit diesem Tag kann die Zeichensatzcodierung eingestellt werden, die beim Speichern des Verzeichnisnamens verwendet werden soll. Dies kann eine der folgenden speziellen Konstanten sein:

`#ZIP_FL_ENC_UTF_8:`
Verwendet die UTF-8-Codierung. Dies ist die Standardeinstellung.

`#ZIP_FL_ENC_CP437:`
: Verwendet die Codepage 437-Codierung. Da dies die Standardcodierung unter MS-DOS war, war es auch die Standardcodierung des ursprünglichen Zip-Formats. Wenn Sie also maximale Kompatibilität benötigen, können Sie diese Codierung verwenden. Aber denken Sie daran, dass sie nur westliche Zeichen speichern kann.

(V1.2)

EINGABEN

`id`	Identifikator des zu verwendenden Zip-Archivs
`idx`	Datei zum Umbenennen
`newname$`	Neuer Name für die Datei
`table`	optional: Tabelle mit weiteren Optionen (siehe oben) (V1.2)

4.13 zip.SetDefaultPassword

BEZEICHNUNG

zip.SetDefaultPassword – legt das Standard-Passwort fest

ÜBERSICHT

```
zip.SetDefaultPassword(id, pwd$)
```

BESCHREIBUNG

Dieser Befehl kann verwendet werden, um das Standard-Passwort zum Verschlüsseln und Entschlüsseln von Dateien festzulegen und wird benutzt, wenn kein anderes Passwort angegeben wird. Sie müssen den Identifikator eines Zip-Archivs in `id` und ein Passwort in `pwd$` übergeben. Wenn Sie eine leere Zeichenkette in `pwd$` übergeben, wird das Standard-Passwort nicht festgelegt.

Befehle, die das Standard-Passwort verwenden, wenn kein anderes Passwort explizit angegeben wird, sind zip.AddFile(), zip.ExtractFile(), und zip.SetFileEncryption().

EINGABEN

`id`	Identifikator des zu verwendenden Zip-Archivs
`pwd$`	Neues Standard-Passwort oder leere Zeichenkette, um das Standard-Passwort aufzuheben

4.14 zip.SetFileComment

BEZEICHNUNG

zip.SetFileComment – legt den Dateikommentar fest

ÜBERSICHT

```
zip.SetFileComment(id[, c$, idx])
```

BESCHREIBUNG

Mit diesem Befehl kann der Kommentar für eine Datei innerhalb des von `id` angegebenen Zip-Archivs oder für das gesamte Zip-Archiv gesetzt werden. Wenn der optionale Parameter `idx` angegeben wird, wird der durch `c$` angegebene Kommentar für die Datei dieses Indexes festgelegt. Wird der optionale Parameter `idx` weggelassen oder auf -1 gesetzt, wird der durch `c$` angegebene Kommentar für das gesamte Zip-Archiv gesetzt.

Sie können den Kommentar einer Datei oder des gesamten Archivs auch entfernen, indem Sie das Argument `c$` weglassen oder auf eine leere Zeichenkette setzen.

Beachten Sie, dass die Änderung am Zip-Archiv nicht sofort erfolgt, sondern verschoben wird, bis Sie zip.CloseArchive() aufrufen.

Dieser Befehl kann auch auf Verzeichnisse angewendet werden. Siehe Abschnitt 3.5 [Zip-Archive Grundlagen], Seite 8, für Details.

EINGABEN

`id`	Identifikator des zu verwendenden Zip-Archivs
`c$`	Optional: legt den Kommentar fest oder der Kommentar wird mit einer leeren Zeichenkette entfernt
`idx`	Optional: Index der Datei, deren Kommentar gesetzt werden soll; Wenn dies weggelassen oder auf -1 gesetzt wird, wird der Kommentar für das gesamte Archiv gesetzt (standardmäßig -1)

4.15 zip.SetFileCompression

BEZEICHNUNG

zip.SetFileCompression – legt die Dateikomprimierung fest

ÜBERSICHT

```
zip.SetFileCompression(id, idx, method[, flags])
```

BESCHREIBUNG

Dieser Befehl legt die Komprimierungsmethode für die Datei im Index `idx` in dem in `id` angegebenen Zip-Archiv auf die in `method` angegebene Komprimierungsmethode fest. Das optionale Argument `flags` kann verwendet werden, um die Komprimierungsstufe zu definieren, wobei 1 die schnellste und 9 die höchste Komprimierung ist. Zulässige Werte sind 1-9 oder 0, um die Standardeinstellungen der Komprimierung zu verwenden.

Der Parameter `method` muss eine der folgenden Konstanten sein:

`#ZIP_CM_DEFAULT:`
: Dies ist die Standardeinstellung. Derzeit das gleiche wie `#ZIP_CM_DEFLATE`.

`#ZIP_CM_STORE:`
: Die Datei wird unkomprimiert gespeichert.

`#ZIP_CM_BZIP2:`
: Die Datei wird mit dem bzip2-Algorithmus komprimiert.

`#ZIP_CM_DEFLATE:`
: Die Datei wird mit dem ZLIB-Algorithmus und Standardoptionen komprimiert.

Beachten Sie, dass nur `#ZIP_CM_DEFLATE` und `#ZIP_CM_STORE` als universelle unterstützt gelten.

Beachten Sie auch, dass die Änderung des Zip-Archivs nicht sofort erfolgt, sondern verschoben wird, bis Sie zip.CloseArchive() aufrufen.

EINGABEN

`id`	Identifikator des zu verwendenden Zip-Archivs
`idx`	Index der Datei, deren Komprimierung eingestellt werden soll
`method`	Gewünschte Komprimierungsmethode (siehe oben)
`flags`	Optional: Gewünschter Komprimierungsgrad von 1 (am schnellsten) bis 9 (am höchsten) oder 0 für die Standardeinstellung der Komprimierung (standardmäßig 0)

4.16 zip.SetFileEncryption

BEZEICHNUNG

zip.SetFileEncryption – setzt die Dateiverschlüsselung

ÜBERSICHT

```
zip.SetFileEncryption(id, idx, method[, pwd$])
```

BESCHREIBUNG

Dieser Befehl legt die Verschlüsselungsmethode für die Datei im Index `idx` in dem von `id` angegebenen Zip-Archiv fest. Die gewünschte Verschlüsselungsmethode muss im Parameter `method` übergeben werden. Optional können Sie im Argument `pwd$` ein Passwort angeben. Wenn das Argument `pwd$` weggelassen oder auf eine leere Zeichenkette gesetzt wird, wird das mit zip.SetDefaultPassword() festgelegte Standard-Passwort verwendet.

Der Parameter `method` muss eine der folgenden Konstanten sein:

`#ZIP_EM_NONE:`
: Keine Verschlüsselung.

`#ZIP_EM_AES_128:`
: Winzip AES-128 Verschlüsselung.

`#ZIP_EM_AES_192:`
: Winzip AES-192 Verschlüsselung.

`#ZIP_EM_AES_256:`
: Winzip AES-256 Verschlüsselung.

Beachten Sie, dass die Änderung am Zip-Archiv nicht sofort erfolgt, sondern verschoben wird, bis Sie zip.CloseArchive() aufrufen.

EINGABEN

`id`	Identifikator des zu verwendenden Zip-Archivs
`idx`	Index der Datei, deren Verschlüsselung eingestellt werden soll
`method`	Gewünschte Verschlüsselungsmethode (siehe oben)

`pwd$`	Optional: Gewünschtes Passwort für die Datei oder leere Zeichenkette, um das Standard-Passwort zu verwenden (voreingestellt ist eine leere Zeichenkette)

4.17 zip.SetFileTime

BEZEICHNUNG

zip.SetFileTime – legt den Zeitstempel der Datei fest

ÜBERSICHT

```
zip.SetFileTime(id, idx, time$)
```

BESCHREIBUNG

Dieser Befehl legt den Datumsstempel für die Datei im Index `idx` in dem von `id` angegebenen Zip-Archiv fest. Der Datumsstempel muss im Standard-Hollywood-Datumsformat übergeben werden: `dd-mmm-yyyy hh:mm:ss`

Beachten Sie, dass die Änderung am Zip-Archiv nicht sofort erfolgt, sondern verschoben wird, bis Sie zip.CloseArchive() aufrufen.

EINGABEN

`id`	Identifikator des zu verwendenden Zip-Archivs
`idx`	Index der Datei, deren Datumsstempel geändert werden soll
`time$`	Gewünschter Datumstempel

Anhang A Lizenzen

A.1 LibZip Lizenz

A.2 AES-Verschlüsselungsunterstützung

Index