PKZIP®/SecureZIP® for i5/OS®

User's Guide

SZIU-V9R0001

PKWARE Inc.

PKWARE Inc.
648 N Plankinton Avenue, Suite 220
Milwaukee, WI 53203

Main office: 888-4PKWARE (888-475-9273)

Sales: 937-847-2374 (888-4PKWARE / 888-475-9273)
Sales - Email: pksales@pkware.com
Support: 937-847-2687
Support - http://www.pkware.com/support/system-i
Fax: 414-289-9789
Web Site: http://www.pkware.com

9.0 Edition (2006)

PKZIP for i5/OS, *SecureZIP for i5/OS*, *SecureZIP for z/OS*®, *PKZIP for z/OS, PKZIP for UNIX*, and *PKZIP for Windows* are just a few of the many members in the PKZIP family. PKWARE, Inc. would like to thank all the individuals and companies—including our customers, resellers, distributors, and technology partners—who have helped make PKZIP the industry standard for trusted ZIP solutions. PKZIP enables our customers to efficiently and securely transmit and store information across systems of all sizes, ranging from desktops to mainframes.

This edition applies to the following PKWARE, Inc. licensed programs:

PKZIP for i5/OS (Version 9, Release 0, 2006)
SecureZIP for i5/OS (Version 9, Release 0, 2006)
SecureZIP Partner for i5/OS (Version 9, Release 0, 2006)

Contents

Preface

This manual covers both ***PKZIP for i5/OS*** and ***SecureZIP for i5/OS***.

PKZIP for i5/OS provides powerful, easy-to-use data compression on the AS/400, iSeries and i5. ***PKZIP for i5/OS*** Enterprise Edition also includes support for passphrase-based decryption of encrypted files powered by trusted RSA® BSAFE. Files created by ***PKZIP for i5/OS*** use the widely-adopted ZIP format and can be accessed on all major platforms throughout the enterprise—from iSeries to PC.

SecureZIP for i5/OS provides powerful, easy-to-use data compression and data protection on the AS/400, iSeries and i5. ***SecureZIP for i5/OS*** delivers high-performance data compression and protects data with digital signatures and trusted RSA BSAFE encryption, either passphrase- or certificate-based, with key lengths of up to 256 bits. Like ***PKZIP for i5/OS***, ***SecureZIP for i5/OS*** uses the widely-adopted ZIP format and creates files that can be accessed on all major platforms throughout the enterprise.

This manual also covers ***SecureZIP Partner***. ***SecureZIP Partner*** is a special version of SecureZIP that is available through the PKWARE PartnerLink program. The PKWARE PartnerLink program provides a straightforward, secure way for an organization to exchange sensitive information with outside partners who perhaps do not have SecureZIP.

SecureZIP Partner for i5/OS differs from the full ***SecureZIP for i5/OS*** in that it only extracts archives *from*, and only creates and encrypts archives *for*, a PartnerLink sponsor. Contact PKWARE for more information on ***SecureZIP Partner***.

Notices

Note: The only changes to this manual from the previous 2006 version are updated patent information and the URL for PKWARE support. All significant improvements to *PKZIP* and *SecureZIP for i5/OS* are included in the current version. Visit www.pkware.com for more information.

To better align our products with IBM naming conventions and to support the future development of new products on the IBM System z and System i platforms, PKWARE has changed the names of its large-platform products to reference the compatible IBM operating systems instead of specific platforms. For example, with version 9.0, the former ***SecureZIP for iSeries*** is now called ***SecureZIP for i5/OS***.

About this Manual

This manual provides the information needed to utilize ***PKZIP*** and ***SecureZIP for i5/OS*** in an operational environment. It is assumed that people using this manual have a good understanding of (Control Language) CL and dataset processing. Note that the contents of this manual apply to the following operating systems:

- OS/400 V5R1M0 and above
- iSeries
- i5/OS

This manual is intended for persons using both ***PKZIP*** and ***SecureZIP for i5/OS***. The manual assumes that the reader has a good understanding of CL and file processing.

Conventions Used in this Manual

Throughout this manual, the following conventions are used:

PKZIP**i* (bold-italicized) refers to both ***PKZIP for i5/OS and ***SecureZIP for i5/OS*** products.

Program, screen display and printout examples may show either SecureZIP or PKZIP constructs. Unless specifically denoted within SecureZIP feature sections, the samples also apply to PKZIP.

If a line has (*SecureZIP*), it applies only to ***SecureZIP for i5/OS.***

If a section is flagged at the beginning with the phrase, *Requires SecureZIP*, formatted as shown below, that section applies only to ***SecureZIP for i5/OS***:

Requires SecureZIP

The use of the Courier font indicates text that may be found in control language (CL), parameter controls, or printed output.

The use of *italics* indicates a value that must be substituted by the user, for example, a dataset name. It may also be used to indicate the title of an associated manual or the title of a chapter within this manual.

Bullets (•) indicate items (or instructions) in a list.

The use of <angle brackets> in a command definition indicates a mandatory parameter.

The use of [square brackets] in a command definition indicates an optional parameter.

A vertical bar (|) in a command definition is used to separate mutually exclusive parameter options or modifiers.

Related Publications

PKZIP/SecureZIP for i5/OS product manuals include:

- ***PKZIP/SecureZIP for i5/OS* System Administrator's Guide** - Provides detailed information to assist the system administrator with the installation and administrative requirements necessary to use ***SecureZIP for i5/OS*** in an operational environment.
- ***PKZIP/SecureZIP for i5/OS* User's Guide** - Provides detailed information on the product set in OS/400 and i5/OS operating environments. Also provided is a general introduction to data compression, SECZIP specific data compression, and an overview on how to use ***PKZIP and SecureZIP for i5/OS***, SECZIP control cards, and parameters.
- ***PKZIP/SecureZIP for i5/OS* Messages and Codes** - This provides information on the messages and codes that are displayed on the consoles, printed outputs, and associated terminals.

Related IBM Publications

IBM manuals relating to the ***PKZIP*^{i}** product include:

- **System Messages:** This manual documents messages issued by the i5/OS operating system. The descriptions explain why the component issued the message, provide the actions of the operating system, and suggest responses by the applications programmer, system programmer, and/or operator.
- **OS/400 CL Programming (SC41-5721):** This manual provides a wide-range discussion of iSeries Advanced Series programming topics, including: Control language programming, iSeries Advanced Series programming concepts, objects and libraries, and message handling.
- **OS/400 CL Reference (SC41-5722 thru SC41-5726):** This manual may be used in the iSeries Information Center to find information on the following CL reference topics: OS/400 commands, OS/400 objects, command description format, command parts, command syntax, about syntax diagrams, CL character sets and values, object naming rules, expressions in CL commands, and command definition statements.
- **Integrated File System Introduction (SC41-5711):** This book provides an overview of the integrated file system includes these topics:
  - What is the integrated file system?
  - Why might you want to use it
  - Integrated file system concepts and terminology
  - Interfaces you can use to interact with the integrated file system
  - APIs and techniques you can use to create programs that interact with the integrated file system
  - Characteristics of individual file systems
- **File Management (SC41-5710):** This manual describes the data management portion of the Operating System/400 licensed program. Data management provides applications with access to input and output file data that is external to the application. There are several types of these input and output files, and each file type has its own characteristics. In addition, all of the file types share a common set of characteristics.

- **DDS Reference (RBAF-P000-00):** This manual contains detailed instructions for coding the data description specifications (DDS) for files that can be described externally. These files are the physical, logical, display, printer, and intersystem communications functions, hereafter referred to as ICF files.

Related Information on the Internet

PKWARE, Inc.

www.pkware.com

FTP site

Product manuals - ftp://bigiron.pkware.com/pub/manuals/i5OS

Product downloads - ftp://bigiron.pkware.com/pub/products

- ***PKZIP for i5/OS***
  ftp://bigiron.pkware.com/pub/products/pkzip/i5OS
- ***SecureZIP for i5/OS***
  ftp://bigiron.pkware.com/pub/products/securezip/i5OS
- ***SecureZIP Partner for i5/OS***
  ftp://bigiron.pkware.com/pub/products/partnerlink/i5OS

National Institutes of Standards and Technology

Computer Security Resource Center - http://csrc.ncsl.nist.gov

Information on the AES development - http://csrc.nist.gov/encryption/aes

Information on Key Management -
http://csrc.nist.gov/CryptoToolkit/tkkeymgmt.html

RSA BSAFE® Content Library – http://www.rsasecurity.com/content_library.asp

Release Summary

New Features 9.0

New features in ***PKZIP for i5/OS*** and ***SecureZIP for i5/OS*** Release 9.0 include:

- 1Step2Tape Feature – The ability to create an archive directly to tape with out any disk files
- ***SecureZIP*** now supports multiple contingency keys with the use of inlist for a type code
- Expanded maximum passphrase length from 200 to 260 alphanumeric characters
- Ability to support Adopt Authority for archive files in libraries (added with V9.0.1)

New Commands 9.0

There are no new commands for version 9.0.

Command Changes & Defaults 9.0

The following commands have changed since version 8.1. Each command and parameter listed below should be reviewed before activating ***PKZIP[i]*** 9.0:

PKZIP

TYPARCHFL()	*TAPF is added as an option to specify that archive is to be written directly to tape. *XDB added as an option to specify that archive is to be created or updated exclusively in the QSYS library file system.
MSGTYPE()	An option is added that controls the amount of copyright information displayed at startup. Default is *NORMAL.
PASSWORD()	The key word *INLIST in the first 7 bytes indicates that the contents following will be an inlist file description where the passphrase will be retrieved.
TAPFOVR()	The TAPFOVR command is used to control the attributes when creating an archive directly to tape.

PKUNZIP

TYPARCHFL()	*XDB added as an option to specify that archive is to be read exclusively in the QSYS library file system.
MSGTYPE()	An option is added that controls the amount of copyright information displayed at startup. Default is *NORMAL.
PASSWORD()	The key word *INLIST in the first 7 bytes indicates that the contents following will be an inlist file description where the passphrase will be retrieved.

New Products 8.2

The following product has been added to the PKWARE SecureZIP suite for the i5/OS operating environment:

- ***SecureZIP Partner for i5/OS***

New Features 8.2

New features in ***PKZIP for i5/OS*** and ***SecureZIP for i5/OS*** Release 8.2 include:

- New compression algorithms with various custom controls
- Significant performance improvements with new compression algorithms
- New ZIP64 signal constraint checks to avoid building large archives
- New default internal translation tables for EBCDIC to ADCII

- A separate input archive can be specified other than the archive file to created. This allows an inputted archive to be preserved
- A special key word *COPY for the FILES parameter has been added that allows a zip run that just copies files from another archive
- The ability to extract zSeries files created with RDW (EBCDIC variable length records)
- i5/OS PKWARE Save/Restore Application feature (iPSRA)

New Commands 8.2

There are no new commands for version 8.2.

Command Changes & Defaults 8.2

The following commands have changes since version 8.1. Each command and parameter listed below should be reviewed before activating ***PKZIP***^{i} 8.2:

PKZIP

ARCHIVE()	Two additional option added (1. ZIP64 check and 2. optional Input archive name). Defaults are backward compatible.
COMPRESS()	Additional options have been added. Nine (9) new compression levels for Level and a new option for compression method (Deflate or Deflate64). Defaults are backward compatible.
FTRAN()	Default has changed to *ISO88591. See Upgrade/Migration notes #1.
TRAN()	Default has changed to *ISO88591. See Upgrade/Migration notes #1.
FILES()	Revise to accommodate save commands for the iPSRA.

PKUNZIP

FTRAN()	Default has changed to *ISO88591. See Upgrade/Migration notes #1.
TRAN()	Default has changed to *ISO88591. See Upgrade/Migration notes #1.
RSTIPSRA()	The iPSRA Restore command

Migration Considerations from 5.x versions

Upgrade/Migration notes #1:

Installations previously using text translation tables other than ISO9959_1 or PKZ037419 for TRAN or FTRAN should review the data translation characters used.

The newer default tables in*ISO88591 use the IBM ICONV standard character sets for IBM-037 EBCDIC and ISO-8859-1 ASCII code page 819.

In general, the newer default table is better for general-purpose text translation than the older ASCIIUS, ASCIIUSE, ASCIIUK, and ASCIIUKE tables. However, the older tables are still provided for compatibility in case installation-dependent processing requires translation of specialized character sets. In fact the older tables are also provided as a selection in the TRAN and FTRAN parameters as *INTERNAL. The new default for TRAN and FTRAN is *ISO88591. If it is desired to continue with previous defaults, change the default in the PKZIP, PKZSPOOL, and PKUNZIP commands source and recompile the commands or use the CHGCMDDFT command.

For example:

- **CHGCMDDFT CMD(MYZIPLIB/PKZIP) NEWDFT('FTRAN(*INTERNAL) TRAN(*INTERNAL)')**
- **CHGCMDDFT CMD(MYZIPLIB/PKUNZIP) NEWDFT('FTRAN(*INTERNAL) TRAN(*INTERNAL)')**

User Help and Contact Information

For Licensing, please contact the Sales Division at 937-847-2374 or email PKSALES@PKWARE.COM.

For Technical Support assistance, please contact the Product Services Division at 937-847-2687 or visit the Support Web site.

Appendix F lists the types of information needed to resolve issues with the product.

1 Getting Started

PKZIP[i] is a broad, flexible product on the i5/OS, and AS/400 platforms, allowing for compression and decompression of files. It is fully compliant with other PKZIP-compatible compression products running on other operating systems.

Because the PKZIP standard for text data storage is ASCII, ***PKZIP[i]*** facilitates conversion between the ASCII and EBCDIC character sets. Therefore, compressed text files can be transferred between IBM mainframe environments and systems using the ASCII character sets, including UNIX, DOS, ***SecureZIP for zSeries***, and ***PKZIP for zSeries***.

In addition to PKZIP-format archive support, ***PKZIP[i]*** can also produce and manipulate (GNU) GZIP-format archives. See chapter 10.

PKZIP and PKUNZIP Commands

PKZIP[i] uses two main commands—PKZIP and PKUNZIP—to control its high-performance data compression functionality. The PKZIP command launches a utility that compresses files and places them in a ZIP format archive. PKUNZIP reverses this process: it decompresses data in a ZIP archive created by PKZIP or another file compression program and restores the files to their original form. Both commands are controlled by options that allow a variety of functions to be performed.

Multiple levels of processing control are available through the use of customized option modules, shared command lists, and individual job inputs. In addition to file selection, features such as compression levels and performance selections can be specified. Also, a 32-bit cyclic redundancy check (CRC) is a standard feature used to guarantee data integrity.

A ZIP archive is platform-independent; therefore, data compressed (*ZIPPED*) on one platform, for example, UNIX, can be decompressed (*UNZIPPED*) on another platform, for example, OS/400 and MVS/ESA, by using a compatible version of PKUNZIP.

Basic Features of *PKZIP[i]*

PKZIP[i] is generally compatible with ***PKZIP*** 2.x, and as such, has the following features:

Compliance with compression programs on other platforms, including Windows, LINUX, UNIX, DOS, ***SecureZIP for zSeries***, and ***PKZIP for zSeries***.

- User-selected compression ratios.
- Storage capability of 65,535 files within one ZIP Archive.
- Compression of files of up to 4 gigabytes.
- A maximum ZIP archive size of 4 gigabytes.
- Data integrity assurance using 32-bit CRC error detection.
- Translation of data to a system-independent format, thus providing easy file transfers within a mixed or varied file environment.

***PKZIP**i* also offers a series of extended features such as creation of GZIP archive, spool files support, large file support (files greater than 4 GB and files in archive exceeding 65,535), advanced encryption, and self-extracting archives.

Initializing the License

Evaluation Period

You may obtain a key from the Sales Division to use to generate an evaluation license that allows full use of the product for 30 days. Contact PKWARE anytime during this period to obtain licensing to use the product beyond the initial period.

You can reach the Sales Division at 937-847-2374 or email pksales@pkware.com.

For technical support, contact the Product Services Division at 937-847-2687 or online at the Support Web site.

When you receive the license control card information from PKWARE, you build the license data set using the Build License program. Running the INSTPKLIC command updates the LICENSE data set and reports the license status of ***PKZIP**i* at your location.

Release Licensing

Each release of ***SecureZIP for i5/OS*** and ***PKZIP for i5/OS*** requires that a new license key be obtained from Customer Service and that a new license record be generated. The new release will fail with AQZ9077 "License Keys have invalid version setting" if the license file is used from a previous release.

Show System Information

To report on the status of a license at your location, you can run the environment "WHATOSV" program by doing a program call:➔CALL WHATOSV. It will provide a report similar to:

```
PKWARE WHATOSV Current Operating Environment  Thu May 15 12:05:49 2006

SecureZIP for i5/OS Version  9.0.0  with build date 2006/05/14
       Current PKZIP Library is PKW90051S
IBM iSeries Model 9406, Type  270-23E7
Serial Number <010-7X8WT  >, PRC Group < P10>,  OS is at V5R2M0.

Press ENTER to end terminal session.
```

The output of this report is what you will need to send to your reseller or PKWARE sales representative to obtain a DEMO code.

Note: The *PKZIP*^{i} Library must be added to the library list prior to running this program.

Please have the output of this report handy when speaking with your reseller or account rep. You will be expected to supply the following additional information:

- Company name
- Company contact
- Phone number
- Contact email

Applying a License Key or Authorization Code

Installing the PKZIP license activation keys is done by adding the licensing information obtained from PKWARE, Inc. into a source file member (one is provided with distribution library call PKZLICIN) and then running the install license program to activate.

By executing the INSTPKLIC command, the LICENSE dataset will be updated and a report will be produced that will reflect the state of ***PKZIP***^{i} at your location.

Trial activation is accomplished by first editing the member PKWARELIC and adding the company customer record and keys supplied by PKWARE, Inc. One way of editing the member would to use the following command with the correct library:

➔EDTF FILE(PKW90051S/PKZLICIN) MBR(PKWARELIC)

or

➔STRSEU SRCFILE(PKW90051S/PKZLICIN) SRCMBR(PKWARELIC)

Remember since this a source file member and you use the EDTF command that the data will start in column 13, because the source sequence number and date stamp is in the true columns 1 thru 12.

For example:

➔ EDTF FILE(PKW90051S /PKZLICIN) MBR(PKWARELIC)

```
 Edit File: PKW90051S /PKZLICIN(PKWARELIC)
 Record :       1   of       3 by   8            Column :   13     92 by  74
 Control :

CMD ..+....2....+....3....+....4....+....5....+....6....+....7....+....8....+.
     ************Beginning of data**************
    *LICENSED BY PKWARE, Inc   06/03/03 Tait Hamiel
    55 A4CMD1NR 000014581 PKWARE Internal Demo Customer
    99 CMDOAXB1 20030703 0107X8WTP10
     ************End of Data*******************



 F2=Save   F3=Save/Exit   F12=Exit   F15=Services   F16=Repeat find
```

Notice in this case the columns on the ruler shows column 13 for the first column of the license data.

For example:

➔ **STRSEU SRCFILE(PKW90051S/PKZLICIN) SRCMBR(PKWARELIC) :**

```
 Columns . . . :    1  71           Edit                    PKW90051S/PKZLICIN
 SEU==>                                                        PKWARELIC
 FMT **  ...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7
        ************** Beginning of data ************************************
0001.00 *LICENSED BY PKWARE, Inc   06/03/03 Tait Hamiel
0002.00 55 A4CMD1NR 000014581 PKWARE Internal Demo Customer
0003.00 99 CMDOAXB1 20030703 0107X8WTP10
        ****************** End of data ***************************************




 F3=Exit   F4=Prompt   F5=Refresh   F9=Retrieve   F10=Cursor   F11=Toggle
 F16=Repeat find       F17=Repeat change          F24=More keys
```

Once you have typed or copied the license information provided by PKWARE, you will need to save these changes by pressing F3 and exit the edited member by pressing F3 again. Next, run the install program using the following command:

➔**INSTPKLIC INFILE(*LIBL/PKZLICIN) INMBR(PKWARELIC) or prompt F4**

```
                      Install SecureZIP for i5/OS License (INSTPKLIC)

 Type choices, press Enter.

 Type . . . . . .   *INSTALL      *INSTALL, *VIEW
 Input Control File . . . . . .   PKZLICIN      Name, PKZLICIN
   Library name . . . . . .     *LIBL       Name, *LIBL
 Control Member . . . . . .   pkwarelic         Name, *FIRST




                                                                      Bottom
 F3=Exit   F4=Prompt   F5=Refresh   F12=Cancel   F13=How to use this display
 F24=More keys
```

By executing the INSTPKLIC command, the LICENSE dataset will be updated and a report will be produced that will reflect the state of ***PKZIP***^{i} at your location.

```
SecureZIP(R) for i5/OS Version 9.0.0,  2006/05/26
Portions copyright (C) 1989-2006 PKWARE, Inc. All rights reserved.
PKZIP Reg. U.S. Pat. and Tm. Off. Patent No. 5,051,745
PKZIP Reg. U.S. Pat. and Tm. Off.  Patent No.  7,793,099
PKZIP Reg. U.S. Pat. and Tm. Off.  Patent No.  7,844,579
Other patent applications pending
Portions of this software include RSA BSAFE(R) cryptographic or security protocol
software from RSA Security Inc.
SecureZIP(R) is a trademark of PKWARE, Inc.
Machine ID = 01061B5F, Processor Group =  P05, OS=V5R3M0
Rec -  1 *LICENSED BY PKWARE   05/26/06 WSS
Rec -  2 *SecureZIP with Enterprise License
Rec -  3 57 MR6CCP2B 000015319 PKWARE, INC.
Rec -  4 99 HH6QYPKK 20060626 01061B5FP05
Evaluation Edition being installed
SecureZIP Module      - Evaluation set to expire in 31 days on 20060626
Compression           - Evaluation set to expire in 31 days on 20060626
Decompression         - Evaluation set to expire in 31 days on 20060626
GZIP                  - Evaluation set to expire in 31 days on 20060626
Enhanced Decryption   - Evaluation set to expire in 31 days on 20060626
Spool Files           - Evaluation set to expire in 31 days on 20060626
Large Files           - Evaluation set to expire in 31 days on 20060626
Self Extracting       - Evaluation set to expire in 31 days on 20060626
iPSRA Save/Restore    - Evaluation set to expire in 31 days on 20060626
TapeOut IO Handler    - Evaluation set to expire in 31 days on 20060626
License File PKW90051S/PKZLIC(PKZLIC) Updated successfully
```

Reporting the *PKZIP^{i} for i5/OS* License

By using the INSTPKLIC TYPE(*VIEW) command, the current licensing settings will be displayed.

```
SecureZIP(R) for i5/OS Version 9.0.0,  2006/05/26
Portions copyright (C) 1989-2006 PKWARE, Inc. All rights reserved.
PKZIP Reg. U.S. Pat. and Tm. Off. Patent No. 5,051,745
PKZIP Reg. U.S. Pat. and Tm. Off.  Patent No.  7,793,099
PKZIP Reg. U.S. Pat. and Tm. Off.  Patent No.  7,844,579
Other patent applications pending

Portions of this software include RSA BSAFE(R) cryptographic or security protocol
software from RSA Security Inc.
SecureZIP(R) is a trademark of PKWARE, Inc.
Machine ID = 01061B5F, Processor Group =  P05, OS=V5R3M0
*****************************************
A License Report requested on 0107X8WT  from CPU Serial#
9.0 Product Licensed to Customer # 000014581 -PKWARE Internal Demo Customer
*****************************************
Compression           -DEMO with 23 Days remaining (07/03/2003)
      Contact PKWARE, Inc. for Licensing
*****************************************
Decompression         -DEMO with 23 Days remaining (07/03/2003)
      Contact PKWARE, Inc. for Licensing
*****************************************
GZIP                  -DEMO with 23 Days remaining (07/03/2003)
      Contact PKWARE, Inc. for Licensing
*****************************************
IFS File Handlers     -DEMO with 23 Days remaining (07/03/2003)
      Contact PKWARE, Inc. for Licensing
*****************************************
Database File Handlers-DEMO with 23 Days remaining (07/03/2003)
      Contact PKWARE, Inc. for Licensing
*****************************************
Advanced Encryption   -DEMO with 23 Days remaining (07/03/2003)
      Contact PKWARE, Inc. for Licensing
*****************************************
Spool Files           -DEMO with 23 Days remaining (07/03/2003)
```

```
      Contact PKWARE, Inc. for Licensing
*****************************************
Self Extracting       -DEMO with 23 Days remaining (07/03/2003)
      Contact PKWARE, Inc. for Licensing
*****************************************
Press ENTER to end terminal session.
```

PKZIP and SecureZIP for i5/OS Grace Period

PKWARE recognizes that there may be periods where the licensing environment established by the customer is no longer valid. Circumstances such as disaster recovery processing or the installation or upgrade of new processors will affect the environment.

To accommodate the installation, ***PKZIP^{i}*** has a process that will allow you to continue to use the product for a grace period of seven days when the established licensing environment is no longer valid. Note that the user *must* have write authority on the license dataset to invoke the grace period. This authority is only required the first time PKZIP/PKUNZIP is run after a CPU change has occurred; it is not required after the grace period has been successfully invoked (this is one time per CPU).

During the grace period, error messages will be displayed on the job log and/or display/printout for each execution of ***PKZIP^{i}***. At the end of the period, if the license is not updated, the product will no longer function for the new CPUs except to VIEW an archive. You must contact PKWARE at pkcustomerservice@pkware.com during the grace period to obtain licensing to allow extended use.

Invoking *PKZIP^{i}* Services

Three main commands control ***PKZIP^{i}*** functionality in the OS/400 operating environments. The commands are:

- PKZIP - Launches compression utility
- PKUNZIP - Launches archive extraction utility
- PKZSPOOL - Launches compression utility for spool files

Each of the commands can be invoked interactively, submitted for a batch run, or used anywhere that an i5/OS command can be issued.

Help panels for each command can be activated by using the F1 (help) key.

PKZIP^{i} Differences from other Platforms

This section covers the differences between ***PKZIP^{i}*** and other versions, including versions that run on other operating systems or platforms. Most of the differences are due to the QSYS library file type system and the i5/OS object-oriented base.

Attributes (non-extended)	Various MS/DOS options support the selection of files by file attributes such as hidden, read-only, and system. These attributes are not meaningful on the OS/400 file system.
ANSI comments	Because OS/400 does not support ANSI control codes, related options are not supported. When unzipping from an archive, the archive comment will be displayed, but ANSI control codes in this comment will not be masked out. This could cause attribute changes on the i5/OS display.
Archive file date controls *PKZIP*	DOS options control whether the ZIP file date is updated or retained when altering the archive. Because the last used date on OS/400 is not under program control or alterable by a command, these options are not supported.
Archive Comments *PKZIP*	DOS options allow editing of comments for individual files in an archive. This version supports editing of a file's text description, but is not recommended for batch running, or for a large number of files due to the interactive message responses required.
File naming differences	The files used in the QSYS library file system have their own naming style. Each file associated with a library file and members would be depicted as library/file(member). Usually, all file names are stored as open system file names with directories, ending with a file name. For a detailed description and techniques see chapter 5.
HELP	***PKZIP for DOS™*** has options to display a list of commands. Because ***PKZIP^{i}*** uses i5/OS commands, the help system is built for each command and is activated by PF1 on each parameter.
Mixed Case Filenames	When using the IFS (integrated file system), the file names are case sensitive and act like other file systems (UNIX, DOS, Windows, etc.). When using the QSYS library file system, the file names are always in UPPER CASE. Occasionally, when trying to update and archive (or select from an archive), you may encounter a case sensitive search. Use PKUNZIP view to get the exact name stored. This would be appropriate when doing a PKZIP TYPE(*DELETE) where the selection file would need to match.

Use of SAVF Method

At this time, only physical files with attributes of PF-DTA, PF-SRC, and SAVF in the QSYS file system, stream files and directories in the IFS, and spool files can be processed by ***PKZIP^{i}.*** Also, some special database functionality such as triggers, file constraints, alternate collating sequence, and large object fields, are not stored in the archives.

To overcome some of the restrictions listed above, ***PKZIP^{i}*** can compress and decompress SAVF. The objects to be compressed are saved to a SAVF using SAVOBJ or SAVLIB. The SAVF is then compressed to an archive using PKZIP. To restore the data, first use PKUNZIP to re-create the SAVF, and then use RSTOBJ or RSTLIB to

restore objects from within the decompressed SAVF. SAVF are binary and only pertain to OS/400.

Another solution may be to utilize the i5/OS PKWARE Save/Restore Application Feature (iPSRA) feature, where the save command can issued with the save API based on the command defined in the FILES parameter. See chapter 6 on the iPSRA feature for details.

Data Compression

Because data compression techniques reduce file size, a compressed data file will use less storage space and can be transferred in a faster, more efficient manner. A file can be compressed (a ZIP candidate) to a compact size (ZIPPED file), and then to use the file again, it must be uncompressed or extracted to its original size (UNZIPPED file).

One easy data compression method eliminates repeating or redundant data by replacing it with representative information that will be used when restoring the data. An example of this data compression technique is the Run-Length Encoding method, which applies to redundant data where a repeating character (the run) is represented as a count or value (the length). The compressed form is the repeated character with its count.

Example: B 2 2 2 2 E H H H H H H H H H

Compressed: B *4 2 E *9 H

Note: The efficiency of this method is dependent upon the amount of redundancy in the data.

To perform a thorough compression operation, more advanced algorithms and enhanced techniques are required. ***PKZIP***^{i} uses just such methods to achieve maximum results.

ZIP Archives

PKZIP^{i} is capable of storing compressed data in ZIP archives. There is no limit to the number of archives you may create.

ZIP archive capability:

- ZIP archive refers to any valid ZIP-format file created by a ***PKZIP®*** 4.x-compatible product.
- ZIP64 refers to ZIP archives that include the ZIP64 format that can handle more than 65,534 files and files that exceed 4 GB. (See “Large Files Considerations.”).
- Each standard archive can store up to 65,534 files.
- Files that are over 4 GB have to be archived with GIZP or by using the Large File Support.
- Each standard archive may contain up to 4 GB of data. ZIP64 is required for larger archives.

For each file in the archive, the following information is stored with the compressed data:

- Filename.
- File directory date and time.
- File's initial CRC value (see Cyclic Redundancy Check).
- Method of compression used.
- ***PKZIP**i* version required for file extraction.
- File size, uncompressed.
- File size, compressed.

Some files may contain the following additional information:

- The version of ***PKZIP**i* that created the file.
- File attributes.
- Any comment about the file.
- Any comment about the archive.
- Platform specific attributes (see Cross Platform Compatibility).
- If encrypted and what method of encryption.

Cyclic Redundancy Check

Cyclic redundancy check (CRC) is a method used to verify the integrity of a data file after it is restored from a ZIP archive.

Before a file is compressed, a ***PKZIP**i* algorithm computes a 32 bit hexadecimal value for its data. The CRC value is stored in a file that is within the ZIP archive. When the data in the file is extracted, ***PKZIP**i* processes it again using the same algorithm to produce a second CRC value. Once the file is processed, the original CRC value is compared to the new CRC value to ensure that they match.

Note: If the data is the same as its previous state, the same CRC value will be produced. When the two CRC values are compared, and should the extracted value not match the stored, initial value, the integrity of the file is in question and ***PKZIP**i* reports the results. In this case, it is possible the data was corrupted within the ZIP Archive.

Encryption

Requires SecureZIP

SecureZIP for i5/OS can encrypt data for security control and provide a passphrase lockout for extracting data. Various security levels are available, with multiple encryption algorithms. See chapter 2 for a description of security features in ***SecureZIP for i5/OS***.

File Selection and Name Processing

This section discusses how file selection is performed for ZIP processing with ***PKZIP[i]***. The primary commands used for ZIP processing are discussed here, along with some overview notes and known restrictions.

This section also discusses how files are selected within an iSeries environment. Remember, ZIP directory entries within a ZIP archive will be defined in a system-independent format, which is not iSeries compatible.

Note: Directory entries within a ZIP archive are actually in a format compatible with UNIX systems and have been translated into the ASCII character set. In addition, the dataset level separators are typically set as the forward slash ("/"), not the period (".") as in iSeries, although this can be controlled through command actions in ***PKZIP[i].***

See chapter 5 for further information on how ***PKZIP[i]*** handles file name interchanges between iSeries and common ZIP format.

Primary File Selection Inputs

PKZIP[i] will only process:

- iSeries objects of type FILE (only with attributes PF-SRC, PF-DTA, and SAVF).
- IFS stream files (*STMR) and IFS directories(*DIR).
- Spool files.

Other objects must first be unloaded into an iSeries save file (SAVF) before they can be processed by ***PKZIP[i]*** (see: Use of SAVF Method) or use the Save Applications data with the iPSRA feature. See chapter 6.

The FILES parameter in both PKZIP and PKUNZIP specifies which files are to be processed for all files except spool files (SPLF have their own selection parameters). One or more names can be specified, and each name is in either OS/400 QSYS format, or IFS format, depending on F2ZTYPE settings. An asterisk may be used at the end of the library name, file name, or member name to select names beginning with the prefix used. To select all members of a file, *ALL may be used. To select all files in a library *ALL may be used (as long as it is qualified by at least a library name), for example, FILES('mylib/*ALL'). If *ALL is specified without at least a qualifying library name, the specification is ignored and no files will be selected.

The ***PKZIP[i]*** QSYS file system expands a partial file specification in several ways to make file specification more convenient. Each file specification may consist of a filename; a library name; a file name and member name; a library name and file name; or a library name, file name, and member name. iSeries SAVF may also be selected, but because a *SAVF file does not contain members, a SAVF will not be selected if a member name was included in the file specification.

In the Integrated file system, each file specification may consist of a directory, a path of directories, a directory and file, or a path of directories and file.

The various combinations that may be used are shown below:

File Type	*File specification*	*Expanded As*	*Notes*
QSYS	library*/	library*/*all(*all)	Finds all files in libraries beginning with *library*.
	fileinlib	*LIBL/fileinlib(*ALL)	Searches library list for all files called *fileinlib*. If a matching file is found, all of its members will be selected. If a SAVF is found, it will be selected.
	fileinlib*(mem*)	*LIBL/fileinlib*(mem*)	Searches library list for all files beginning with *fileinlib*. If a matching file is found, members beginning with *mem** will be selected. If a SAVF is found, it will NOT be selected because the file specification includes a member name.
	library*/file*	library*/file*(*ALL)	Searches libraries that begin with *library prefix* and for files that begin with *file prefix*. If a matching file is found, all of its members will be selected. If a SAVF is found, it will be selected.
	library*/file*(memo*)	library*/file*(mem*)	Searches libraries that begin with library prefix and files that begin with file prefix. If a matching file is found, members beginning with mem prefix will be selected. If a SAVF is found, it will not be selected because the file specification includes a member name.
IFS	Dir/*	Dir/*all	Searches all files in path DIR.
Spool Files	N/A		Uses parameters: SPLFILE, SFUSER, SFQUEUE, SFFORM, SFUSRDTA, SFSTATUS, SFJOBNAM, and/or SPLNBR.

File Type	File specification	Expanded As	Notes
iPSRA	Full Save Command		SAV, SAVLIB, SAVOBJ, SAVCHGOBJ, or SAVDLO

Note: If parameter TYPE(*DELETE) is used, then the file name format for these names must be in MS/DOS format (that is, if CVTFLAG has not been used). See the FILES keyword. Files may also be excluded. See the EXCLUDE keyword.

The valid parameter values for the FILES keyword are as follows:

'file specification 1' 'file specification 2'...'file_specification nn'

This is the list of one or more file specifications, separated by spaces.

For example:

mylib/myfile(prf*)

mylib/*all(*all)

By default, ***PKZIP[i]*** does a match on files in the QSYS library system with no case sensitivity and in the IFS with case sensitivity. Some IFS file systems contain case sensitive file names. To force ***PKZIP[i]*** to perform non-case sensitive file name matching use TYPFL2ZP(*IFS2).

File Exclusion Inputs

Using similar file specification techniques as described above in the Primary file Selection Inputs section, ***PKZIP[i]*** can specify from one to many file patterns that will be used to exclude files that were selected with the FILE parameter. The files can be inputted into the command parameter EXCLUDE or into a text file that can be processed by parameter EXCLFILE.

Care should be taken when using wildcards excluding inputs to ensure that FILES and EXCLUDE parameters select the desired files.

Input ZIP Archive Files

During a FRESHEN or UPDATE request, files contained within the existing ZIP archive are added to a candidate list. Names stored previously are used to search the system files for viability (any file names not found in the system remain in the ZIP archive).

SPOOL File Selecting

The FILES parameter is not used to select spool files for compression, but instead uses its own selection parameters.

There are eight positional parameters that can be specified to select the spool files: the SPOOL FILE NAME (SPLFILE), the SPOOL FILE NUMBER (SPLNBR), the user that created the files (SFUSER), the OUTQ that the file is residing (SFQUEUE), the form type specified (SFFORM), the user data tag associated with the spool file (SFUSRDTA), the status of the spool file (SFSTATUS), or the specific job name/user

name/job number (SFJOBNAM). Only files that meet all of the selection values will be selected.

If the parameter SFJOBNAM is coded, the job must exist and the parameter SFUSER will be ignored, since it is already part of the SFJOBNAM parameter.

Large Files Considerations

Large File Support Summary

The large file support feature known as ZIP64 throughout this manual was added to ***PKZIP for i5/OS*** in release 5.6. This separately licensed feature of ***PKZIP^{i}*** provides several enhancements relating to capacity, size, and performance. Some of the key features include:

Processing support (ZIP and UNZIP) for Archives enabled with the standard ZIP64 formats from other platforms.

An increased ZIP archive file capacity is raised from 65,534 to the theoretical limit of 4,294,967,295 files.

An increased user file size handler, raised from 4 Gigabytes minus 1 byte (32 bit binary counter) to a theoretical limit of 9 Exabytes (64 bit binary counter).

An increased support for ZIP archive sizes exceeding 4 Gigabytes (same as user file size limit).

The preceding values are given only as theoretical limits. In practice, there are reasonable limitations due to the availability of resources along with processing tolerances.

Note: 4 GB or Gigabyte is equal to 4,294,967,295 bytes. 9 EB or Exabyte is equal to 9,223,372,036,854,775,807 bytes.

Large File Support File Capacities

The original .ZIP file format has faithfully met the needs of computer users since it was introduced by PKWARE in 1989. As computer technology has advanced over time, storage capacities have increased dramatically. These increases make the numbers and sizes of files that seemed unimaginable ten years ago a reality today. To extend the utility of the .ZIP file format to meet these changing system needs, PKWARE extended the .ZIP file format to support more than 65,535 files per archive and archive sizes greater than 4 Gigabytes (GB). This is known as the ZIP64 format.

The specification for the .ZIP file format has been publicly available and distributed by PKWARE in a file called APPNOTE.TXT. This file documents the internal data structures and layout that define a .ZIP archive. The extensions introduced by PKWARE fully supports all the features of your existing archives and newer versions of PKZIP that supports these new extensions will continue to read all of your current archives. Prior to the ***PKZIP for i5/OS*** 5.6 release, versions of PKZIP on the OS/400 were limited to storing no more than 65,534 files in a .ZIP archive.

Another limitation that existed prior to the 5.6 version of the ***PKZIP for i5/OS*** was that a single .ZIP archive or files in archive could not be larger than 4 GB (4,294,967,295 bytes). The extended ZIP64 file format specification available with

PKZIP 5.6 supports creating .ZIP archives containing over 4 billion files and with sizes larger than 9 quintillion bytes. These are only theoretical limits and most iSeries systems and other computer systems in common use today do not have enough storage capacity, CPU or available memory to create and store ZIP64 archives approaching these limits.

The practical limits imposed by a typical iSeries system in use today and configured with various memory sizes will support compressing up to approximately 265,000 files. Compressing this number of files can take a long time, not only for the compression process, but to manage the directories and properties of each of these files.

Your available system resources (processor speed, DADS, Memory, and other processing) limits the performance you can expect from PKZIP*i* when processing large numbers of files or large archives. If you are compressing large numbers of files on an iSeries with insufficient memory or other resources you can expect slow processing.

When compressing large files, it is a good idea to have your archives set up to be stored in the IFS rather than in a library/file. The overhead is much less when storing the archive in the IFS. It is even more important when updating or adding to an archive where the temporary archive will also be processed in the IFS

Versions of ***PKZIP for i5/OS*** prior to 5.6 will not recognize these new features and will be unable to view or extract any files in your archives that are dependent on these ZIP64 features. Also, any ZIP compatible programs you may be using from other companies will not be able to access all of the contents of your large archives. They may report that an archive is too large, or they may incorrectly report that the archive has errors. To ensure access to data in your large archives, always use genuine PKZIP/SecureZIP from PKWARE.

Cross Platform Compatibility

Cross platform compatibility provides ***PKZIP****i* its ability to allow data to move between different computer operating environments. ***PKZIP****i* was designed for cross platform use. Regardless of platform, ***PKZIP****i* archives are compatible with ***SecureZIP for zSeries***, ***PKZIP for zSeries, PKZIP for MVS, PKZIP for OS/400***, ***PKZIP for UNIX***, ***PKZIP for LINUX***, ***PKZIP for DOS***, and ***PKZIP for Windows*** to name a few. Because ***PKZIP****i* automatically converts the data between EBCDIC and ASCII, files prepared on the host are readable on any PC or UNIX system. The internal format of a ZIP archive is identical regardless of which platform compressed the files that the archive contains. If you want to transfer data across platforms using any other ZIP utility, you should always run a test to verify the cross platform compatibility.

PKZIP*i* uses the same ZIP file format used by other ZIP compatible products, independent of the platform on which it is running. ***PKZIP****i* archives are not platform dependent allowing greater flexibility in file usage. Data can be zipped on one platform, for example UNIX, and unzipped onto another platform, such as OS/400. To do this, ***PKZIP****i* converts the data structure into the ZIP format and saves the appropriate file information in the ZIP archival directory entries.

ZIP File Format Specification

The following table lists features of the ZIP file format specification supported on the zSeries and iSeries platforms. The notation *(EE)* on some entries—for example, *PK8.2(EE)*—stands for *Enterprise Edition*.

ZIP Feature	*Version*	*MVS/zSeries*	*OS400/i5/OS*
Default	1.0		
File represents a volume label	1.1	Not supported	Not supported
File represents a folder	2.0	Not supported	Not supported
Deflate compression	2.0	2.x	2.x
Traditional encryption	2.0	2.x	2.x
Deflate64 compression	2.1	Not supported	Not supported
DCL Implode compression	2.5	Not supported	Not supported
File is a patched data set	2.7	Not supported	Not supported
File uses Zip64 size extensions	4.5	5.6	5.6
BZip2 compression	4.6	Not supported	Not supported
DES encryption	5.0	SZ8.0, PK8.2(EE)	SZ8.0, PK8.2(EE)
3DES encryption	5.0	SZ8.0, PK8.2(EE)	SZ8.0, PK8.2(EE)
RC2 encryption	5.0	SZ8.0, PK8.2(EE)	SZ8.0, PK8.2(EE)
RC4 encryption	5.0	SZ5.5, PK8.2(EE)	SZ5.5, PK8.2(EE)
AES encryption	5.1	SZ8.0, PK8.2(EE)	SZ8.0, PK8.2(EE)
Certificate encryption using non-OAEP key wrapping	6.1	8.0 (SecureZIP)	8.0 (SecureZIP)
Central Directory Encryption (File Name Encryption)	6.2	8.0 (SecureZIP)	8.0 (SecureZIP)

If you want to transfer data across platforms using any other ZIP-compatible product, you should check with the supplier first to confirm which versions of PKZIP it is compatible with.

For more information regarding data formats, see “Data Format - Text Records vs. Binary Records” in chapter 3 for a discussion regarding special considerations when transferring files between different platform types.

PKZIP/SecureZIP for i5/OS Restrictions

Due to various iSeries processing characteristics, the following restrictions should be carefully reviewed to determine the best way to proceed when using ***PKZIP^{i}***:

PKZIP^{i} in the QSYS file system will only work with objects that have an object type of *FILE and an attribute of PF-DTA, PF-SRC, and SAVF. To process other objects

such as *PGM, *CMD, etc., use the SAVF method (see “Use of SAVF Method” in chapter 1).

***PKZIP**^{i}* in the integrated file system (IFS) will only work with stream files (*STRM) and directories (*DIR).

Special database functionality, such as triggers, file constraints, alternate collating sequence, and logical files are not stored in an archive. To maintain this functionality, use the SAVF method (see “Use of SAVF Method”).

Special database fields for large objects (LOB) are not supported. These fields include: character large objects (CLOBs), double-byte character large objects (DBCLOBs), and binary large objects (BLOBs). In cases where the database contains one of these types of fields, use the SAVF Method.

2 Introduction to Data Security

Requires SecureZIP

This chapter details how ***SecureZIP for i5/OS*** can strongly encrypt data for security control and protection. Much of the reference information in this chapter derives from the National Institutes of Standards and Technology. The NIST Computer Security Resource Center web site, *http://csrc.ncsl.nist.gov/*, contains FAQ's and documentation relating to computer security along with the Federal Information Processing Standard (FIPS) documents. In addition, the PKWARE web site, *WWW.PKWARE.COM*, contains information relating to security and links to the RSA Security, Inc web site that describes in detail the BSAFE implementation used in ***SecureZIP for i5/OS***.

The following sections describe encryption, authentication, types of algorithms in use, information about specific mandates requiring the use of secure data and how ***SecureZIP for i5/OS*** will secure that data.

Encryption

Encryption provides confidentiality for data. The data to be protected is called plaintext. Encryption transforms the plaintext data into an unreadable form, called ciphertext, using an encryption key. Decryption transforms the ciphertext back into plaintext using a decryption key. Several algorithms have been approved in FIPS for the encryption of general purpose data. Each of these algorithms is a symmetric key algorithm, where the encryption key is the same as the decryption key. In order to maintain the confidentiality of the data encrypted by a key, the key must be known only by the entities that are authorized to access the data. These symmetric key algorithms are commonly known as block cipher algorithms, because the encryption and decryption processes each operate on blocks (chunks) of data of a fixed size.

FIPS 46-3 and FIPS 197 have been approved for the encryption of general-purpose data. The protection of keys is discussed below under Key Management.

SecureZIP for i5/OS uses symmetric key algorithms when encrypting user data.

Note: ***PKZIP for i5/OS*** provides support for passphrase-based encryption and decryption using a 96-bit "Standard" encryption algorithm that is supported by older ZIP-compatible utilities. In addition, ***PKZIP for i5/OS*** Enterprise Edition supports

the decryption of all passphrase-based algorithms provided in ***SecureZIP for i5/OS***.

Authentication

Authentication is the process of validating digital signatures that may be attached to files in an archive or to an archive's central directory.

Authentication is a separate operation from data encryption. Whereas encryption is concerned with preventing parties from accessing sensitive data (such as private medical or financial information), authentication confirms that information actually comes unchanged from the purported source.

Authenticating digitally signed data both verifies the signature and validates the signed data.

Data Integrity

SecureZIP uses a cyclic redundancy check (CRC) to ensure that data is successfully transferred into and out of a ZIP archive. The CRC process creates a unique hash value "thumbprint" from the original data stream. The thumbprint is regenerated at the receiving end and compared with the hash of the source for equality. The thumbprint value is stored independently of the data stream and is used during UNZIP processing to complete validation of the data.

SecureZIP extends the concept of the CRC in two ways for the purpose of providing a tamper-resistant container within the ZIP archive. First, more rigorous HASH algorithms (MD5 and SHA-1) are used (as specified by the PKCFGSEC command with the parameter SIGNPOL) in addition to the 32-bit CRC to accurately reflect the uniqueness of the data stream. Second, the hash value is encrypted in a digital signature using a private-key certificate for the purpose of tamper detection after file extraction.

For more information regarding SHA-1 (Secure Hash Algorithm), see FIPS PUB 180-1, describing the Secure Hash Standard, at http://www.itl.nist.gov/fipspubs/fip180-1.htm.

SecureZIP for i5/OS provides the parameter SIGNERS (*ALL,*FILE, *ARCHIVE), to initiate the creation of digital signatures within the ZIP archive. The AUTHCHK command is used to perform a tamper check operation using the digital signature and hash.

Digital Signature Validation

SecureZIP makes use of certificate-based encryption within the public key infrastructure (PKI) to generate and validate digital signatures. PKI provides an authentication chain for certificates to guarantee that the signature was created by the purported source. ***SecureZIP*** supports the certificate chain authentication process by including necessary identification information within the ZIP archive. Subsequently, the certificate(s) used for signing can be authenticated through a complete chain of trust.

To complete the chain of trust, a *root* (or self-signed) certificate representing the certificate's issuing organization is installed on the authenticating system. This

provides the receiving organization with the authority to declare how the final trust sequence should be treated. Signatures based on certificates from certificate authorities (CA) that are not authorized or trusted are declared as being *untrusted* by ***SecureZIP***.

Additional facets of validating a certificate's viability for use include a defined range of dates within which a certificate may be used and whether the certificate has been declared to have been revoked. Configurable SecureZIP policies (EXPIRED and REVOKED attributes) provide support to ensure that the certificates involved in authentication also adhere to these restrictions.

SecureZIP for i5/OS provides a means to install and access the certificates necessary for signing and authentication. The AUTHCHK command, along with configured policy settings governs the type (archive directory or data files) and level of authentication that is to be performed.

Digital Signature Source Validation

A final step in completing the authentication process is to ensure that the archive and/or file data was sent from a particular source. Up to this point, using the previous two aspects of authentication, we are certain that the archive directory and/or files were signed with a private-key certificate that came from a trusted source (CA) and that the data stream has not been tampered with since it was placed into the ZIP archive. However, these steps alone do not guarantee that a different party under the same root/CA chain did not perform the signing operation.

SecureZIP for i5/OS provides an optional parameter in the AUTHCHK command to declare the specific party from whom the data is expected.

Public-Key Infrastructure and Digital Certificates

Public-Key Infrastructure (PKI)

Use of digital certificates for encryption and digital signing relies on a combination of supporting elements known as a *public-key infrastructure* (PKI). These elements include software applications such as SecureZIP that work with certificates and keys as well as underlying technologies and services.

The heart of PKI is a mechanism by which two cryptographic keys associated with a piece of data called a certificate are used for encryption/decryption and for digital signing and authentication. The keys look like long character strings but represent very large numbers. One of the keys is private and must be kept secure so that only its owner can use it. The other is a public key that may be freely distributed for anyone to use to encrypt data intended for the owner of the certificate or to authenticate signatures.

How the Keys Are Used

With encryption/decryption, a copy of the public key is used to encrypt data such that only the possessor of the private key can decrypt it. Thus anyone with the public key can encrypt for a recipient, and only the targeted recipient has the key with which to decrypt.

With digital signing and authentication, the owner of the certificate uses the private key to *sign* data, and anyone with access to a copy of the certificate containing the public key can authenticate the signature and be assured that the signed data really proceeds unchanged from the signer.

Authentication has one additional step. As an assurance that the signer is who he says he is—that the certificate with Bob's name on it is not fraudulent—the signer's certificate itself is signed by an issuing certificate authority (CA). The CA in effect vouches that Bob is who he says he is. The CA signature is authenticated using the public key of the CA certificate used. This CA certificate too may be signed, but at some point the *trust chain* stops with a self-signed *root* CA certificate that is simply trusted. The PKI provides for these several layers of end-user public key certificates, intermediate CA certificates, and root certificates, as well as for users' private keys.

X.509

X.509 is an International Telecommunication Union (ITU-T) standard for PKI. X.509 specifies, among other things, standard formats for public-key certificates. A public-key certificate consists of the public portion of an asymmetric cryptographic key (the public key), together with identity information, such as a person's name, all signed by a certificate authority. The CA essentially guarantees that the public key belongs to the named entity.

Digital Certificates

A digital certificate is a special message that contains a public key and identify information, such as the owner's name and perhaps email address, about the owner. An ordinary, end-user digital certificate is digitally signed by the CA that issued it to warrant that the CA issued the certificate and has received satisfactory documentation that the owner of the certificate is who he says he is. This warrant, from a trusted CA, enables the certificate to be used to support digital signing and authentication, and encryption of data uniquely for the owner of a certificate.

For example, Web servers frequently use digital certificates to authenticate the server to a user and create an encrypted communications session to protect transmitted secret information such as Personal Identification Numbers (PINs) and passphrases.

Similarly, an email message may be digitally signed, enabling the recipient of the message to authenticate its authorship and that it was not altered during transmission.

To use PKI technology in ***SecureZIP for i5/OS*** for encryption and to attach digital signatures, you must have a digital certificate.

Certificate Authority (CA)

A certificate authority (CA) is a company (usually) that, for a fee, will issue a public-key certificate. The CA signs the certificate to warrant that the CA issued the certificate and has received satisfactory documentation that the owner of the new certificate is who he says he is.

Private Key

A digital certificate contains both private and public portions of an asymmetric cryptographic key together with identity information, such as a person's name and (possibly) email address. The private portion of the key is called the *private key* and is used to decrypt data encrypted with the associated public key and to attach digital signatures.

A private key must be accessible solely by the owner of the certificate because it represents that person and provides access to encrypted data intended only for the owner.

SecureZIP for i5/OS uses a private key maintained in x.509 PKCS#12 format. This means that the private key cannot be accessed unless a passphrase is entered for each SecureZIP request.

Public Key

A *public key* consists of the public portion of an asymmetric cryptographic key in a certificate that also contains identity information, such as the certificate owner's name.

The public key is used to authenticate digital signatures created with the private key and to encrypt files for the owner of the key's certificate.

For information on the *digital enveloping* process ***SecureZIP for i5/OS*** uses for certificate-based encryption, see the *Secure .ZIP Envelopes* whitepaper at the PKWARE Web site.

Certificate Authority and Root Certificates

End entity certificates and their related keys are used for signing and authentication. They are created at the end of the trust hierarchy of certificate authorities. Each certificate is signed by its CA issuer and is identified in the "Issued By" field in the end certificate. In turn, a CA certificate can also be issued by a higher level CA. Such certificates are known as *intermediate* CA certificates. At the top of the issuing chain is a self-signed certificate known as the *root*.

SecureZIP for i5/OS uses public-key certificates in PKCS#7 format. The intermediate CA certificates are maintained independently from the ROOT certificates.

Types of Encryption Algorithms

FIPS 46-3, Data Encryption Standard (DES)

The FIPS (Federal Information Processing Standards) specification 46-3 formerly specified the DES algorithm for use in Federal government applications. In 2004, the specification was changed such that DES is no longer approved for Federal government applications.

Triple DES Algorithm (3DES)

Triple DES is a more recent algorithm related to DES. Triple DES is a method for encrypting data in 64-bit blocks using three 56-bit keys by combining three successive invocations of the DES algorithm.

ANSI X9.52 specifies seven modes of operation for 3DES and three keying options: 1) the three keys may be identical (one key 3DES), 2) the first and third key may be the same but different from the second key (two key 3DES), or 3) all three keys may be different (three key 3DES). One key 3DES is equivalent to DES under the same key; therefore, one key 3DES, like DES, will not be approved after 2004. Two key 3DES provides more security than one key 3DES (or DES), and three key 3DES achieves the highest level of security for 3DES. NIST recommends the use of three different 56-bit keys in Triple DES for Federal Government sensitive/unclassified applications.

SecureZIP for i5/OS uses three-key 3DES when Triple DES is selected as the data encryption algorithm.

Advanced Encryption Standard (AES)

The Advanced Encryption Standard (AES) encryption algorithm specified in FIPS 197 is the result of a multiyear, worldwide competition to develop a replacement algorithm for DES. The winning algorithm (originally known as Rijndael) was announced in 2000 and adopted in FIPS 197 in 2001.

The AES algorithm encrypts and decrypts data in 128-bit blocks, with three possible key sizes: 128, 192, or 256 bits. The nomenclature for the AES algorithm for the different key sizes is AES-x, where x is the size of the AES key. NIST considers all three AES key sizes adequate for Federal Government sensitive/unclassified applications.

Please see http://www.nist.gov/public_affairs/releases/g00-176.htm a press release recapping NIST's position

SecureZIP for i5/OS uses AES as the default encryption algorithm.

Comparison of the 3DES and AES Algorithms

Both the 3DES and AES algorithms are considered to be secure for the foreseeable future. Below are some points of comparison:

- 3DES builds on DES implementations and is readily available in many cryptographic products and protocols. The AES algorithm is new; although many implementers are quickly adding the algorithm to their products, and protocols are being modified to incorporate the algorithm, it may be several years before the AES algorithm is as pervasive as 3DES.
- The AES algorithm was designed to provide better performance (e.g., faster speed) than 3DES.
- Although the security of block cipher algorithms is difficult to quantify, the AES algorithm, at any of the key sizes, appears to provide greater security than 3DES. In particular, the best attack known against AES-128 is to try every possible 128-bit key (i.e., perform an exhaustive key search, also known as a brute force attack)). By contrast, although three key 3DES has a

168-bit key, there is a “shortcut” attack on 3DES that is comparable, in the number of required operations, to performing an exhaustive key search on 112-bit keys. However, unlike exhaustive key search, this shortcut attack requires a lot of memory. Assuming that such shortcut attacks are not discovered for the AES algorithm, the uses of the AES algorithm may be more appropriate for the protection of high-risk or long-term data.

- The smallest AES key size is 128 bits; the recommended key size for 3DES is 168 bits. The smaller key size means that fewer resources are needed for the generation, exchange, and storage of key bits.
- The AES block size is 128 bits; the 3DES block size is 64 bits. For some constrained environments, the smaller block size may be preferred; however, the larger AES block size is more suitable for cryptographic applications, especially those requiring data authentication on large amounts of data.

See http://www.nist.gov/public_affairs/releases/g00-176.htm for a press release describing NIST’s position on the two algorithms.

With a block cipher algorithm, the same plaintext block will always encrypt to the same ciphertext block whenever the same key is used. If the multiple blocks in a typical message were to be encrypted separately, an adversary could easily substitute individual blocks, possibly without detection. Furthermore, data patterns in the plaintext would be apparent in the ciphertext. Cryptographic modes of operation have been defined to alleviate these problems by combining the basic cryptographic algorithm with a feedback of the information derived from the cryptographic operation.

FIPS 81, DES Modes of Operation, defines four confidentiality (encryption) modes for the DES algorithm specified in FIPS 46-3: the Electronic Codebook (ECB) mode, the Cipher Block Chaining (CBC) mode, the Cipher Feedback (CFB) mode, and the Output Feedback (OFB) mode.

SecureZIP for i5/OS uses Cipher Block Chaining for data encryption.

RC4

The RC4 algorithm is a stream cipher designed by Rivest for RSA Security. It is a variable key-size stream cipher with byte-oriented operations. The algorithm is based on the use of a random permutation. Analysis shows that the period of the cipher is overwhelmingly likely to be greater than 10^{100}. Eight to sixteen machine operations are required per output byte, and the cipher can be expected to run very quickly in software. Independent analysts have scrutinized the algorithm and it is considered secure.

RC4 is used for secure communications, as in the encryption of traffic to and from secure web sites using the SSL protocol.

Key Management

The proper management of cryptographic keys is essential to the effective use of cryptography for security. Keys are analogous to the combination of a safe. If the combination becomes known to an adversary, the strongest safe provides no security against penetration. Similarly, poor key management may easily compromise strong algorithms. Ultimately, the security of information protected by cryptography directly

depends on the strength of the keys, the effectiveness of mechanisms and protocols associated with keys, and the protection afforded the keys.

Cryptography can be rendered ineffective by the use of weak products, inappropriate algorithm pairing, poor physical security, and the use of weak protocols. All keys need to be protected against modification, and secret and private keys need to be protected against unauthorized disclosure. Key management provides the foundation for the secure generation, storage, distribution, and destruction of keys. Another role of key management is key maintenance, specifically, the update/replacement of keys.

Further information is available on key management at the NIST Computer Security Resource Center web site: http://csrc.nist.gov/CryptoToolkit/tkkeymgmt.html

Passphrases and PINS

FIPS 112, *Password Usage*, provides guidance on the generation and management of passphrases (passwords) that are used to authenticate the identity of a system user and, in some instances, to grant or deny access to private or shared data. This standard recognizes that passphrases are widely used in computer systems and networks for these purposes, although passphrases are not the only method of personal authentication, and the standard does not endorse the use of passphrases as the best method.

The passphrase used to encrypt a file with SecureZIP may be from 1 to 260 characters in length. Different passphrases may be used for various files within a ZIP archive, although only one passphrase may be specified per run.

The passphrase is not stored in the ZIP archive and, as a result, care must be taken to keep passphrases secure and accessible by some other source.

Recipient Based Encryption

Passphrase-based encryption depends on both the sender and receiver knowing, and providing intellectual input (the passphrase) in clear text. The passphrase is used to derive a binary master session key for each decryption run. No key information is kept within the ZIP archive, so both parties must retain the passphrase in an external location.

Recipient-based encryption provides a means by which the master session key (MSK) information can be hidden, protected, and carried within the ZIP archive. This is done by using technique known as digital enveloping with public key encryption. The technique requires that the creating process have a copy of the recipient's public key digital certificate, which is used to protect and store the MSK. In addition, the receiving side must have a copy of the recipient's private key digital certificate. With these two pieces of information in place, there is no need for users to retain or recall a passphrase for decryption.

Integrity of Public and Private Keys

Public and private keys must be managed properly to ensure their integrity. The key owner is responsible for protecting private keys. The private signature key must be

kept under the sole control of the owner to prevent its misuse. The integrity of the public key, on the other hand, is established through a digital certificate issued by a certificate authority that cryptographically binds the individual's identity to his or her public key. Binding the individual's identity to the public key corresponds to the protection afforded to an individual's private signature key.

A PKI includes the ability to recover from situations where an individual's private signature key is lost, stolen, compromised, or destroyed; this is done by revoking the digital certificate that contains the private signature key's corresponding public key. The user then creates or is issued a new public/private signature key pair, and receives a new digital certificate for the new public key.

The certificate authority (CA) plays a critical role in ensuring the integrity of public keys in the PKI. Upon being presented with proper evidence of identity (usually through a separate entity called a *registration authority*), the CA issues a digital certificate which contains the applicant's public key, identity, and other information (such as duration of the certificate), all signed by the CA's private signature key. The certificate may then be distributed or placed in publicly available databases, called *repositories*.

Data Encryption

SecureZIP for i5/OS security functions include strong encryption tools using RSA BSAFE and the PKWARE implementation of the Advanced Encryption Standard. ***SecureZIP for i5/OS*** provides the option for passphrase encryption using DES, RC4, 3DES and AES.

RSA *High-Quality Security* - RSA Security submits its Crypto-C products for FIPS 140 testing and validation. FIPS 140-1 and FIPS 140-2 are U.S. Government standards which specify the security requirements to be satisfied by a cryptographic module. RSA Security supports this testing and certification with over 20 years of experience in the security industry.

SecureZIP for i5/OS uses a multi-layer key generation process, based on a user-specified passphrase of up to 260 characters, and/or a user's digital certificate, that creates a unique internal key for each file being processed. In addition, the same passphrase will result in a different system generated key for each file.

SecureZIP for i5/OS also implements the use of Cipher Block Chaining (CBC) to further enhance industry standard encryption algorithms. This feature ensures that each block of data is uniquely modified, further protecting the data from fraudulent access.

SecureZIP for i5/OS encryption is activated through the use of the PASSWORD and ENTPREC parameters. If a value is present for either setting, whether through commands or default settings, then encryption will be attempted in accordance with other settings (for example, -ADVCRYPT); however, if ADVCRYPT(*NONE) is specified, then encryption will be bypassed.

Operating System Levels

V5R1M0 or above is required to run certificate-based operations.

Windows Compatibility

When using BSAFE AES encryption with recipients, there is a cross-system compatibility issue to be addressed by the user community. Windows operating systems running pre-Windows XP may experience a decryption problem depending on the state of the private-key certificate on the workstation. During the Windows certificate import process, a dialog check-box "Mark the private key as exportable" may be selected. If this option was not selected, then Windows will not allow an AES encrypted file to be decrypted unless the master session key was wrapped with 3DES.

The setting of the parameter is enterprise wide and is set using the PKCFGSEC command. When turned on, the MSK3DES flag is set in the NDH/DIB; indicating that the master session key information is protected with 3DES when recipients are specified.

PKZIP for Windows has a variance in processing for 6.0 and 7.x due to OAEP processing. PKZIP for Windows 5.0 through 6.0 used OAEP processing. However, that was found to be incompatible with SmartCards, so 6.1 and above began setting the NO_OAEP flag in the NDH/DIB flags and stopped creating OAEP encryption-mode files.

SecureZIP for i5/OS will always set NO_OAEP, therefore PKZIP for Windows 5.0 - 6.0 will not be able to read recipient-based files from the large platforms.

SecureZIP for i5/OS should be able to detect whether the NO_OAEP flag is set and successfully extract either. No change in logic is required within the SecureZIP high-level code, but the low-level EVTCERTD code should handle the switch based on the flag.

What is Filename Encryption?

Someone who cannot decrypt the contents of an archive may still be able to infer sensitive information just from the unencrypted names of files. To prevent this, you can encrypt the names of files in addition to their contents. Encrypted file names can be viewed in the clear—that is, unencrypted—only when the archive is opened by an intended recipient, if the archive was encrypted using a recipient list, or by someone who has the passphrase, if the archive was encrypted using a passphrase.

SecureZIP for i5/OS encrypts file names using your current settings for (strong) encryption method and algorithm. File names can be encrypted using either strong passphrase encryption or a recipient list (or both). You must use one of the strong encryption methods: you cannot encrypt file names using traditional, ADVCRYPT(ZIPSTD), which uses a 96-bit key.

Encrypting names of files and folders in an archive encrypts and hides a good deal of other internal information about the archive as well. To encrypt file names, ***SecureZIP for i5/OS*** encrypts the archive's central directory, where virtually all such metadata about the archive is stored. Be aware, however, that archive comments are not encrypted even when you encrypt file names. Do not put sensitive information in an archive comment.

User Encryption Examples

Below are examples of how to invoke encryption processing using PKZIP commands.

Zip Compress File(s) and Write to an Archive File

This is the main PKZIP compression screen. Here you specify the method and mode of encryption.

```
                        File Compression      (PKZIP)

Type choices, press Enter.

Archive Zip File name  . . . . .   '/yourpath/encryption/as400.des3.zip'

List Include file or pattern . .   '/yourpath/encryption/*.txt'
               + for more values

Type of processing . . . . . .   *ADD          *ADD, *UPDATE, *FRESHEN ..
Compression Level  . . . . . .   *SUPERFAST    *FAST, *NORMAL, *MAX...
File Types . . . . . .           *DETECT       *DETECT *TEXT *BINARY ........
Advance  Encryption :
               Method  . . . . . > 3des         ZIPSTD, AES128, AES192...
               Mode  . . . . . . > BSAFE        PKWARE, BSAFE

                                                                   More...
F3=Exit   F4=Prompt   F5=Refresh   F10=Additional parameters   F12=Cancel
F13=How to use this display        F24=More keys
Parameter ARCHIVE required.
```

Placing the cursor on the “Method” and hitting F4 presents the next screen that allows you to select one of the encryption methods to use.

```
                  Specify Value for Parameter ADVCRYPT

Type choice, press Enter.
             Method  . . . . . > 3DES

  ZIPSTD
  AES128
  AES192
  AES256
  3DES
  DES
  RC4_128
```

When the next screen appears if you do not enter a passphrase no encryption processing is completed on the file(s) to be archived. If you desire encryption, you must enter the passphrase twice; once in the Archive Passphrase and again in the Verify Passphrase.

```
                        File Compression      (PKZIP)

Type choices, press Enter.

Archive Passphrase . . . . . .

Verify  Passphrase . . . . . .

Archive File Type  . . . . . .   *ifs          *DB, *IFS
Files to Zip Type  . . . . . .   *ifs          *DB, *IFS, *IFS2, *DBA, *SPL
Before/After Selection . . . . .   *NO          *NO, *BEFORE, *AFTER
Date for Selection . . . . . .   0             Date mmddyyyy
File Name actions  . . . . . .   *SUFFIX       *NONE, *DROP, *SUFFIX
External Conversion Flags  . . .   *NONE        Character value, *NONE
Create Self Extract Archive  . .   *MAINTAIN    *MAINTAIN, WINDOWS, AIX...
                                                                  Bottom
F3=Exit   F4=Prompt   F5=Refresh   F10=Additional parameters   F12=Cancel
F13=How to use this display        F24=More keys
```

Following is the output of the PKZIP run.

```
Scanning files in *IFS for match  ...
Found  2 matching files
Compressing /yourpath/encryption/appnote.txt in BINARY mode
Add  /yourpath/encryption/appnote.txt  --  Deflating (69%)  encrypt(BSAFE 3DES)
Compressing /yourpath/encryption/readme.txt in BINARY mode
Add  /yourpath/encryption/readme.txt  --  Deflating (58%)  encrypt(BSAFE 3DES)
PKZIP Compressed 2 files in Archive /yourpath/encryption/as400.des3.zip
PKZIP Completed Successfully
```

Commands generated from the PKZIP screen using the retrieve key after the PKZIP run.

```
                                Command Entry                          COSMOS
                                                          Request level:   4
 Previous commands and messages:

   (No previous commands or messages)



                                                                        Bottom
 Type command, press Enter.
 ===> PKZIP ARCHIVE('/yourpath/encryption/as400.des3.zip')
FILES('/yourpath/encryption/*.txt') ADVCRYPT(3DES BSAFE) PASSWORD() VPASSWORD()
TYPARCHFL(*IFS) TYPF
```

Display the contents of an Archive File

When the files within an archive have strong encryption the “!” (bang) character is placed in front of the file name to inform you that you must have the correct passphrase to view or extract the file.

```
                       File Extraction     (PKUNZIP)

Type choices, press Enter.

Archive Zip File name  . . . . .   '/yourpath/encryption/as400.des3.zip'

List Include file or pattern . .   *ALL
              + for more values

Type of processing . . . . . .   *VIEW         *VIEW, *EXTRACT, *NEWER...
```

```
File Types . . . . . .   *DETECT       *DETECT *TEXT *BINARY ...


   Archive:  /yourpath/encryption/as400.des3.zip   33451 bytes   2 files
     Length  Method    Size  Ratio  Date   Time  CRC-32    Name
    -------- ------  ------- -----  ----   ----  ------    ----
       97182 Defl:F    30230  69% 01-30-04 13:16 223c2ea4 !/yourpath/encryption/
   appnote.txt
        5747 Defl:F     2710  53% 10-06-03 15:14 d193af9b !/yourpath/encryption/
   readme.txt
    --------         -------  ----                            -------
     102929            32940  68%                             2 files
   PKUNZIP extracted     0 files
   PKUNZIP Completed Successfully
```

Incorrect Passphrase Use

The following example shows the program's response to an incorrect passphrase. The error message indicates that the file(s) were skipped because of an incorrect passphrase and that PKUNZIP completed with errors.

```
   PKUNZIP Archive: /yourpath/encryption/as400.des.zip
   Archive Comment:"PKZIP for i5/OS by PKWARE"
   Searching Archive /yourpath/encryption/as400.des.zip  for files to extract
   skipping: /yourpath/encryption/appnote.txt  incorrect passphrase
   skipping: /yourpath/encryption/readme.txt  incorrect passphrase
   Caution:  zero files tested in /yourpath/encryption/as400.des.zip.
   2 file(s) skipped because of incorrect passphrase
   PKUNZIP Completed with Errors
   Press ENTER to end terminal session.
```

3 ZIP Files

A ZIP archive is the storage facility for files that are compressed (or simply stored) using the ***PKZIP*** product. The basic archive can hold up to 65,535 files, which may have been compressed by up to 99% of their original size. Data integrity is validated by a cyclic redundancy check (CRC) to maintain integrity of the data from the compression through the extraction process. If the archive contains the ZIP64 archive format, the archive can support more than the 65,535 files and can be larger than 4 GB (see "Large Files Considerations" in chapter 1).

In addition to the data, file attributes are retained, allowing extraction of the same file characteristics without the need of control card specifications. An archive can exist in three possible states during processing, described as "old archive," "temporary archive," and "new archive." An explanation of the functions of each of these is described in the sections below.

A ZIP archive is transferable between platforms. That is, files that are compressed by ***PKZIP*** on one platform may be extracted by ***PKZIP*** on a different platform, maintaining identical data.

This chapter describes the types of files used by ***PKZIP^{i}*** and provides a description of the way in which they are accessed by ***PKZIP^{i}*** ZIP archives.

PKZIP^{i} (by default) creates a new archives in the *DB file system as members of PF-DTA files with 132-byte records. The archive file is given a text field of "file created by ***SecureZIP for i5/OS***" or "file created by ***PKZIP for i5/OS***". The archive member is given a text field of "Member created by ***SecureZIP for i5/OS***" or "Member created by ***PKZIP for i5/OS***". If you wish to create your own archive (perhaps to have a larger record size, for performance), then you can do so, but try to adhere to the following:

- When you create the file, do not create any members in it.
- After having created the file, change the MAXMBRS parameter for the file from 1 to *NOMAX.

A ZIP archive holds files internally in one of several formats, which are compatible with other platforms supported by ***PKZIP***. These formats are described here, and several commands are available for transforming files into one of these formats as they are compressed. You may specify in which format a file is stored using the FILETYPE(*BINARY) or FILETYPE(*TEXT) command parameters. OS/400 SAVF are always stored as *BINARY type. If you do not specify FILETYPE(*BINARY) or

(*TEXT), then the PKZIP and PKUNZIP programs both will default to FILETYPE(*DETECT). For more information, see FILETYPE(*DETECT).

"Old" ZIP Archive

Starting with ***PKZIP[i]*** Version 8.2, an optional input archive can be specified that can be a different name than the archive that will be created for an output archive file. If this is present, it is considered to be the "Old" ZIP archive. Otherwise the first ARCHIVE parameter is considered to be the "Old" ZIP archive.

The new input archive parameter (2nd option of ARCHIVE) allows the ability to preserve the input archive and create a new archive with a different name. This would allow the new archive to take on new attributes such as FNE or non FNE archive. The one requirement is that both archives must reside on the same file system such as IFS or the QSYS Library file system.

When there is not inputted archive, the 1st option of the ARCHIVE parameter for PKZIP is known as the old ZIP archive, except when the TYPE(*ADD) parameter is being used to create a new ZIP archive. The old ZIP archive may have been created by ***PKZIP[i]*** during an earlier operation or may have been created by ***PKZIP*** on another platform and transferred from there. When a ZIP archive is being updated (or when PKUNZIP is extracting files from a ZIP archive), the necessary details are taken from the old ZIP archive. It should be noted that when ***PKZIP[i]*** is updating a ZIP archive, it takes the necessary data from the old ZIP archive, merges it with any new data, and transfers it to a new ZIP archive (in a temporary member in the same iSeries file as the old archive). When all updating is completed, ***PKZIP[i]*** deletes the old ZIP archive and then renames the new ZIP archive to the same name as the old ZIP archive. For this reason a file containing a ZIP archive should allow for at least one temporary member to be allocated. When ***PKZIP[i]*** creates an archive file, it uses MAXMBRS(*NOMAX).

"Temporary" Archive File

A temporary archive file refers to an archive work in progress. ***PKZIP[i]*** will always use a temporary archive file and its definition depends on the file system. If the file system type is IFS, then the temporary archive file will be in the same **directory** of the specified new archive. If the file system type is QSYS, the temporary archive file will become a **member** of the specified archive file. The temporary file or member will have a unique name PKnnnnnnnn (where nn represents an internal random number). When the file has been completed successfully, the temporary name will be renamed to the specified name in the ARCHIVE parameter. If this is a process in which an old archive is being updated, then (if successful) the old archive will be deleted before the rename. If a problem occurs, the temporary archive may stay with the temporary name. View the job log if this happens to determine the status of the archive.

“New” ZIP Archive

When the processing of the temporary dataset is finalized, ***PKZIP^{i}*** creates a new ZIPPED archive that is the modified “after” version of the old archive. The modified name of the old archive and specified allocation information is transferred automatically to the new archive after updating, and the old Archive is deleted. A new ZIP archive is created when an old ZIP archive is updated, or when a TYPE(*ADD) parameter (see chapter 7) is used with ***PKZIP^{i}*** where there is no old ZIP archive.

Self-Extracting Archive

The self extracting programs are held as binary entities in the file PKZIPSFX of the ***PKZIP^{i}*** library. The appropriate member is loaded and the executable data copied to the beginning of the Archive as a preamble when requested.

The resulting archive can still be processed by ***PKZIP^{i}*** as a normal ZIP Archive.

When an input archive containing a self-extraction preamble is passed to ***PKZIP^{i}*** for PKZIP processing and no value is supplied by SELFXTRACT, the default of *MAINTAIN will keep any preamble if one exist. If the parameter SELFXTRACT(*REMOVE) is supplied then the PREAMBLE is removed when writing the new archive.

A self-extracting archive can be created from an existing archive by using SELFXTRACT with a valid self-extractor. If the original archive contained a preamble, it will be removed and the newly specified preamble will be inserted.

When transferring a self-extracting archive to a target system, be sure to transfer the archive in binary format and adhere to requirements for executables in that environment. (For example, a Windows program should be saved with an application extension of EXE, and a UNIX file attribute should have executable authorization set via the UNIX chmod command).

The self-extraction programs provided are at the 2.5 level of PKZIP. As such, the following restrictions apply to the operation of the self-extraction program(s). Care should be taken to control the creation of the self-extracting archive within these restrictions, although the resulting archive may still be processed with PKZIP programs at higher levels that support these features.

- The number of files in the archive should be limited to 65,535 or less.
- Strong encryption is not supported.
- The size of the archive should not exceed 2 gigabytes: Most Windows and UNIX systems will not load executables larger than 2 GB.
- The uncompressed size of individual files should be less than 2 gigabytes for target systems which are Windows or LINUX (less than 4 gigabytes for other UNIX systems).

To assist in the usage of the self-extraction programs on the target systems, some of the command parameters are listed below. Note that some parameters may not be valid on all systems. By executing the transferred self-extracting archive on the target system with “-help”, the commands syntax appropriate to that system will be displayed.

```
Usage: sfx.exe [options] [.ZIP archive] [files...]

Where sfx.exe = the name of the self-extracting executable file

Options:
after        extract files that are newer than or equal to a specified date
               suboptions:
                  "date specification" [format: mmddyy or mmddyyyy]
               e.g.: sfx.exe -aft=12311999 file.zip

before       extract files that are older than a specified date
               suboptions:
                  "date specification" [format: mmddyy or mmddyyyy]
               e.g.: sfx.exe -bef=12311999 file.zip

console      display the contents of specified archived files on your screen
               e.g.: sfx.exe -con= file.zip readme.txt

directories  recreate directory path while extracting including any
                  sub-directories
               e.g.: sfx.exe -dir file.zip

exclude      exclude specified files from being extracted
               e.g.: sfx.exe -exc=*.txt file.zip

extract      extract files from the .ZIP archive
               suboptions:
                  all [extract everything in archive]
                  freshen [extract if newer than destination copy]
                  update [extract if newer or not in destination directory]
               e.g.: sfx.exe -ext=all file.zip

help         display help screen
               e.g.: sfx.exe -help

Id         preserve original file uid/gid. Must be root/file owner (UNIX only)

include      include specified files for extraction
               e.g.: sfx.exe -inc=*.txt file.zip

larger       extract files that are the specified size (in bytes) and larger
               suboptions:
                  a numerical value (in bytes) that indicates a minimum desired
                  file size
               e.g.:sfx.exe -larger=400

license      displays license information
               e.g.: sfx.exe -lic

locale       reads and/or adjusts the locale variable for date and time format
             input
               suboptions:
                  environment [read system variable and apply accordingly]
                  "valid country name" [for example locale=germany]
               e.g.: sfx.exe -loc=us -aft=12311999 file.zip

lowercase    change filenames to lower case on extraction
               e.g.: sfx.exe -lowercase

mask         remove specified file attributes upon extraction
               suboptions:
                  archive [mask archive attribute from file(s)/folder(s)]
                  hidden [mask hidden attribute from file(s)/folder(s)]
                  system [mask system attribute from file(s)/folder(s)]
                  readonly [mask read-only attribute from file(s)/folder(s)]
                  none [do not mask attributes from file(s)/folder(s)]
                  all [mask all attributes from file(s)/folder(s)]
               e.g.: sfx.exe -mask=archive,readonly file.zip

more         display output one screen at a time
               e.g.: sfx.exe -more file.zip
```

```
newer         process only those files that are newer than a specified
              (calendar) day in the past
                 suboptions:
                   a numerical value (in calendar days) that indicates some
                   date in the past relative to the current date
                 e.g.: sfx.exe -newer=2

noextended    suppress the extraction of extended attributes
                 e.g.: sfx.exe -noex file.zip

older         process only those files that are older than a specified
              (calendar) day in the past
                 suboptions:
                   a numerical value (in calendar days) that indicates some
                   date in the past relative to the current date
                 e.g.: sfx.exe -older=2

overwrite     overwrite existing files
                    prompt [prompt before overwriting]
                    all [always overwrite]
                    never [never overwrite]
                 e.g.: sfx.exe -o=all file.zip

password      specify a decryption passphrase
                 e.g.: sfx.exe -pass=grendel file.zip

print         print the specified archived file
                 suboptions:
                    "print device name" [for example print=lpt1]
                 e.g.: sfx.exe -print=lpt2 file.zip readme.txt

silent        suppress warning messages when extracting
                 e.g.: sfx.exe -silent file.zip

smaller       extract files that are the specified size (in bytes) and
              smaller
                 suboptions:
                    a numerical value (in bytes) that indicates a maximum desire
                    file size
                 e.g.:sfx.exe -smaller=400

sort          sort files when extracting
                 suboptions:
                    crc [sort by crc value]
                    date [sort by date of the file]
                    extension [sort by file extension]
                    name [sort by file name]
                    natural [sort in the order that the file was archived]
                    ratio [sort by compression ratio]
                    size [sort by file size]
                    none [do not sort]
                 e.g.: sfx.exe -sort=size file.zip

test          test the integrity of archived files
                 suboptions:
                    all [test everything in archive]
                    freshen [test if newer than destination copy]
                    update [test if newer or not in destination directory]
                 e.g.: sfx.exe -test=all file.zip

times         preserve specified file date/time stamp
                 suboptions:
                    access [preserve accessed date/time stamp on extraction]
                    modify [preserve modified date/time stamp on extraction]
                    create [preserve created date/time stamp on extraction]
                    all [preserve all date/time stamps on extraction]
                    none [do not preserve date/time stamps on extraction]
                 e.g.: sfx.exe -time=access,modify file.zip

translate     translate the end of line sequence for give operating system
```

```
                suboptions:
                   DOS [convert to DOS style line endings]
                   MAC [convert to MAC style line endings]
                   unix [convert to unix style line endings]
                e.g.:sfx.exe -translate=unix

version      display SFX version and return appropriate value to the shell
                suboptions:
                   major [return major version number]
                   minor [return minor version number]
                   step [return step or patch version number]
                e.g.: sfx.exe -ver=step

volume       restore the volume label when extracting
                e.g.: sfx.exe -vol file.zip

warning      prompt to continue after warning message
                e.g.: sfx.exe -warn file.zip
```

Data Format - Text Records vs. Binary Records

Binary data is stored in a ZIPPED archive in its original format. Binary data may be graphics or numbers that are already in "computer format." Therefore, no translation is done, and EBCDIC will remain EBCDIC. The length of binary records in UNZIP processing is determined by the archive's fixed-length records. ***PKZIP***^{i} will fill the available block automatically according to allocation specifications.

In the context of ZIPPED archives, a "text file" is one that is stored in the ASCII format. A text file contains records of data, each separated by a delimiter to signify the end of the record.

Note: An EBCDIC file containing text information (such as source code) can be stored in its original format by using BINARY, but it is not considered to be a "text" file within the ZIP architecture.

PKZIP^{i} uses the default line delimiter CR-LF (X'0D0A') at the end of each text record. Text file members in the QSYS library file system use new line characters (NL=X'15') internally. ***PKZIP***^{i} will handle the CR-LF and NL in both extraction and compressions automatically.

At the time of PKUNZIP file extraction, ***PKZIP***^{i} will convert text data from ASCII to EBCDIC by using a translation table. During installation, several translation tables are available, and the customization process will select one of the translation tables as a default. Additional translation tables may be created through the customizing procedure.

Situations may arise in unique platform interchanges, or when working with text files from other countries where the default text translation table is not adequate. Users may select any available translation table by using TRAN and FTRAN parameters.

PKZIP^{i} extracts text records stored in the ZIP archive by examining data for record delimiter and file terminator indicators. Using these indicators, ***PKZIP***^{i} aligns records in accordance with target file attributes.

Text files (such as program source code) are held within an archive using the ASCII character set for compatibility with other versions of ***PKZIP®***. For these to be usable on OS/400, they must be converted to the IBM EBCDIC character set. Additionally, the carriage return and line feed characters must be removed before writing lines to

a file because OS/400 files are record-based and do not use control characters to separate records or lines. Text files usually have spaces at the end of a line. When using the text file handlers, ***PKZIP**i* has less data to read because the input/output routines remove trailing spaces and replace them with a new line character. This improves ***PKZIP**i* performance.

When extracting files from an archive, ***PKZIP**i* must know whether to perform text conversions. ***PKZIP**i* stores an indicator in the archive file's local header defining if a file is binary or text-based. Because this indicator may be wrong in some circumstances, use the FILETYPE keyword to specify whether text conversions are required. When adding files to an archive, ***PKZIP**i* will flag the file according to the FILETYPE used.

***PKZIP**i* uses translation tables that should be suitable for most customers, but some users may wish to alter the tables. The procedure for changing the translation tables is discussed. If text files are only used on iSeries, then the FILETYPE(*EBCDIC) may be used. This uses iSeries files "as is" for the file (which are faster for text files), but does not translate the data to ASCII. This will provide a small improvement in performance.

Additionally, ***PKZIP**i* will translate each character in a text file from EBCDIC character format to ASCII character format by default. This is done using one of the two internal translation tables, which are named UKASCII and USASCII. It is recognized that these translation tables may not suffice for all countries or all situations, especially on those sites where text files are received from several different countries for processing into a single format. The source of the translation tables used by the PKZIP and PKUNZIP programs has been supplied, together with instructions for modifying the tables to create additional files (see Appendix D for details). This enables sites to modify the translation table as required.

In a case where FILETYPE is neither *TEXT nor *BINARY, *DETECT is the default mode. ***PKZIP*** will read up to 64K of data from the input file and scan it for non-translatable text characters using the active text translation table. If any characters will not translate successfully using this method, the entire file will be treated as if *BINARY has been used.

Note: One exception to this is X'00' or the NULL terminator character, which is commonly used in C language. The NULL character will be allowed within the files. If file type is of a file in the archive is unknown whether it is text or binary, the user may use the TYPE(*VIEW) and VIEWOPT(*DETAIL) parameters to examine the file attributes.

File Attributes

Within each ZIP archive there are two different directories providing information about the files held in that archive. A local directory is included at the front of each file, with information pertaining to each file (for example: file size and date ZIPPED), and a central directory is located at the end of the ZIP archive. The central directory lists the complete contents of the ZIP archive and is the primary source of information for UNZIP processing.

***PKZIP**i* stores extended attributes about the file that can be useful in recreating the file during UNZIP processing. See the *System Administrator's Guide*.

PC Shared Drives Format

One common mistake made when extracting a text file to a shared drive folder in the IFS where the file will be used by a Windows application is to extract the file in text mode. Extracting a file as a TEXT file on the iSeries will cause PKUNZIP to translate the file to the EBCDIC format since this is the native iSeries format. The Windows application expects the file to be in ASCII, so therefore this file should be extracted using binary, since the files are stored in ASCII in the archive.

4 File Extraction Process

Extracting Files to the QSYS Library File System

When extracting files to the QSYS library file system using TYPFL2ZP(*DB), some items to consider are 1) does the file exist or will a new file be create?, 2) did the file come from ***PKZIP[i]*** or did it come from another platform?, 3) are the files text type files from another platform where I need to know the record length, etc. These are just a few questions that might impact how you want to extract the files.

If the file does not exist and if the file did not come from ***PKZIP[i]***, you should provide a record length for the file with the parameter DFTDBRECLN. If the file is coming from ***PKZIP[i]*** or ***PKZIP for zSeries*** the record length will be in the extra data. If the file is from ***PKZIP[i]*** and the parameter was DBSERVICE(*YES), the complete database definition from the extract data for database will be used to create the file.

If the file is to be created as text and the record length is too short then you will receive messages indicating the records are being truncated.

Two common parameters that are used to alter or guide the extraction process are the EXDIR and DROPPATH parameters. EXDIR provides the path library or library/file that the file will be extracted to when no library or path exist for the files in the archive. Of course, this is where the DROPPATH comes in to drop the first path or library with *LIB or to remove all paths in a name with *ALL.

For example, files in an archive might look like this:

```
Archive/#1:
My Document/myfiles/test/myheader.txt
My Document/myfiles/test/mydata.txt
My Document/myfiles/test/mytrailer.txt

Archive/#2:
    QGPL/QCLSRC/MYCL01
    QGPL/QCLSRC/MYCL02
    QGPL/QCLSRC/MYCL03
    QGPL/QCLSRC/MYCL04
```

In archive #1 let's assume that all three text files are of different records lengths. If we want to extract each with their own length, we would have to make three runs to

create the files with different parameters. Or we could use CRTPF and create each of the files so the files would exist with the proper record length.

➔ PKUNZIP ARCHIVE('Archive/#1') FILES('My Document/myfiles/test/myheader.txt') TYPE(*EXTRACT) EXDIR('MYLIB/MYHEADER') DROPPATH(*ALL) DFTDBRECLN(50) CVTTYPE(*DROP)

➔ PKUNZIP ARCHIVE('Archive/#1') FILES('My Document/myfiles/test/mydata.txt') TYPE(*EXTRACT) EXDIR('MYLIB/MYDATA') DROPPATH(*ALL) DFTDBRECLN(150) CVTTYPE(*DROP)

➔ PKUNZIP ARCHIVE('Archive/#1') FILES('My Document/myfiles/test/myheader.txt') TYPE(*EXTRACT) EXDIR('MYLIB/MYTRAILER') DROPPATH(*ALL) DFTDBRECLN(20) CVTTYPE(*DROP)

The commands above would create three files in library MYLIB, with all files having different record lengths.

Now suppose the files already exist with the names and record lengths. In this case, we could do all three files at once with:

➔ PKUNZIP ARCHIVE('Archive/#1') TYPE(*EXTRACT) EXDIR('MYLIB/?MBR') DROPPATH(*ALL) CVTTYPE(*DROP) OVERWRITE(*YES)

The MBR will force each member name to also become the file name.

In archive #2, let's assume that we want to extract the CL source member and place them in a different library call MYNEWLIB. If the QCLSRC file does not exist and the archive was not built with DBSERVICE(*YES), then you would need to do a

➔ CRTSRCPF FILE(MYNEWLIB/QCLSRC)

to have the file setup correctly for source files. If the QCLSRC file already exist in MYNEWLIB or the archive was built with DBSERVICE(*YES), then no special handling is required.

Authority Settings

When extracting files into the QSYS library file system, whether the files came from the AS/400 or another platform, the authorities are not taken from the archive, but from the user's current environment settings. The file's authority is not stored in the archive.

If a library is required to be created, PKUNZIP would create the library as if the current user was issuing a CRTLIB command. For standard settings, it might create the library with the following authority settings:

```
              Data     --Object Authorities--
User         Authority  Exist  Mgt  Alter  Ref
*PUBLIC      *RWX
USER         *RWX         X     X     X     X.
```

If the file does not exist, PKUNZIP will be required to create the file. If the file does exist no authorities are changed. If the file is created, the authority will be the same as if the user was issuing a CRTPF command in their environment. For most standard settings, it would create the file with the following authority settings:

```
             Data      --Object Authorities--
User        Authority  Exist  Mgt  Alter  Ref
*PUBLIC     *RWX
MYOWNER     *RWX         X     X     X     X
```

Extracting Files to the IFS

When extracting files to the integrated file system with TYPFL2ZP(*IFS), record lengths are not a concern as they were in the QSYS library file system. The main considerations when extracting to the IFS is “what paths do you want for the file, or should the file be stored in EBCDIC or ASCII”.

Path Considerations

If the name of files in the archive, starts with ‘/’, then with no other changes this will be extracted to the root of the system with the first name in the path. This form of pathname is called a fully qualified path.

If the name does not start with a ‘/’, the item will be extracted to the paths based on the current directory (DSPCURDIR). This form of pathname is called a relative path.

In both cases if the path(s) does not exist, the path(s) will be created with the attributes of the parent folder.

Changing the path(s)

In cases where the path that is stored with a name of the file in archive is not desired, then using the EXDIR and DROPPATH parameters should help guide the file to where it should be placed.

Using EXDIR, you can define the path of the file(s) that will be extracted. If you need to remove the path of the file in the archive, you can use DROPPATH(*ALL) to remove all the paths before extracting or you can use DROPPATH(*LIB) to remove only the first path name.

Again the coding of EXDIR follows the same rule with regards to fully qualified path or relative path.

File Type Considerations

When extracting a file, the decision to whether the contents of file should be stored in ASCII or EBCDIC needs to be made.

If the file is not a text file, it does not matter and should be stored as binary. If the file is text, and will be used by a PC program, chances are the data is expected to be in ASCII. Since the files are stored in the archive as ASCII, these files should be extracted as TYPEFILE(*BINARY). If the file is to be used by an AS/400 application or will be translated later, then chances are the file should be stored in EBCDIC. In this case use TYPEFILE(*TEXT) to extract the file in EBCDIC.

Authority Settings

If directories are required to be created during the extraction, the authority settings will be created according to the create directory definitions of the DTAAUT(*INDIR) and OBJAUT(*INDIR) parameters.

The authority for the directory being created is determined by the directory it is being created in. The directory immediately preceding the new directory determines the authority. A directory created in the root is assigned the public authority given to objects in the root directory. A directory created in QDLS for a folder defaults to *EXCLUDE for a first level folder. If created in the second level or greater, the authority of the previous level is used. The QOpenSys and root file systems use the parent directory's DTAAUT value.

The object authority is based on the authority for the directory where this directory is being created.

For IFS files, the access permissions flags of the file are captured. For example:

- S_IRUSR - Read permission for the file owner
- S_IWUSR - Write permission for the file owner
- S_IXUSR - Search permission (for a directory) or execute permission (for a file) for the file owner
- S_IRGRP - Read permission for the file's group
- S_IWGRP - Write permission for the file's group
- S_IXGRP - Search permission (for a directory) or execute permission (for a file) for the file's group
- S_IROTH - General read permission
- S_IWOTH - General write permission
- S_IXOTH - General search permission (for a directory) or general execute permission (for a file)

These access permission flags will be set for the owner that is running the PKUNZIP job and not the original owner.

Other user permissions from the parent folder will also be set for the file.

For example, the folder being extracted into has *PUBLIC as *EXCLUDE, the extracted file will also have *PUBLIC as *EXCLUDE.

Extracting zSeries Variable Length Records (RDW/ZDW)

In the zSeries, PKZIP can compress variable length records and store the files known as RDW or ZDW into an archive. The format of these records contains a 4 byte length (store in little Endian) followed by the record itself for that length. These records are stored in binary, therefore EBCDIC.

By using the TYPE(*DETECT), PKZIP will remove the record length before extracting the records. The ending format will differ depending on if the extraction is to a database file or to the IFS.

To extract to a fixed length database file, each record will be extracted and placed in the database forcing each record to be a fixed record in the database with no translation.

To extract to the IFS, each record will have a New Line (NL 0x15) character inserted at the end of each record. The records are still variable in length but with a separator. If the file was an object or load module, the results will be unpredictable.

If a file is extracted with TYPE(*TEXT), the results are unpredictable.

If a file is extracted with type(*BINARY), then the file is extract as is, including the 4 byte length field in front of each record.

Extracting Spool Files

When extracting spool files with PKUNZIP, the attributes at the time of compression, will be preserved except for new spool file numbers. That will be generated. Parameter SPLUSRID is for the user ID on the new extracted spool file. If it is *DFT the original user ID will stay with the new spool file. Parameter SFQUEUE is for the OUTYQ and OUTQ library that the new extracted spool file will be placed. If *DFT is specified then the original OUTQ will be used to place the spool file.

Note on extracting Spool Files: To create or extract spool file with PKUNZIP, the user must have *USE authority to the API QSPCRTSP. The normal setting for the API QSPCRTSP is Authority PUBLIC(*EXCLUDE). The API authority is set this way so that system administrators can control the use of this API. This API has security implications because you can create spooled file from the data of another spooled file. To allow user to extract spool files change the API authority on a need basis.

When extracting a spool file with PKUNZIP, the new spooled file will be created with attributes based on values taken from the spooled file attributes when PKZIP archived the spool file. The spool file's file number, job, job user, job number, date, and time are controlled by IBM OS/400 operating system during the creation of a spool file.

The new spool file that is created by the PKUNZIP is spooled under one of two jobs and is dictated by IBM's create spool file API. The job is determined by the user-name field from the attributes. If the user name is the current user, it is a part of the user's job and is owned by the user profile that the job was started with. First the user profile for the user name must already exist. When using the user id override (parameter SPLUSRID), the spool file will be now belong to the override user.

If the ownership of the new spooled file is assigned to a different user by a different user profile name in the user-name field from the attributes, then the current user must have *SPLCTL authority to assign the spooled file to another user. When this is done, the new spooled file is by the user specified in the user name field or override parameter. The new spooled file is then part of a special system job (QPRTJOB) that is created for each user.

The new spooled file is placed on the output queue specified in the output queue name field from the original spool file attributes. If the parameter SFQUEUE is used it will override the attribute for the output queue.

In both cases, the spooled file name is the one contained in the spooled file attributes parameter. The spooled file number will be the next sequential one available for the job that the spooled file becomes a part of.

OS/400 authority requirements when extracting spool files:

- *Special Authority* - *SPLCTL. This authority is needed if you are creating a spooled file for another user.
- *Output Queue Authority* -*USE
- *Output Queue Library Authority* - *EXECUTE
- *Object QSPCRTSP API Authority* -*USE

The following are several examples of results of extracting spool files:

Start with archiving the following spool files that were created with job MYJOB1 and user EVWSS:

File	File Nbr	Job	User	Number	Date	Time
QSYSPRT	397	MYJOB1	EVWSS	010893	12/11/02	13:35:29
QSYSPRT	398	MYJOB1	EVWSS	010893	12/11/02	13:36:09
QSYSPRT	399	MYJOB1	EVWSS	010893	12/11/02	13:36:09

Now extract with job MYJOB1 and user EVWSS but now on a different day and job number:

File	File Nbr	Job	User	Number	Date	Time
QSYSPRT	2	MYJOB1	EVWSS	010927	12/12/02	09:57:42
QSYSPRT	3	MYJOB1	EVWSS	010927	12/12/02	09:57:42
QSYSPRT	4	MYJOB1	EVWSS	010927	12/12/02	09:57:42

Next extract with job MYJOB2 and user EVWSS but now on a different day and job number:

File	File Nbr	Job	User	Number	Date	Time
QSYSPRT	2	MYJOB2	EVWSS	010928	12/12/02	09:59:06
QSYSPRT	3	MYJOB2	EVWSS	010928	12/12/02	09:59:06
QSYSPRT	4	MYJOB2	EVWSS	010928	12/12/02	09:59:06

Next, using the user, override with the SPLUSRID(WSS) and submit MYJOB1 with user EVWSS.

File	File Nbr	Job	User	Number	Date	Time
QSYSPRT	26	QPRTJOB	WSS	010118	12/12/02	10:02:50
QSYSPRT	27	QPRTJOB	WSS	010118	12/12/02	10:02:50
QSYSPRT	28	QPRTJOB	WSS	010118	12/12/02	10:02:50

Notice that the job was changed to QPRTJOB since the user being extracted was different than the user running the Job.

Next, signed on as user WSS, submit a job MYJOB11 with the job parameter USER profile specified for user EVWSS.

➔ **SBMJOB CMD(PKUNZIP ARCHIVE('atest/splftst/tst02') TYPE(*EXTRACT)) JOB(MYJOB11) USER(EVWSS)**

File	File Nbr	Job	User	Number	Date	Time
QSYSPRT	2	MYJOB11	EVWSS	010936	12/12/02	10:25:25
QSYSPRT	3	MYJOB11	EVWSS	010936	12/12/02	10:25:25
QSYSPRT	4	MYJOB11	EVWSS	010936	12/12/02	10:25:25

5 i5/OS File Processing Support

***PKZIP*^i^** can support files maintained in both the traditional QSYS library file system and in IFS (integrated file system) along with supporting spool files.

QSYS (Library File System)

The QSYS file system supports the i5/OS library structure. This file system provides access to database files and all other i5/OS object types that the library manages. On the IBM i5/OS system, each QSYS type file (also called a file object) has a description that details the file characteristics and how the data associated with the file is organized into records, and, in many cases, the fields associated for each record. Whenever a file is processed, the i5/OS uses this description.

Of the objects in the library system, ***PKZIP*^i^** will only process physical files that have an attribute type of PF-DTA (physical data files), PF-SRC (physical source file), or SAVF (save files).

QSYS files always exist in a library, and the PF-DTA and PF-SRC files (if data exist) will always have one to many members in the file. Therefore, PF-DTA and PF-SRC files have a name format of “library/file(member).” A SAVF (a special type of i5/OS file for saving and restoring i5/OS objects) does not have any members giving a file format of “library/file.” Because SAVF types are handled in a special way, they are given additional consideration (see SAVF and use of SAVF method).

QSYS Summary

If the archive file is to be in the QSYS library system, set the Archive File Type parameter to TYPARCHFL(*DB). Even though *DB is working with archive files that are in a library file system, the IFS is utilized for performance and for large file support (ZIP64). If you need ***PKZIP*^i^** to use the QSYS library system exclusively, for all processing—for example, to support OS400 features such as Adopt Authority—set the parameter to TYPARCHFL(*XDB).

If the file being compressed or extracted is in the QSYS library system, set parameter TYPFL2ZP(*DB).

If the list files (see Appendix C) are to be in the QSYS library system, set parameter TYPLISTFL(*DB).

Format Summary:

PF-DTA	LIBRARY/FILE(MEMBER)
PF-SRC	LIBRARY/FILE(MEMBER)
SAVF	LIBRARY/FILE

IFS (Integrated File System)

The integrated file system is a part of iSeries which supports stream input/output and storage management similar to personal computer and UNIX operating systems, while providing an integrating structure over all information stored in the iSeries.

The key features of the integrated file system are:

- Support for storing information in stream files that can contain long continuous strings of data. These strings of data might be, for example, the text of a document or the picture elements in a picture. The stream file support is designed for efficient use in client/server applications.
- A hierarchical directory structure that allows objects to be organized by specifying the path through the directories to an object for access to an object.
- A common view of stream files stored locally on iSeries, Integrated Netfinity Server for iSeries, or a remote Windows NT server. Stream files can also be stored remotely on a local Area Network (LAN) server.

Directories and Current Directory

A *directory* is a special object that is used to locate objects by names specified by users. Each directory contains a list of objects that are attached to it, and that list may include other directories.

The *current directory* is the first directory in which the operating system locates files, and where it also stores temporary files and output files. When you request an operation for an object, such as a file, the system searches for the object in the current directory, unless a different directory path is specified. The current directory is similar in nature to the current library. If the file selection does not start with '/' (Root Directory), the files should be in the path of the current directory.

Path and Path Names

A *path name* (also called a *pathname* on some systems) informs the system how to locate an object. The path name is expressed as a sequence of directory names followed by the name of the object. Individual directories and the object name are separated by a slash (/) character. An example might be: directory1/directory2/file.

For convenience, the back slash (\) can be used instead of the slash in integrated file system commands.

There are two ways of indicating a path name:

An *absolute path name* begins at the highest level, or *root directory* (which is identified by the / character). For example, consider the following path from the / directory to the file named *testit*: */mydept/myfiles/testit*.

If the path name does not begin with the / character, the system assumes that the path begins at your current directory. This type of path name is called a *relative path name*. For example, if your current directory is *mydept* and it has a sub-directory named *myfiles* containing the file *testit*, the relative path name to the file is: *myfiles/testit.* Notice that the path name does not include the name of the current directory. The first item in the name is the directory or object at the next level below the current directory.

Stream Files

A *stream file* is a randomly accessible sequence of bytes with no further structure imposed by the system. The integrated file system provides support for storing and operating on information in the form of stream files. Documents that are stored in iSeries folders are stream files. Other examples of stream files are PC files and the files in UNIX systems. An integrated file system stream file is a system object that has an object type of *STMF.

Other IFS Objects

There are other object types (such as link objects, etc.) in the IFS which at this time are not supported by ***PKZIP^{i}.***

File Systems in the IFS

There are currently ten (10) file systems that are part of the integrated file system. Each file system is a major sub-tree in the IFS directory structure. A file system provides the support to access specific segments of storage that are organized as logical units. These logical units on the iSeries are files, directories, libraries, and objects.

Each of these file systems has a set of logical structures and rules for interacting with information in storage. These structures and rules may be (and often are) different from one file system to another. The IFS treats the library support and folders support as separate file systems.

The ten file systems are:

- **"root"** - / **file system.** This file system takes full advantage of stream file support and hierarchical directory structure of the integrated file system. The root file system has the characteristics of the Disk Operating System (DOS) and OS/2 file systems. Most of references throughout this guide refer to the "root" system.
- **QDLS** - **Document Library Services file system.** This file system provides access to documents and folders. See IBM's *Office Services Concepts and Programmer's Guide (SH21-0703)* for additional information.
- **QOPT** - **Optical file system.** This file system provides access to stream data that is stored on optical media (such as CDs). See IBM's *Optical Support (SC41-5310)* for additional information.
- **QSYS.LIB** - **Library file system.** This file system supports the iSeries library structure and provides access to database files and all of the other iSeries object types that the library support manages.

- **NFS** - **Network File System.** This file system provides the user with access to data and objects that are stored on a remote NFS server. An NFS server can export a network file system that NFS clients will then mount dynamically. See IBM's *OS/400 Network File System Support (SC41-5714)* for additional information.
- **QFileSvr.400.** This file system provides access to other file systems that reside on remote iSeries systems. See IBM's *Integrated File System Introduction (SC41-5711)* for additional information.
- **QNetWare** - **QNetWare file system.** This file system provides access to local or remote data and objects that are stored on a server that runs Novell NetWare 4.10 or 4.11 or to standalone PC servers running Novell Netware 3.12, 4.10, 4.11, or 5.0. A user can mount NetWare file systems over existing local file systems dynamically. See *File Management (SC41-5710)* for additional information.
- **QNTC Windows NT Server file system.** This file system provides access to data and objects that are stored on a server running Windows NT 4.0 or higher. It allows iSeries applications to use the same data as Windows NT clients. This includes access to the data on a Windows NT Server that is running on an integrated PC Server. See IBM's *OS/400-iSeries Integration with Windows NT Server (SC41-5439)* for details.
- **QOpenSys** - **Open Systems file system.** This file system is compatible with UNIX-based open system standards, such as POSIX and XPG. Like the root file system, this file system takes advantage of the stream file and directory support that is provided by the integrated file system. In addition, it supports case-sensitive object names. See IBM's *Integrated File System Introduction (SC41-5711)* for additional information.
- **UDFS** - **User-Defined File System.** This file system resides on the Auxiliary storage pool (ASP) of the user's choice. The user creates and manages this file system. See IBM's *Integrated File System Introduction (SC41-5711)* for additional information.

***PKZIP**^{i}* works with all file systems, but the rules of each file system must be adhered to or a file I/O error will most likely occur. In most cases, the files can be compressed and extracted in one run when all the file names and paths meet the file system's rules. When creating an archive file in one file system, one restriction is that when using the TMPPATH option, the temp path must also be in the same file system as the archive files.

On the following pages are rules for some of the most used file systems.

Document Library Services File System (QDLS)

The QDLS file system supports the folders structure. It provides access to documents and folders. Additionally, it supports iSeries folders and document library objects (DLOs) and supports data stored in stream files.

Considerations and Limitations:

- You must be enrolled in the system distribution directory when working with objects in QDLS.

- QDLS converts the lowercase English alphabetic characters *a* through *z* to uppercase when used in object names. Therefore, a search for object names using only those characters is not case sensitive. All other characters are case sensitive in QDLS.
- Each component of the path name can consist of just a name, such as: /QDLS/MYFLR1/MYDOC1 - or - a name plus an extension (similar to a DOS file extension), such as: /QDLS/MYFLR1/MYDOC1.TXT.
- The name in each component can be up to 8 characters long, and the extension (if any) can be up to 3 characters long. The maximum length of the path name is 82 characters, assuming an absolute path name that begins with /QDLS.
- The directory hierarchy within QDLS can be 32 levels deep.
- Must have proper authority within the path.
- The folders in the path must already exist.
- PKZIP will not create folders at this time.
- For more details, see the “Rules for Specifying Folder and Document Names” discussion in the publication *CL Reference*.

Optical File System (QOPT)

The QOPT file system provides access to stream data that is stored on optical media (such as CDs). Additionally, it provides a hierarchical directory structure (similar to PC operating systems such as DOS and OS/2), is optimized for stream file input/output, and supports data stored in stream files (known as DSTMF or Distributed Stream Files).

Considerations and Limitations:

- QOPT converts the lowercase English alphabetic characters a to z to uppercase when used in object names. Therefore, a search for object names using only those characters is not case-sensitive. For more details, see the publication *Optical Support*.
- The path name must begin with a slash (/) and contain no more than 294 characters. The path is made up of the file system name, the volume name, the directory and sub-directory names, and the file name. For example: /QOPT/VOLUMENAME/DIRECTORYNAME/SUBDIRECTORYNAME/FILENAME
- The file system name (/QOPT) is required.
- The volume name is required and can be up to 32 characters long.
- You can include one or more directories or sub-directories in the path name, but QOPT requires none. The total number of characters in all directory names and sub-directory names (including the leading slash) cannot exceed 256 characters. Directory and file names allow any character except X'00' through X'3F', X'FF', lowercase alphabetic characters, and the following characters:
    - Asterisk (*)
    - Hyphen (-)
    - Question mark (?)

    - Quotation mark (")
    - Greater than (>)
    - Less than (<)
- The file name is the last element in the path name. The file name length is limited by the directory name length in the path. The directory names and file name combined cannot exceed 256 characters, including the leading slash.

For more details on path name rules in the QOPT file system, see the “Path Name Rules” discussion in the publication *Optical Support*.

Using QSYS.LIB via the Integrated File System Interface

Even though ***PKZIP**i* accesses the QSYS library file system directly, there is an ability to access the QSYS.LIB file system through the integrated file system interface. In using the integrated file system interface, you should be aware of the following considerations and limitations:

- File handling restrictions in the QSYS.LIB file system are:
- Logical files are not supported.
- Physical files supported for text mode access are program-described physical files containing a single field and source physical files containing a single text field. Physical files supported for binary mode access include externally-described physical files in addition to files supported for text mode access.
- If any job has a database file member open, only one job is given write access to that file member at any given time. Other requests are allowed read-only access.
- In general, the QSYS.LIB file system does not distinguish between uppercase and lowercase in the names of objects. A search for object names achieves the same result, regardless of whether characters in the names are uppercase or lowercase. If a name is enclosed in quotation marks, the case of each character in the name is preserved. A search involving quoted names, therefore, is sensitive to the case of the characters in the quoted name.
- Each component of the path name must contain the object name followed by the object type of the object. For example: /QSYS.LIB/TESTLIB.LIB/MYFILE.FILE/MYFILE.MBR. The object name and object type are separated by a period (.). Objects in a library can have the same name if they are different object types, so the object type must be specified uniquely to identify the object.
- The object name in each component can be up to 10 characters long, and the object type can be up to 6 characters long.
- The directory hierarchy within QSYS.LIB can either be two or three levels deep (two or three components in the path name), depending on the type of object being accessed. If the object is a database file, the hierarchy can contain three levels (library, file, or member), otherwise, there can be only two levels (library or object). The combined length of each component name plus the number of directory levels determine the maximum length of the path name. If / and QSYS.LIB are included as the first two levels, the directory hierarchy for QSYS.LIB can be up to five levels deep.

- The characters in names are converted to code when the names are stored. Quoted names, however, are stored using the code page of the job.

For information about code pages, see the publication *National Language Support*.

IFS Summary

Only directories and stream files are supported by ***PKZIP^{i}.***

If the archive file is to be in IFS, set parameter TYPARCHFL(*IFS).

If the file being compressed or extracted is in IFS, set parameter TYPFL2ZP(*IFS)

If the files to be selected for compression are to be non-case sensitive set parameter TYPARCHFL(*IFS2).

If the list files are to be in IFS (see Appendix C), set parameter TYPLISTFL(*IFS).

Format Summary:

Directory	Directory1/directory2	will be current directory
Stream File	filename or directory/filename	will be current directory
Full Path	/Directory1/Directory2/filename	

For more information, see the IBM publication *Integrated File System Introduction* (SC41-5711) or visit the IBM web site.

SAVF

SAVF, denoted by the OS/400 system TYPE(*FILE) and ATTR(SAVF), is a special form of file designed specifically to handle save/restore data in the iSeries system.

Some SAVF special characteristics are:

- The SAVF is always processed as binary with all records being 528 characters in length.
- Only a save and restore iSeries function can update or change data.
- A SAVF will not be selected if a member name is included in the file specification.
- A SAVF is a means to compress other iSeries object types (programs, modules, commands, logical files, triggers, etc.) that are in the iSeries system by first doing a SAVLIB or SAVOBJ for those objects to a SAVF. Then you can compress and extract the SAVF.

Compressing a SAVF file

The only difference when compressing a SAVF is not to specify a member (only library/file). If a member is specified, then no SAVF types will be compressed.

Extracting Records into a SAVF file

It is helpful before extracting records from a ZIP archive to be aware of what file names and file attributes are being stored for the compressed file. VIEWOPT(*DETAIL) may be used on the archive to verify the information. An attribute is stored in the archive header that identifies if the file is a SAVF. The PKUNZIP program will also retain the original attribute from the extended attributes, such as SAVF description and library description.

A common problem in some iSeries environments is that some users may not have the authority to the SAVF commands which can result in failures.

Overwriting Current SAVF File

When extracting a compressed file, it may be desirable to overwrite the existing file. By using the OVERWRITE(*YES) parameter, PKUNZIP will first issue a CLRSAVF command to clear the save file. This demonstrates why care should be taken when extracting a SAVF.

Compressing Spool Files

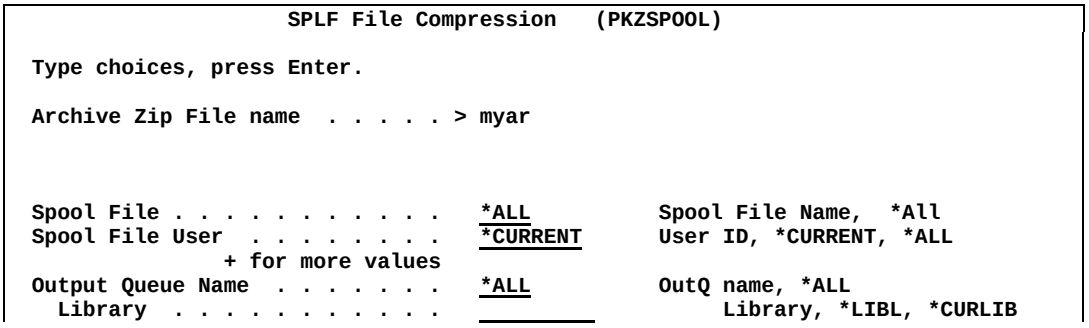
PKZIP^{i} has the ability to select, compress and extract spool files. Not only can a spool file be compressed, they can be converted to other document formats that will allow the document file to be distributed and read by other media and software.

All spool files are eligible for compression but only spool file types *SCS, *IPDS are supported for text document conversion.

By using the PKZIP command and setting parameter TYPFL2ZP(*SPL), other parameters will be shown to help select the spool files. To assist, a new command PKZSPOOL is provided to sequence the selections and to eliminate parameters that are not valid for the selection of spool files.

Spool file parameters specifies the group of spool files that are to be selected. Eight positional values can be specified to select the spool files: the spool file name (SPLFILE), the spool file number (SPLNBR), the user that created the files (SFUSER), the OUTQ that the file is residing (SFQUEUE), the form type specified (SFFORM), the user data tag associated with the spool file (SFUSRDTA), the status of the spool file (SFSTATUS), or the specific job name/user name/job number (SFJOBNAM). Only files that meet all of the selection values will be selected. A sample of the default selection parameters is shown in the window below:

Selection sample using the PKZSPOOL command.



```
Print Form Type  . . . . . .   *ALL          Form Type, *STD, *ALL
Print File User Specified Data    *ALL       User Data,  *all
Spool Files Status . . . . . .   *ALL         *ALL, *READY, *HELD...
             + for more values
Spool File Job Name  . . . . .   _________    Job name, blank for all
  User . . . . . .           .   _________    User Id
  Job Number . . . . . .         ______       Job Number
Spooled file number  . . . . .   *ALL         1-9999, *ALL, *LAST
Target File Format . . . . . .   *SPLF        *SPLF, *TEXT, *PDF, *TEXT1...
Target File Name . . . . . .     *GEN1
Type of processing . . . . . .   *ADD         *ADD, *UPDATE, *FRESHEN ..
Compression Level  . . . . . .   *SUPERFAST   *NO, *FAST, *NORMAL, *MAX...
File Types . . . . . .           *DETECT      *DETECT *TEXT *BINARY ........
Zip Spool Files  . . . . . .     *SPL         *SPL
Archive Passphrase . . . . . .
```

After defining what spool files are to be selected for compression, you will need to define the file format the spool file should be stored in the archive. At this time, there three formats: *SPLF (spool file native mode), *TEXT (ASCII text document with three variations of how a new page is handled) and *PDF (Adobe portable document format).

For use with *TEXT and *PDF there are three variations of storing the file name in the archive with the parameter SFTGFILE. SFTGFILE (*GEN1) will generate a very specific name using most of the spool file name attributes to form the file name so that it will not be a duplicated. The name will be built as follows:

> "Job-Name**/**User-Name**/#**Job-Number**/**Spool-File-Name**/F**spool- File-Number.Suffix"
>
> For example: "MYJOB/BILLS/#152681/INVOICE/F0021.SPLF"
>
> The suffix is dependent on the SFTARGET setting. *SPLF can only be stored as SFTGFILE (*GEN1).

SFTGFILE (*GEN1P) will generate the same specific name generated by *GEN1 except the '/' for folders will all be replaced by '.' to make the file name one lone name. For example:

> MYJOB.BILLS.#152681.INVOICE.F0021.SPLF

SFTGFILE (*GEN2) uses the spool file name and appends the spool file number followed by the suffix that is depended on the SFTARGET setting. Caution should be taken in that a duplicate file name in the archive could be created. An example of GEN2 is a spool file INVOICE with spool file number of 21 that will be converted to a text file will generate a file name of INVOICE21.TXT.

In cases where a very specific name is desired for the file in archive name, SFTGFILE() can be coded with the name. This is designed for selecting only one file at a time otherwise file names will be duplicated. Alternatively, you could add coding to the CVTNAME routine and use the CVTFLAG to generate the desired file name.

6 i5/OS PKWARE Save/Restore Application Feature (iPSRA)

The i5/OS PKWARE Save/Restore Application (iPSRA) feature enables ***PKZIP***^{i} to compress/encrypt iSeries save files directly to a file in an archive. The process produces a result similar to creating a save file first and then compressing and/or encrypting it into an archive, but the iPSRA feature economizes on time and disk space by skipping the intermediate step. The iPSRA process can be integrated with your existing backup/recovery procedures and systems on the iSeries.

iPSRA assists not only with compressing your save data but with encrypting the data for offsite storage. The iPSRA process can execute multiple save operations with one compression run, making it unnecessary to run repeated individual save commands.

The iPSRA feature integrates ***PKZIP***^{i} compression and encryption technology with IBM iSeries Save and Restore APIs. The iPSRA feature works with the PKZIP command to save objects and with the PKUNZIP command to restore them.

To use the iPSRA feature, you must have a working knowledge of the save/restore commands in their native mode. The same uses and restrictions apply to the save/restore commands in ***PKZIP***^{i} as to the native commands. The use and format of the outfiles with any of the save or restore commands are the same as with the native commands. For information about the native save and restore commands, see the IBM manuals that describe these commands, or the IBM Web site:

http://publib.boulder.ibm.com/html/as400/infocenter.html

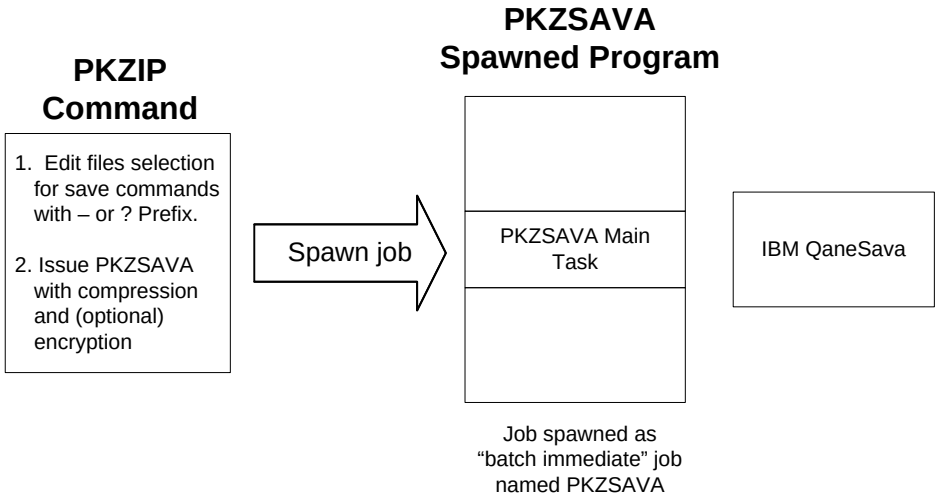
Save/Restore Command Overview

The iPSRA feature supports the following Save command types:

- Save (SAV) command
- Save Object (SAVOBJ) command
- Save Document Library Object (SAVDLO) command
- Save Library (SAVLIB) command
- Save Changed Object (SAVCHGOBJ) command

With the iPSRA feature, PKZIP spawns a batch immediate program named PKZSAVA that processes save command data and causes it to be compressed—and optionally encrypted—into an archive instead of being saved directly to disk. PKZSAVA uses the

IBM API, which is a pre-started job in the QSYSWRK subsystem. The figure below gives an overview of the process.



Saving Data

To use the iPSRA feature to save data, you enter command strings for SAV, SAVOBJ, and/or SAVLIB in the FILES parameter of PKZIP. Prefix each command string with a hyphen (-) or question mark (?).

- A '-' tells PKZIP that the entire command is entered and is ready to be validated
- A '?' causes PKZIP to prompt for command parameters to enter before validating command syntax

The following example shows a PKZIP command to save the library MYLIB and some other objects:

➔ **PKZIP ARCHIVE('/MYpath/myarchive') FILES('-SAVLIB LIB(MYLIB) DEV(MYNAME1)' '?SAVOBJ DEV(MYNAME2)')**

The command line above creates a saved library file or iPSRA file of the library MYLIB and compresses it into the archive /MYpath/myarchive as file SAVLIB01_MYNAME1. The command line uses the '?' prefix with the command SAVOBJ to prompt for the objects to be saved. These must be specified at the prompt. The SAVOBJ command creates an iPSRA file named SAVOBJ02_MYNAME2 in the archive.

Restoring Data

The restore of a save object with a RSTLIB command works similarly except that RSTLIB is used with PKUNZIP and only one file can be restored at a time, to assure proper building of the saved objects.

Examples are given at the end of this chapter.

Syntax

Save command parameters must be consistent with the save command type entered and must be separated by at least one blank character. Refer to the Control Language (CL) documentation for detailed information about valid parameters when you save objects to save files.

File Names Used for Saved Data

The file name PKZIP uses for saved data in an archive is based on the type of save command and the name used in the save command DEV parameter. If the DEV parameter is *SAVF, then the name comes from the SAVF parameter. See the table below for examples.

Command	***File Name***
SAVOBJ DEV('MYPAYROLL')	SAVOBJnn_ MYPAYROLL
SAVLIB DEV(*SAVF)	SAVF(MYLIB/MYSAVF)
SAV DEV('MYSAVFDEVPATH')	SAVnn_ MYSAVFDEVPATH

The *nn* in the example file names is the sequence number of the command for one PKZIP run.

PKZIP removes from an archive existing files that have the same name as a new save file. To help you avoid duplicating file names when updating an existing archive, PKZIP checks to see if there are any iPSRA files in the archive. If there are, PKZIP uses the largest *nn* number plus 1 as the starting *nn* for iPSRA files to be added. This helps avoid accidentally overwriting iPSRA files when updating an archive.

Extended Data in Archive

Each save operation creates specific data in the extended data area to tell PKUNZIP that a file to be restored is a file of the special SAVE type. The extended data also provides history information that can be displayed with VIEWOPT(*ALL) to show the original command, the job when the save data was created, the target release used for the save, and the spawned job PKZSAVA.

```
Filename: SAVLIB01_DEDSAV01
Detected File type:     SAVE Apps. Data
Created by:             PKZIP(R) for i5/OS  9.0
Zip Spec to Extract:    2.0  Or Greater
Compression method:     Deflated   [Superfast]
Date and Time           2005 Aug 8 14:08:58
Compressed size:        42876 bytes
Uncompressed size:      413808 bytes
32-bit CRC value (hex): 334d1674
Extended attributes:    yes, [Length = 134]
File Save Apps Data:<savlib lib(deD) dev(dedsav01) OUTPUT(*OUTFILE)
  OUTFILE(WSS/TEM01)>
                    :TGTRLS(*Current) Save Job <019627/EVWSS/PKZSAVA>
File Comment:"none"
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Found 1 file, 413808 bytes uncompressed, 42876 bytes compressed:   90%
SecureUNZIP  extracted     0 files
SecureUNZIP   Completed Successfully
```

```
                        Additional Message Information

Message ID . . . . . :   AQZ0895
Date sent  . . . . . :   08/08/05      Time sent  . . . . . :   14:09:20

Message . . . . :   File Save Apps Data:<savlib lib(deD) dev(dedsav01)
  OUTPUT(*OUTFILE) OUTFILE(WSS/TEM01)>

This shows that the file is a Save Application data file type 4 and the
  command that was used to store the save file in the archive.  Save Data
  created with PKZIP job 019586/EVWSS/EVWSSL01. Target System was *Current.
```

Notice in the output above that the spawned job (019627/EVWSS/PKZSAVA) was captured as well as the PKZIP job (019586/EVWSS/EVWSSL01). Information on the spawned job may be needed to do a DSPLOG command.

The target release (TGTRLS) is also shown to note the target setting for the run.

Notes and Restrictions

All IBM restrictions and security requirements apply to the use of the Save and Restore commands in iPSRA. Some additional restrictions are noted below.

PKZIP

- QTEMP cannot be specified for the library name on the OUTFILE or SAVACTMSGQ parameters.
- Some parameters of the save commands that are not used by iPSRA are ignored. For example, CLEAR, DTACPK, and so on.
- Objects saved by PKZIP can only be restored using the restore from application with PKUNZIP, and they can only be restored to a release of the operating system that is the same or later than the version from which the objects were saved.
- The save parameters are only completely validated when PKZIP submits the save command for processing.
- A target release VxRxMx value prior to V5R1M0 is not valid: The iPSRA feature is not supported prior to version 5, release 1, modification level 0 of PKZIP. The version, release, and modification level depend on the save operation being performed. See the valid values for the TGTRLS parameter table in the iSeries Backup and Recovery Manual for a complete list of valid values.
- All compression methods except the Terse compression method are supported with iPSRA.
- Positional options on the save commands must contain the parameter. For example, the save library command

  SAVLIB WSS DEV(*SAVF) SAVF(TESTWSS)

  would not work to save the library WSS. The WSS is a positional parameter, where it is assumed WSS was the LIB option. The correct approach with iPSRA is to use the command:

  -SAVLIB LIB(WSS) DEV(*SAVF) SAVF(TESTWSS)

- The save operation must be completely successful or it aborts. If any object is not saved for any reason, the PKZIP job assumes a failure. Do not include an object that will not save, as this object will cause a major failure.

PKUNZIP

- QTEMP cannot be specified for the library name on the OUTFILE parameter.
- To ensure that all objects are properly restored, only one restore command can be processed per run.
- Some parameters of the restore commands are not used by iPSRA and are ignored. For example, VOL, SEQNBR, and so on.
- The user must have the required security for the restore command.

Using OUTPUT and OUTFILE with the Save Commands

A save command can have an OUTFILE parameter that is used to build a file listing the objects saved with that command. When an OUTFILE is specified for a save command, PKZIP automatically archives the outfile in the same archive as the iPSRA file with the name specified. The outfile provides a record of the contents of an iPSRA file. An outfile has the format and restrictions defined by IBM for the save commands. Use of outfiles is optional.

If OUTPUT(*PRINT) or OUTFILE(*PRINT) is used with a save or restore command, the printout is produced by the IBM API job and not with the PKZIP job nor the spawned PKZSAVA job. Therefore, it appears in the special OUTQ job named QPRTJOB for each user.

How to Use the Save Application Feature

The save option is activated by entering a save command in the FILES parameter of PKZIP, prefixed with a hyphen (-) or question mark (?). Multiple save commands can be entered to select multiple sets of files in the same pass. The save commands do not need to be the same, nor do they need to use the same prefix.

If a command fails the pre-command processor, PKZIP issues the message AQZ0332, which shows the failed command. The reason for the failure appears in the job log prior to this message.

If a failure occurs during the processing of the save commands, the reason for the failure appears in the job log of the spawned job. If any errors occur in the spawned job, a job log will be forced. There is no pre-check processing on security or on the objects themselves. All data verification is handled by the save API.

For example, the following command tries to save a library (NOLIB) that does not exist:

➔ **PKZIP ARCHIVE('/yourpath/BILLS/X5TESTL.ZIP') TYPARCHFL(*IFS) FILES('-SAVLIB LIB(NOLIB) DEV(TESTSAV01) OUTPUT(*PRINT) ')**

The log of the PKZIP output might look like this:

```
Scanning files in *DB for match  ...
Found  0 matching files
1 Save Command(s) selected
Command:<SAVLIB LIB(NOLIB) OUTPUT(*PRINT)>
Compressing SAVLIB01_TESTSAV01 in SAVE Apps. Data mode
Save Operation encountered an error.  See Job Log of PKZSAVA save job for fur
ther details.
iPSRA Initialization Failure has occurred
iPSRA Failed.  Save command not successful.
SecureZIP Copied 1 files from input archive
SecureZIP Compressed 0 files in Archive /yourpath/BILLS/X5TESTL.ZIP
SecureZIP Completed with Errors
Press ENTER to end terminal session.
```

The job log of the PKSAVF output might look like this:

```
CPF3781    Diagnostic 30   08/08/05  14:49:10.428528  QANESERV     QSYS
From module . . . . . :   QANESERV
From procedure  . . . . . :   QaneSendPgmMsg__FP14qanec_CTLBLK_tPcT2iN24
Statement . . . . . :   19
To module . . . . . :   QP0ZPCPN
To procedure  . . . . . :   InvokeTargetPgm__FP11qp0z_pcp_cb
Statement . . . . . :   187
Message . . . . :   Library NOLIB not found.
   Cause . . . . . :   The library specified for the save or restore
   operation does not exist on the system. Recovery  . . . :   Do one of the
   following and try the request again: If this is a save operation, correct
   the library name on the LIB parameter. If this is a restore operation,
   correct the library name specified on the SAVLIB or RSTLIB parameter, or
   use the Create Library (CRTLIB) command to create the library by
   specifying LIB(NOLIB). If this is a restore operation and VOL(*SAVVOL)
   was specified, the save library must exist in the auxiliary storage pool
   specified on the RSTASPDEV parameter. If RSTASPDEV(*SAVASPDEV) and
   RSTASP(*SAVASP) are specified along with VOL(*SAVVOL), then the save
   library must exist in the system ASP. To restore a library that is new to
   the system, specify VOL(*MOUNTED) instead of VOL(*SAVVOL).
```

How to Use the Restore Application Feature

To restore an iPSRA file from archive, code the restore command in the RSTIPSRA parameter of PKUNZIP. The RSTIPSRA parameter is defined as a command entry, so do not use quotes around the command. Enclose the entire restore command in parentheses: RSTIPSRA(command). To be prompted at the command, place the cursor on the RSTIPSRA entry and press the F4 key.

If an archive contains more than one file, you must use the FILES parameter to select the file you want to match up with the RSTIPSRA parameter. PKUNZIP restores only one iPSRA file per run.

If any object is not restored, PKUNZIP issues the message AQZ1007 and creates a job log for the PKZRSTA job. You should review the log to find any object that was not restored and the reason.

If a partial restore is performed, then the CRC and/or hash calculation for authentication does not take place, and the warning message AQZ1000 is displayed. This situation can arise if the save operation was a SAVLIB, but the restore operation restores only a few objects with the RSTOBJ.

Database Considerations for Save and Restore

Below are some tips for working with the save and restore functions.

- When you save an object to a save file or using iPSRA, you can prevent the system from updating the date and time of the save operation by specifying UPDHST(*NO) on the save command.
- When you restore an object, the system always updates the object description with the date and time of the restore operation. Display the object description and other save/restore related information by using the Display Object Description (DSPOBJD) command with DETAIL(*FULL).
- To display the last save/restore date for a database file, type: DSPFD FILE(filename) TYPE(*MBR).

Sample Jobs

iPSRA Example 1

The following example saves the library DED and prints the output of the save. It also saves the file object TESTFILE from the library TESTLIB with several options of the SAVOBJ. These save application files are compressed with a default setting and will be encrypted using a passphrase.

➔ **PKZIP ARCHIVE('/yourpath/bills/testsavx1.zip') TYPARCHFL(*IFS) FILES(**
'-SAVLIB LIB(DED) DEV(DEDSAV01) OUTPUT(*PRINT) '
'-SAVOBJ OBJ(TESTFILE) LIB(TESTLIB) DEV(TESTOBJ11) OBJTYPE(*FILE) TGTRLS(V5R1M0) UPDHST(*NO) PRECHK(*YES) OUTPUT(*PRINT) ')
PASSWORD('bills00000') VPASSWORD('bills00000')

```
Scanning files in *DB for match  ...
Found  0 matching files
2 Save Command(s) selected
Command:<SAVLIB LIB(DED)  OUTPUT(*PRINT)>
Compressing SAVLIB01_DEDSAV01 in SAVE Apps. Data mode
Add  SAVLIB01_DEDSAV01  --  Deflating (90%)  encrypt(BSAFE AES 256Key)
Command:<SAVOBJ OBJ(TESTFILE) LIB(TESTLIB) OBJTYPE(*FILE)
UPDHST(*NO) PRECHK(*YES) OUTPUT(*PRINT)>
Compressing SAVOBJ02_TESTOBJ11 in SAVE Apps. Data mode
Add  SAVOBJ02_TESTOBJ11  --  Deflating (79%)  encrypt(BSAFE AES 256Key)
SecureZIP Compressed 2 files in Archive /yourpath/bills/testsavx1.zip
SecureZIP Completed Successfully
```

iPSRA Example 2

The following commands display the contents of the archive:

➔ **PKUNZIP ARCHIVE('/yourpath/bills/testsavx1.zip') TYPARCHFL(*IFS) TYPE(*VIEW)**

```
Archive:  /yourpath/bills/testsavx1.zip, 1358415 bytes, 2 files, 1 Segment
  Length  Method    Size  Ratio  Date   Time  CRC-32    Name
 -------- ------  ------- -----  ----   ----  ------    ----
   430192 Defl:S    43718  90% 08-09-05 08:16 ac1f8407 !SAVLIB01_DEDSAV01
  6325776 Defl:S  1313814  79% 08-09-05 08:16 101311d4 !SAVOBJ02_TESTOBJ11
 --------         -------  ----                          -------
 6755968          1357532  80%                            2 files
```

➔ **PKUNZIP ARCHIVE('/yourpath/bills/testsavx1.zip') TYPARCHFL(*IFS) TYPE(*VIEW) VIEWOPT(*ALL)**

```
Archive Comment:"SecureZIP for i5/OS"
Filename: SAVLIB01_DEDSAV01
Detected File type:     SAVE Apps. Data   Encrypt=Strong Encrypted
Created by:             PKZIP(R) for i5/OS  9.0
Zip Spec to Extract:    5.1  Or Greater
Compression method:     Deflated   [Superfast]
Date and Time           2005 Aug 9 08:16:16
Compressed size:        43718 bytes
Uncompressed size:      430192 bytes
32-bit CRC value (hex): ac1f8407
Extended attributes:    yes, [Length = 130]
Strong Encryption AES 256 Key  (BSAFE).
Algorithm Key 256,  Security type Passphrase
Number Certifcate Recipients 0
Recipient List:
File Save Apps Data:<SAVLIB LIB(DED) DEV(DEDSAV01)  OUTPUT(*PRINT)>
                    :TGTRLS(*Current) Save Job <019700/EVWSS/PKZSAVA>
File Comment:"none"
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Filename: SAVOBJ02_TESTOBJ11
Detected File type:     SAVE Apps. Data   Encrypt=Strong Encrypted
Created by:             PKZIP(R) for i5/OS  9.2
Zip Spec to Extract:    5.1  Or Greater
Compression method:     Deflated   [Superfast]
Date and Time           2005 Aug 9 08:16:16
Compressed size:        1313814 bytes
Uncompressed size:      6325776 bytes
32-bit CRC value (hex): 101311d4
Extended attributes:    yes, [Length = 217]
Strong Encryption AES 256 Key  (BSAFE).
Algorithm Key 256,  Security type Passphrase
Number Certifcate Recipients 0
Recipient List:
File Save Apps Data:<SAVOBJ OBJ(TESTFILE) LIB(TESTLIB) DEV(TESTOBJ11) OBJTYPE
(*FILE) TGTRLS(V5R1M0) UPDHST(*NO) PRECHK(*YES) OUTPUT(*PRINT)>

                    :TGTRLS(*Current) Save Job <019701/EVWSS/PKZSAVA>
File Comment:"none"
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Found 2 files, 6755968 bytes uncompressed, 1357532 bytes compressed:   80%
```

iPSRA Example 3

Now we want to restore the saved library DED to a new library called DEDNEW and then restore the object TESTFILE to the new DEDNEW library. PKUNZIP can perform only one restore at a time, so the operation requires two steps.

Step1.

➔ **PKUNZIP ARCHIVE('/yourpath/bills/testsavx1.zip') TYPARCHFL(*IFS)
FILES('SAVLIB01_DEDSAV01') TYPE(*EXTRACT)
RSTIPSRA(RSTLIB SAVLIB(DED) DEV(RSTDED) output(*print)
RSTLIB(DEDNEW))
PASSWORD('bills00000')**

```
UNZIP Archive: /yourpath/bills/testsavx1.zip
Archive Comment:"SecureZIP for i5/OS"
Searching Archive /yourpath/bills/testsavx1.zip  for files to extract
Command:<RSTLIB SAVLIB(DED) RSTLIB(DEDNEW) OUTPUT(*PRINT)>
Extracting file SAVLIB01_DEDSAV01
Inflating *iPSRA:SAVLIB01_DEDSAV01 iPSRA File
SecureUNZIP  extracted     1 files
SecureUNZIP   Completed Successfully
```

Step 2.

➔ **PKUNZIP ARCHIVE('/yourpath/bills/testsavx1.zip') TYPARCHFL(*IFS)
FILES('SAVOBJ02_TESTOBJ11') TYPE(*EXTRACT)
RSTIPSRA(RSTOBJ OBJ(TESTFILE) SAVLIB(TESTLIB)
DEV(RSTTEST) OBJTYPE(*FILE) RSTLIB(DEDNEW)
OUTPUT(*PRINT)) PASSWORD('bills00000')**

```
UNZIP Archive: /yourpath/bills/testsavx1.zip
Searching Archive /yourpath/bills/testsavx1.zip  for files to extract
Command:<RSTOBJ OBJ(TESTFILE) SAVLIB(TESTLIB) OBJTYPE(*FILE)
  RSTLIB(DEDNEW) OUTPUT(*PRINT)>
Extracting file SAVOBJ02_TESTOBJ11
Inflating *iPSRA:SAVOBJ02_TESTOBJ11 iPSRA File
SecureUNZIP  extracted     1 files
SecureUNZIP   Completed Successfully
```

iPSRA Example 4

The following example shows that we can restore one or more objects from an iPSRA file that was created with SAVLIB.

➔ **PKUNZIP ARCHIVE('/yourpath/bills/testsavx1.zip') TYPARCHFL(*IFS)
FILES('SAVLIB01_DEDSAV01') TYPE(*EXTRACT)
RSTIPSRA(RSTOBJ OBJ(MYFILE2) SAVLIB(DED) DEV(RST1FILE)
OBJTYPE(*FILE) RSTLIB(DEDNEW) OUTPUT(*PRINT))
PASSWORD('bills00000')**

```
UNZIP Archive: /yourpath/bills/testsavx1.zip
Searching Archive /yourpath/bills/testsavx1.zip  for files to extract
Command:<RSTOBJ OBJ(MYFILE2) SAVLIB(DED) OBJTYPE(*FILE)
RSTLIB(DEDNEW) OUTPUT(*PRINT)>
Extracting file SAVLIB01_DEDSAV01
Inflating *iPSRA:SAVLIB01_DEDSAV01 iPSRA File
SecureUNZIP  extracted     1 files
SecureUNZIP   Completed Successfully
```

iPSRA Example 5

The following example demonstrates the use of OUTFILE in a save command and shows how PKZIP automatically adds the outfile to the archive.

```
➔ PKZIP ARCHIVE('/yourpath/bills/iPSRA_test/x3.zip')
  TYPARCHFL(*IFS) FILES(
      '-SAVLIB LIB(DED) DEV(DEDSAV01) OUTPUT(*OUTFILE)
      OUTFILE(ATEST/DEDSAV01)'
      '-SAV DEV('IFS_testpkcs7*') OBJ(('/ajunk/testpkcs7/*'))
  OUTPUT('/yourpath/bills/iPSRA_test/File01_SAV')  ')
```

```
Scanning files in *DB for match  ...
Found  2 matching files
2 Save Command(s) selected
Command:<SAVLIB LIB(DED) OUTPUT(*OUTFILE) OUTFILE(ATEST/DEDSAV01)>
Compressing SAVLIB01_DEDSAV01 in SAVE Apps. Data mode
Add  SAVLIB01_DEDSAV01  --  Deflating (90%)
Compressing ATEST/DEDSAV01(DEDSAV01) in TEXT mode
Add  ATEST/DEDSAV01/DEDSAV01  --  Deflating (98%)
Command:<SAV OBJ(('/AJUNK/TESTPKCS7/*'))
   OUTPUT('/yourpath/BILLS/IPSRA_TEST/FILE01_SAV')>
Compressing SAV02_IFS_TESTPKCS7* in SAVE Apps. Data mode
Add  SAV02_IFS_TESTPKCS7*  --  Deflating (63%)
Compressing /yourpath/BILLS/IPSRA_TEST/FILE01_SAV in BINARY mode
Add  /yourpath/BILLS/IPSRA_TEST/FILE01_SAV  --  Deflating (61%)
SecureZIP Compressed 4 files in Archive /yourpath/bills/iPSRA_test/x3.zip
SecureZIP Completed Successfully
```

Four files are stored in the archive. Two are the iPSRA files, and the other two are the outfiles in the commands.

iPSRA Example 6

The example below shows a restore error.

```
➔ PKUNZIP ARCHIVE('/yourpath/bills/testsavx1.zip') TYPARCHFL(*IFS)
  FILES('SAVLIB01_DEDSAV01') TYPE(*EXTRACT)
  RSTIPSRA(RSTOBJ OBJ(TESTFILE) SAVLIB(TESTLIB)
  DEV(RSTTEST) OBJTYPE(*FILE) RSTLIB(DEDNEW)
  OUTPUT(*PRINT) ) PASSWORD('bills00000')
```

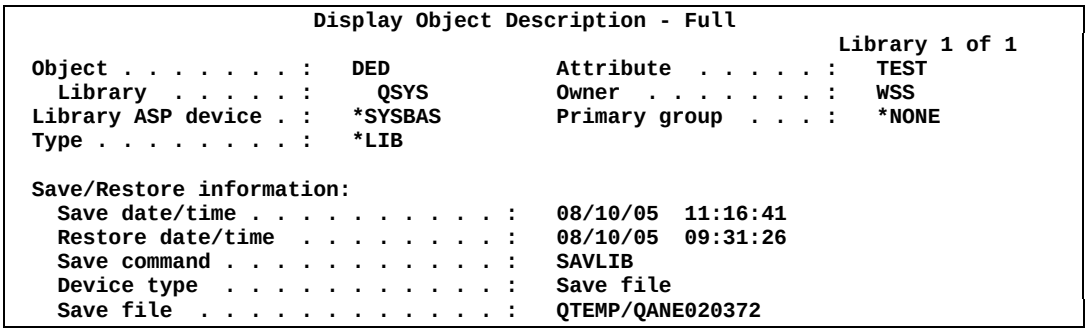
```
UNZIP Archive: /yourpath/bills/testsavx1.zip
Archive Comment:"SecureZIP for i5/OS"
Searching Archive /yourpath/bills/testsavx1.zip  for files to extract
Command:<RSTOBJ OBJ(TESTFILE) SAVLIB(TESTLIB) OBJTYPE(*FILE)
RSTLIB(DEDNEW) OUTPUT(*PRINT)>
Extracting file SAVLIB01_DEDSAV01
Inflating *iPSRA:SAVLIB01_DEDSAV01 iPSRA File
Restore Operation encountered an error.  See Job Log of PKZRSTA restore
job for further details.
SecureUNZIP  extracted     0 files
SecureUNZIP found     1 file(s) in Error


                      Additional Message Information

 Message ID . . . . . :   AQZ1007
 Date sent  . . . . . :   08/09/05      Time sent  . . . . . :   09:54:57

 Message . . . . :   Restore Operation encountered an error.  See Job Log of
   PKZRSTA restore job for further details.
```

```
 DSPSPLF FILE(QPJOBLOG) JOB(019721/EVWSS/PKZRSTA) for job log and
   detail on why the restore operation failed.  Possible problem may be that
   some or all of the objects may not have been restored due to some restore
   setting.

Since OUTPUT(*PRINT) was in effect you could view the restore output:

*...+....1....+....2....+....3....+....4....+....5....+....6....+....7....
 5722SS1 V5R3M0 040528                RESTORE OBJECT INFORMATION
 OBJECT NAME  . . . . :   TESTFILE
 SAVE LIBRARY . . . . :   TESTLIB
 OBJECT TYPE  . . . . :   *FILE
 SAVE FILE NAME . . . :   QANE019357
 SAVE FILE LIBRARY  . :   QTEMP
 LABEL  . . . . . . . :   *SAVLIB
 OPTION . . . . . . . :   *ALL
 MEMBER OPTION  . . . :   *MATCH
 SAVE DATE/TIME . . . :
 ALWOBJDIF. . . . . . :   *NONE
 RESTORE LIBRARY  . . :   DEDNEW
 RESTORE ASP  . . . . :   *SAVASP
Specified file for library TESTLIB not found.
                         * * * * * E N D  O F  L I S T I N G * * * * *

Or from the DSPSPLF FILE(QPJOBLOG) JOB(019721/EVWSS/PKZRSTA) the job log will
show the actual IBM Restore error messages:

CPF3806    Diagnostic           20   08/09/05  09:54:57.125128  QANESERV     QSYS
 From module . . . . . . . :   QANESERV
 From procedure  . . . . . :   QaneSendPgmMsg__FP14qanec_CTLBLK_tPcT2iN24
 Statement . . . . . . . . :   19
 To module . . . . . . . . :   QP0ZPCPN
 To procedure  . . . . . . :   InvokeTargetPgm__FP11qp0z_pcp_cb
 Statement . . . . . . . . :   187
 Message . . . . :   Objects from save file QANE019357 in QTEMP not restored.
   Cause . . . . :   The library name in the save file does not match the
   library name that you specified in the SAVLIB parameter. Recovery  . . . :
   Use the DSPSAVF command to display the save file and to determine the
   library from which the objects were saved. Specify the correct library in
   the SAVLIB parameter and try the command again.

CPF3780    Diagnostic           30   08/09/05  09:54:57.125152  QANESERV     QSYS
 From module . . . . . . . :   QANESERV
 From procedure  . . . . . :   QaneSendPgmMsg__FP14qanec_CTLBLK_tPcT2iN24
 Statement . . . . . . . . :   19
 To module . . . . . . . . :   QP0ZPCPN
 To procedure  . . . . . . :   InvokeTargetPgm__FP11qp0z_pcp_cb
 Statement . . . . . . . . :   187
 Message . . . . :   Specified file for library TESTLIB not found.
   Cause . . . . :   The data in the save file or on the tape, diskette or
   optical volume did not match the specified parameters. Recovery  . . . :
   See the previously listed messages. If the restore operation is from
   diskette, tape or optical, display the contents of the volume using the
   appropriate display command specifying the DATA(*SAVRST) parameter. If the
   restore operation uses a save file, display the contents of the save file
   (DSPSAVF command). Correct any errors and then try the request again.
```

iPSRA Example 7

The example below shows how the save information is depicted for an object that was saved with PKZIP iPSRA and UPDHST(*YES) for the save command. Notice that the save file shows the save library of QTEMP and the save file as QANExxxxxx. This is an internal representation of the save API process. The device type will show as a save file.

➔ **DSPOBJD OBJ(DED) OBJTYPE(*LIB) DETAIL(*FULL)**


Display Object Description - Full
Library 1 of 1
Object . . . . . : DED
Attribute . . . . . : TEST
Library . . . . . : QSYS
Owner . . . . . : WSS
Library ASP device . : *SYSBAS
Primary group . . . : *NONE
Type . . . . . : *LIB
Save/Restore information:
Save date/time . . . . . : 08/10/05 11:16:41
Restore date/time . . . . . : 08/10/05 09:31:26
Save command . . . . . : SAVLIB
Device type . . . . . : Save file
Save file . . . . . : QTEMP/QANE020372

7 PKZIP Command

PKZIP Command Summary with Parameter Keyword Format

If the OS/400 command prompt screen is to be used, the command format is simply: PKZIP. There also is a command PKZSPOOL which is the same command as PKZIP, but has the parameter TYPFL2ZP set to *SPL for spool files. The parameters are also re-sequenced to give preference to parameters dealing specifically with spool files.

The command prompt screen is displayed when Enter or PF4 is pressed. The parameter keywords are displayed on this screen, together with the available keyword options. The required options can be selected before PF4 is pressed to accept the selections. If the command and parameter keywords are entered together on the command line the required format is:

PKZIP keyword1(option) keyword2(option) . . . keyword*n*(option)

Keywords are demarcated by spaces. The keyword “ARCHIVE” is the only positional keyword where the keyword is not required. Whenever the word “path” is used, its meaning depends on the file system that is being used. If IFS is used, path refers to the open system true path type. If the library systems or *DB is used, path means library/file and then the file name refers to the member name.

```
TYPE(       *ADD                    )
                {*UPDATE}
                {*FRESHEN}
                {*MOVEA}
                {*MOVEF}
                {*MOVEU}
                {*DELETE}


ADVCRYPT(   {ZIPSTD}                      )
                {AES128}           (SecureZIP Only)
                {AES192}           (SecureZIP Only)
                {AES256}           (SecureZIP Only)
                {3DES}             (SecureZIP Only)
                {DES}              (SecureZIP Only)
                {RC4_128}          (SecureZIP Only)


ARCHIVE(    Archive Zip File name with path       )
    Archive to create                { archive file name with path }
    Optional Input Archive File      { archive file name with path }
```

```
    Output Archive File Disposition {*DEFAULT}
                                    {*PROTECT}
                                    {*OVERWRITE}


ARCHTEXT(   {*NONE}               )
              Archive File Text description

AUTHCHK(    Authenticators        )          (SecureZIP Only)
            Authenticate Type           {*FILE}
                                        {*ARCHIVE}
                                        {*ALL}
            Lookup Type                 {*DB }
                                        {*LDAP}
                                        {*FILE}
                                        {*MBRSET}
                                        {*INLIST}
                                        {*SPONSOR} (SecureZip Partner (write mode)
              only)
            Recipient                   {Recipient String}
            Passphrase (if Private)    {Certificate passphrase}
            Required                    {*RQD }
                                        {*OPT}

AUTHPOL   (  Authenticate Filters:  )       (SecureZIP Only)
Validate Level        { *SYSTEM }
                      {*WARN }
                      {*VALIDATE}
                      {*REQUIRED}
Validate Level        { *NONE }
                      {*ARCHIVE }
Filters               {*SYSTEM }
                      {*ALL}
                      {*NONE}
                      {*TAMPER}
                      {*TRUSTED}
                      {*EXPIRED}
                      {*REVOKED}
                      {*NOTAMPER}
                      {*NOTTRUSTED}
                      {*NOTEXPIRED}
                      {*NOTREVOKED}

COMPAT(     {*NONE}               )
              {*PK400}

COMPRESS(   Compression options   )
  Level         {*SUPERFAST}
                {*FAST}
                {*NORMAL}
                {*MAX}
                {*STORE}
                {*TERSE)
                {E1 thru E9}
  Method        {*DEFLATE32}
                {*DEFLATE64}
                {*STORE}
                {*TERSE)

CRTLIST(    {*NONE}               )
              path/filename
              {*SIMULATE}


CVTDATA(    External Pgm Conversion Extended Data)


CVTFLAG(    {*NONE}               )
              External Pgm Conversion Flags
```

```
CVTTYPE(    {*NONE}                )
               {*DROP}
               {*SUFFIX}

DATEAB(     mmddyyyy                      )

DATETYPE(   {*NO}                  )
               {*BEFORE}
               {*AFTER}

DBSERVICE(  ({*NO}                 )
               {*YES)

DELIM (     ({CRLF}                )
               {CR }
               {LF }
               {LFCR }

DFTARCHREC( {132}                  )
               {decimal number}

DIRNAMES(   {*YES}                 )
               {*NO}

DIRRECRS(   {*NONE}                )
               {*FULL}
               {*NAMEONLY}

ENTPREC(    Lookup Type; Recipient; Passphrase; Required )   (SecureZIP Only)
            Lookup Type                {*DB }
                                       {*LDAP}
                                       {*FILE}
                                       {*MBRSET}
                                       {*INLIST}
                                       {*SPONSOR} (SecureZip Partner (write mode)
               only)
            Recipient                  {Recipient String}
            Passphrase (if Private)     {Certificate passphrase}
            Required                   {*RQD }
                                       {*OPT}
ENCRYPOL    (  Encryption Filters:                )         (SecureZIP Only)
            Validate Level         {*SYSTEM }
                                   {*WARN }
                                   {*VALIDATE}
            Filters                {*SYSTEM }
                                   {*ALL}
                                   {*NONE}
                                   {*TRUSTED}
                                   {*EXPIRED}
                                   {*REVOKED}
                                   {*NOTTRUSTED}
                                   {*NOTEXPIRED}
                                   {*NOTREVOKED}


EXCLFILE(   {*NONE}                )
               path/filename

EXCLUDE(    file_specification 1,         )
               file_specification 2,
               file_specification n

EXTRAFLD(   {*YES}                 )
               {*NO}
               {*CENTRAL}
               {*LOCAL}
```

```
                {*BOTH same as *YES}


ERROPT(     {*END}                 )
                {*SKIP}

FILES(      file_specification 1,         )
                file_specification 2,
                file_specification n
            OR *COPY
FILESTEXT(  {*NO}                  )
                {*ALL}
                {*NEW}
                {*UPDATE}

FILETYPE(   {*TEXT}                )
                {*BINARY}
                {*EBCDIC}
                {*FIXTEXT}
                {*DETECT}

FNE(        {Create FNE | Overwrite FNE}   )       (SecureZIP Only)
                {*YES | *NO}

FTRAN(        {*ISO88591}          )
              {*INTERNAL}
               Member Name

GZIP(       {*YES}                 )
                {*NO}

IFSCDEPAGE( {*NO}                  )
                Code-page

INCLFILE(   {*NONE}                )
                path/filename

MSGTYPE(    Outlist Details:       )
  Type          {*BOTH}
                {*PRINT}
                {*SEND}

  Licence Info {*NORMAL}
                {*SHORT}
                {*NONE}

PASSWORD(   Archive Passphrase           )

PKOVRTAPF (  Archive Tape Overrides:                     )
            Tape Device            {*TAPF }
                                   { Tape Devive }
            Tape File Label        {*TAPF }
                                   { Tape Header Label}
            Tape Sequence Nbr      {*TAPF }
                                   { 1-16777216 }
                                   {*END}
            File expiration date   {*TAPF }
                                   { Date}
                                   {*NONE }
                                   {*PERM}
            End Of Tape Option     {*TAPF }
                                   {*LEAVE}
                                   {*REWIND}
                                   {*UNLOAD}
```

```
SELFXTRACT (   {*MAINTAIN}        )
               {*REMOVE}
               {WINDOWS}
               {AIX}
               {HP_UNIX}
               {SUN_UNIX}
               {LINUXINTEL}

SFUSER (   {*CURRENT}             )
               {user id 1}
               {user id 2}
               {user id 5)

SFQUEUE (  {*ALL}                 )
               {Library/OUTQ }

SFFORM (   {*ALL}                 )
               {*STD}
               {Spool File Form Type }

SFUSRDTA ( {*ALL}                 )
               {Spool File User data}

SFSTATUS ( {*ALL}                 )
               {*READY}
               (*HELD }
               {*CLOSED}
               {*SAVED }
               {*PENDING}
               {*DEFERRED}

SFJOBNAM ( {blanks }                    )
               {*}
               {Job-name//Userr-Name/Job Number}

SFTARGET ( {*SPLF}                )
               {*TEXT}
               {*TEXT1}
               {*TEXT2}
               {*TEXTFC}
               {*PDF}
               {*PDFLETTER}
               {*PDFLEGAL}


SFTGFILE ( {*GEN1}                )
               {*GEN2}
               {*GEN1P}
               {path/filename }

SIGNERS(   Signer         )                        (SecureZIP Only)
           Signing Type                 {*FILE}
                                        {*ARCHIVE}
                                        {*ALL}
           Lookup Type                  {*DB }
                                        {*LDAP}
                                        {*FILE}
                                        {*MBRSET}
                                        {*INLIST}
           Recipient                    {Recipient String}
           Passphrase (if Private)     {Certificate passphrase}
           Required                     {*RQD }
                                        {*OPT}

SIGNPOL   (  Signing Filters:                     )   (SecureZIP Only)
           Validate Level          {*SYSTEM }
                                   {*WARN }
                                   {*VALIDATE}
```

```
           Filters                {*SYSTEM }
                                  {*ALL}
                                  {*NONE}
                                  {*TRUSTED}
                                  {*EXPIRED}
                                  {*REVOKED}
                                  {*NOTTRUSTED}
                                  {*NOTEXPIRED}
                                  {*NOTREVOKED}


STOREPATH( {*NO}                  )
               {*YES}

SPLFILE (  {*ALL}                 )
               {Spool File Name }

SPLNBR (   {*ALL}                 )
               {*LAST}
               {Spool File Number 1-9999}

STOREPATH( {*NO}                  )
               {*YES}

TMPPATH(   {*CURRENT}             )
               Temporary Path

TRAN(        {*ISO88591}          )
             {*INTERNAL}
              Member Name

TYPARCHFL( Archive Type File      )
   Type        {*DB}
               {*IFS}
               {*TAP}
               {*XDB}

  Check ZIP64  {*NONE}
               {*WARN}
               {*FAIL}

TYPFL2ZP(      {*DB}                    )
               {*IFS}
               {*IFS2}
               {*DBA}
               {*SPL}

TYPLISTFL(     {*DB}                    )
               {*IFS}

VERBOSE(   {*NORMAL}              )
               {*NONE}
               {*ALL}
               {*MAX}

VPASSWORD( Archive Verify Passphrase    )
```

PKZIP Command Keyword Details

TYPE

TYPE(ADD|DELETE|EXTRACT|FRESHEN|MOVEA|MOVEF|MOVEU|UPDATE |VIEW)

The TYPE keyword specifies the type of action PKZIP should perform on the ZIP archive.

The possible actions are:

***ADD**	The *ADD option is the default and adds a selection of files to the archive file. If an archive is already present, it will be written over by the new archive file.
***UPDATE**	The *UPDATE option updates files which are already in the archive file with a newer version and will also add newly selected files that are not present in the archive file.
***FRESHEN**	The *FRESHEN option updates ONLY the files which already exist in an archive file. If the date/time of the file is newer than the date/time of the file in the archive, the file will be compressed and replace the one in the archive.
***MOVEA**	The *MOVEA (Move and Add option) option performs the *ADD option, and upon completion of a successful PKZIP command, the actual file will be deleted.
***MOVEF**	The *MOVEF (Move and Freshen option) option performs the *FRESHEN option, and upon completion of a successful PKZIP command, the actual file will be deleted.
***MOVEU**	The *MOVEU (Move and Update option) option performs the *UPDATE option, and upon completion of a successful PKZIP command, the actual file will be deleted.
***DELETE**	The *DELETE option removes entries from the archive file based upon the selection of FILES and EXCLUDE parameters. The format of the FILES and EXCLUDE parameters should be in the format of the files as seen in the archive.

ADVCRYPT

ADVCRYPT(ZIPSTD|AES128|AES192|AES256|DES|3DES|RC4_128 PKWARE|BSAFE)

*Note: PKZIP for i5/OS only support *NONE and ZIPSTD options.*

When a ZIP action is requested to save a file in an archive, and a passphrase is provided, ***SecureZIP for i5/OS*** will use an encryption method to protect the data.

This command value specifies which algorithm to employ.

Possible encryptions are:

ZIPSTD	This algorithm is the original algorithm used in PKZIP 2.x products and is compatible with other PKZIP 2.04g products that support standard encryption. Unless the installation defaults module has been tailored differently, this is the default value for ***PKZIP for i5/OS*** if you choose to encrypt a file.
***NONE**	No Encryption
AES128	Advanced Encryption Standard 128-bit key algorithm, also known as Rijndael.
AES192	Advanced Encryption Standard 192-bit key algorithm, also known as Rijndael.
AES256	Advanced Encryption Standard 256-bit key algorithm, also known as Rijndael. This is the default value for ***SecureZIP for i5/OS***.
DES	Data Encryption Standard.
3DES	Triple Data Encryption Standard.
RC4_128	RC4 is a stream cipher created by RSA.

Usage Notes:

PKUNZIP will detect automatically which encryption method was specified during the ZIP process and operate accordingly.

During a PKZIP (ZIP) run, only one encryption method may be specified, and that method will be used for each file that is operated on.

By executing PKZIP at different times, various files within the archive may be saved with differing levels (and types) of encryption. That is, some files may not be protected at all, while others may have different methods and/or passphrases.

A "+" character is shown in a view to indicate standard encryption protection is used for a file.

A "!" character is shown in a View to indicate advanced encryption (AES) protection is used for a file.

ARCHIVE

ARCHIVE(archive name, (option)input archive name, out archive Disp)

```
Archive Zip File:
       Archive Name  . . . . . .   _____________________________________________
       (Optional) Input  . . . .   *NONE                                        _
       Output Archive Disp.  . .   *DEFAULT      *PROTECT, *OVERWRITE...
```

Or

ARCHIVE ('/yourpath/mypath/myarch.zip')
ARCHIVE ('/yourpath/mypath/myarch.zip' *NONE *OVERWRITE)
ARCHIVE '/yourpath/mypath/myarch.zip' '/yourpath/mypath/oldarch.zip')
ARCHIVE ('MYLIB/MYARCH/NEWZIP1' 'MYLIB/MYARCH/OLDZIP0' *PROTECT)

This parameter specifies the archive files for output and/or input. Currently there are 3 entries for the ARCHIVE parameter (Archive File to create, Optional Input Archive file, and output file disposition).

Archive to create (archive file name with path)

Specifies the path/file name or the library/file name of the ***PKZIP**^{i}* archive to be processed. If the file exists, PKZIP will overwrite the file, otherwise PKZIP will create the file for you. Depending on which file system you choose, the path or library must exist. This is a required parameter.

Optional Input Archive File (archive file name with path)

Specifies the path/file name or the library/file name of an archive the will be used as input. This parameters provides the ability to have an input archive to update but this archive is preserved and not updated. The files in the archive will be copied to the new updated archive along with any new file selections. If an existing archive is to be updated with the same archive name then using the "archive to create" parameter is only required.

Output Archive File Disposition (*DEFAULT| *PROTECT|*OVERWRITE)

Specifies the output archive's disposition if it exist.

***DEFAULT**	This option provides backward compatibility to version prior to 8.2. If no input archive is provided, this option is set to *OVERWRITE. If an Inputted archive is provided then this option will be set to *PROTECT.
***PROTECT**	If the output archive file exist, do not overwrite the archive and fail the run.

***OVERWRITE** If the output archive exist, then overwrite the archive with the new or updated archive.

NOTE: archive file name with path:

The format of "archive file name with path" depends on whether you will be using the archive file in the library file system, or the IFS (Integrated File System).

See parameter TYPARCHFL for file system type information.

Library File System

Format is *library/file(member)*. If *member* is omitted, it will be created with the file name. If the file is not found, it will be created with a default length specified in parameter DFTARCHREC (which has a default of 132). If you want to create a file manually to use a larger record length, create it with no members and with the parameter MAXMBRS with *NOMAX, or with a high excepted limit. If the Library is not specified, the file name will be searched using *LIBL. If the file name is not found, the file will be created in the users *CURLIB. If a library is specified and does not exist, PKZIP will create the library.

Integrated File System (IFS)

Open system path followed by the archive file name. The path and file name can up to 256 characters and may contain embedded spaces.

ARCHTEXT

ARCHTEXT(*NONE| Archive File Text description)

Specifies text that will be stored in the archive as the archive's file comment.

***NONE** No new archive comment will be stored.

***DEFAULT** The default PKWARE comment will be stored.

***CLEAR** Clear any comment that may be stored in an archive.

Archive File Text description

Up to 255 characters that are stored as the archive's file comments.

AUTHCHK

Requires SecureZIP

```
Authenticator Certificates:
    File/Archive   . . . . . .   * ARCHIVE     *ARCHIVE
    LookUp Type  . . . . . . .   *DB           *DB, *LDAP, *FILE, *MBRSET...
    Authenticator  . . . . . .   _____________________________
    Passphrase (If Private)  . . _____________________________
    Required . . . . . . . . .   *RQD          *RQD, *OPT
               + for more values  _
```

Or

```
AUTHCHK((*ARCHIVE *MBRSET
   'pkwareCertAdmin04.pfx'  (passphrase) *RQD))
AUTHCHK((*ARCHIVE *FILE
   '/yourpath/PKWARE/Cstores/public/pkwareCertAdmin04.cer'  () *RQD))
AUTHCHK((*ARCHIVE *DB
   'EM=bill.somebody@pkware.com'  () *OPT))
AUTHCHK((*ARCHIVE *INLIST 'ATEST/INLIST(ENGNEER1)' *N)
```

This parameter specifies that digital signature authentication processing should be performed for specific signers. Separate authentication processing may be specified for either the archive central directory or files by using multiple commands. Optionally, specific signers may be specified to authenticate against. This parameter is used in conjunction with the AUTHPOL parameters and its settings.

It is possible that more than one certificate may be returned for a single common name or email search. As a result, each one will be added to the list of validating sources.

When no specific certificates are requested, any signatories found in the archive are validated in accordance with the systems or current AUTHPOL Filters policy settings.

There are five options for AUTHCHK.

Authenticator Type File/Archive (*ARCHIVE)

Indicates the type of archive authentication to do. If the lookup type is *INLIST, then this option will be ignored and will pickup from the records in the inlist file.

- *ARCHIVE - The archive directory will be authenticated with this authenticator.

Lookup Type (*DB |*FILE |*LDAP |*MBRSET |*INLIST |*SPONSOR)

The lookup type would be the type of authenticator search to be used for the authenticator string to look up the public key.

- *DB - The authenticator string is defined to search using the certificate locator database to access the digital certificate.
- *FILE - The authenticator string is defined to read a specific file in a specific path in the IFS in order to access the digital certificate.

- *LDAP - The recipient string is defined to search using the LDAP server to access the digital certificate.
- *MBRSET - The authenticator string is defined to read this specific file from the enterprise public certificate store to access the digital certificate.
- *INLIST- The authenticator string defines a specific file that will contain one to many AUTHCHK. The TYPLISTFL parameter must specify the file type for the inlist.
- If lookup type is *SPONSOR, the authenticator string is Sponsor Auth file stored in the '…/Sponsor/Auth' folder. If the authenticator string is all numeric, the name will automatically be formatted as A0000000.p7, assuming that the number is the sponsor ID number. *(Write mode only)*

Authenticator (The authenticator string name)

The authenticator string format depends on what was specified for the lookup type.

- If lookup type is *DB, the authenticator string will either be an email address or the common name of the certificate. This depends on the configuration setting in PKCFGSEC parameter CERTDB. To override the default selection mode, you can prefix the string with EM= for email, or CN= for the common name.

  For example:

  AUTHCHK((*ARCHIVE *DB 'CN=bill somebody' () *RQD))

- If lookup type is *FILE, the authenticator string is defined to read a specific file in a specific path of the IFS. This file should be a public key X.509 file or public key X.509 certificate with a private key file.

  For example:

  AUTHCHK((*ARCHIVE *FILE '/yourpath/PKWARE/Cstores/public/pkwareCertAdmin04.cer' () *RQD))

  The digital certificate file 'pkwareCertAdmin04.cer' will be in the full path '/yourpath/PKWARE/Cstores/public'.

- If type is *LDAP, the authenticator string will either be an email address or the common name of the certificate depending on the search mode configuration setting in PKCFGSEC parameter LDAP. To override the default selection mode, you can prefix the string with EM= for email address, or CN= for the common name.

  For example:

  AUTHCHK ((*ARCHIVE *LDAP 'bill.somebody@pkware.com' () *RQD)
  (*ARCHIVE *LDAP 'CN=bill somebody' () *OPT))

- If lookup type is *MBRSET, the authenticator string is defined to read a specific file from the public certificate store and/or the private certificate store of the IFS. This file should be a public key X.509 file or public key X.509 certificate with a private key file.

  For example:

  AUTHCHK((*ARCHIVE *MBRSET 'pkwareCertAdmin04.cer' () *RQD))

The digital certificate file ‘pkwareCertAdmin04.cer’ will be in the full path of the public certificate store defined in the enterprise security configuration public store (parameter CSPUB). If a passphrase is included, the file is searched for in the enterprise security configuration private store (parameter CSPRIV).

- If lookup type is *INLIST, the authenticator string defines a full file name of an input list file that contains records of AUTHCHK shortcut parameters. The type of file will exist in the QSYS library file system if TYPLISTFL(*DB) is set and will be a path file name in the IFS if TYPLISTFL(*IFS) is set. The format of the AUTHCHK shortcut parameters are defined below in the *INLIST usage section.

Passphrase

This designates the passphrase that is *required* for a private key certificate with a private key (PKCS#12 file). When a value is specified, the target must be an X.509 PKCS#12 public key certificate with the private key.

The PASSWORD value may contain blanks and is delimited by the closing right parenthesis ")" of the signing command.

Required (*RQD|*OPT|*SAME)

If *RQD, then this authenticator *must* be found during the selection, and the certificate *must* be a valid certificate with a private key, or the ZIP/UNZIP run will fail.

Usage Notes:

Passphrases are masked out in all output displays.

A local certificate store configuration is required to complete the TRUST processing of this command.

Processing is terminated if none of the requested certificates can be accessed, regardless of the “R” required flag. If multiple requests are made and at least one signature is found, processing continues normally.

For inlist that contains a passphrase to open a private certificate, make sure that the security is sufficient to only allow the owner of the certificate to have read access. Otherwise this would leave a security hole where other users could browse the passphrase.

***INLIST Usage:**

If *INLIST is defined on the AUTHCHK parameter, then the authenticator filed will be a file that SecureZIP will read to include the authenticator. The format is very similar to the AUTHCHK parameter described above except that each line authenticator starts with “{AUTHCHK=” and is terminated by the “}” character, with the semi-colon “;” as a separator for each entry.

{AUTHCHK=Authenticator Type, Lookup Type; Authenticator; Passphrase; Required}

Authenticator *Type*	See Authenticator Type in AUTHCHK
Lookup Type	See Lookup Type in AUTHCHK excluding the INLIST

Authenticator	See Authenticator in AUTHCHK.
Passphrase	See Passphrase in AUTHCHK.
Required	See Required in AUTHCHK, but use RDQ for *RQD and OPT for *OPT.

Examples:

Sample 1: tstauth_db1.inlist.

```
{AUTHCHK=ARCHIVE;DB;EM=PKTESTDB4@nowhere.com;;RQD}
```

Sample 2: tstauth_mb2.inlist.

```
{AUTHCHK=ARCHIVE;MBRSET;pktestdb3.pfx;PKWARE;RQD}
```

AUTHPOL

Requires SecureZIP

```
Authenticate Filters:
    Validate Level     . . . . .   *SYSTEM       *VALIDATE, *WARN, *NONE...
    Validate Type      . . . . .   *ARCHIVE      *ARCHIVE, *NONE
       Filters . . . . . . . . .   *SYSTEM       *SYSTEM, *ALL, *NONE...
               + for more values
```

Or

AUTHPOL(*WARN *ARCHIVE (*SYSTEM))
AUTHPOL(*WARN *FILE (*NOTTRUSTED))
AUTHPOL(*SYSTEM *ALL (*ALL *NOTEXPIRED))

This parameter defines the processing options and filters that should apply if a signed file or signed archive is encountered.

Validate Level (*VALIDATE |*WARN |*REQUIRED |*SYSTEM)

The validate level specifies the type of authentication processing that should take place if a signed archive is encountered. The default is *SYSTEM and, unless it is modified, SecureZIP will use the enterprise setting from PKCFSEC.

- *VALIDATE – Indicates that when authentication takes place and a failure occurs based on the filters, the run will be considered a failure and the message issued when the job terminates will indicate one or more errors during the run.
- *WARN - Indicates that when authentication takes place and a failure occurs, the failure is only to be considered a warning. The messages at the end of the run will not consider any failed authentications as errors.
- *REQUIRED – Indicates that authentication *must* take place and that, if any failure occurs based on the filters, the run will be considered a failure, and the message issued when the job terminates will indicate one or more errors

occurred during the run. If the archive has not been signed, then an error will be issued.

- *SYSTEM – Indicates the authentication processing that is set in the environmental setting will be used.

Validate Type (*ARCHIVE |*NONE)

The validate type specifies that archive authentication should take place if an archive has been signed. The default is *NONE and anything other than *NONE requires the Enhanced Encryption Feature.

- *ARCHIVE - Indicates that only a signed archive will be authenticated.
- *NONE - Indicates no authentication will take place even though a file or archive has been signed.

Filters (*SYSTEM |*ALL |*NONE |*TAMPER |*TRUSTED |*EXPIRED |*REVOKED |*NOTAMPER |*NOTTRUSTED |*NOTEXPIRED |*NOTREVOKED)

The authentication filter policies settings are defined in the enterprise security file supplied by the SecureZIP administrator (See PKCFGSEC). These global policy settings can be revised with sub-parameter values. The variables are cumulative from the global setting.

- ***SYSTEM** – All filter policies are from the global settings.
- ***ALL** - This sub-parameter activates all levels of authentication. If followed by negating sub-levels, then all but those negating levels are activated. For example: *ALL NOTEXPIRED means that expired certificates will not cause an authentication error, but TRUST and TAMPERCHECK must both be satisfied.
- ***NONE** – Will negate all the policies.
- ***TAMPER** – This sub-parameter signifies that a verification of the data stream should be done against the digital signature.
- ***TRUSTED** – This sub-parameter signifies that the entire certificate authority chain must be validated. This includes locating the root (self-signed) certificate on the local system.
- ***EXPIRED** – This sub-parameter signifies that certificate date range validation should be performed on the certificates (including the certificate authority chain). Although the term “expired” is used, a certificate that has not yet reached its valid data range specification will fail.
- ***REVOKED** - A certificate owner may request that the issuing certificate authority declare a certificate to be revoked and thereby no longer consider that certificate to be valid. The authentication operation will fail if any of the certificates in the trust chain are found to have been revoked, or if the revocation status could not be determined
- ***NOTAMPER** – Negates the *TAMPER filter.
- ***NOTTRUSTED** – Negates the *TRUSTED filter.
- ***NOTEXPIRED** - Negates the *EXPIRED filter.
- ***NOTREVOKED** – Negates the *REVOKED filter.

COMPAT

COMPAT(*NONE|*PK400)

Specifies that PKZIP will create and store extended data field information in another supported format or previous version. At this time, only "PKZIP Version 4.0 for OS/400" is supported.

The allowable values are:

***NONE**	The extended data fields will be in ***PKZIP***^{i} versions 5.0 and above formats.
***PK400**	The extended data fields will output to the archive in the format used by "PKZIP Version 4.0 for OS/400" product. This option should be used if the archive file will be extracted by "PKZIP Version 4.0 for OS/400" and the attributes are required to create the files. The files can be extracted without this option, but the files may have to be manually created in order to have the proper attributes (such as record length and text descriptions).

COMPRESS

```
Compression:
              Level . . . . . .   *SUPERFAST    *SUPERFAST, *FAST, *NORMAL...
              Method  . . . . .   *DEFLATE32    *DEFLATE32, *DEFLATE64...
```

Or

COMPRESS(*FAST *DEFLATE64)
COMPRESS(E1 *DEFLATE32)
COMPRESS(*STORE)

This parameter specifies the speed and compression level when zipping a file. Currently there are 2 entries for the COMPRESS parameter (Level and Method).

Compression Level (*SUPERFAST| *FAST| *NORMAL|*MAX|*STORE|*TERSE |E0 thru E9)

The compression level option specifies a compression level and speed to be used. The option works in conjunction with the compression method option and specifies a depth of compression using a sliding scale of values.

The allowable values are:

***FAST**	Fast selection provides ample compression at a fast rate. Same as E2.
***SUPERFAST**	This is the default selection. This will compress in the fastest time, but will compress the files by the least amount. Same as E1.

***MAX** This level provides the maximum compression possible, but will also take the longest in time to process. Same as E4.

***NORMAL** The normal compression level provides good compression amount at a reasonable speed. Same as E3.

***STORE** No compression. Store will also be used if the other methods tried result in a file larger than the original. Same as E0.

***TERSE** This selection provides a terse compression algorithm provided with the iSeries by IBM as an API. This is much faster but is less efficient than FASTEST, and can only be decompressed on the iSeries. Do not use this option if you wish to unzip the archive on another platform.

***E0 thru E9** E0 thru E9 are custom levels that can be used to try and obtain the results based on your input files and desired time and compression results.

The following table shows the balance of degree of compression and speed of compression. The levels range from 0 (fastest speed with no compression) to 9 (highest level of compression, usually taking the longest amount of time and using the most processor time).

Synonym	*Level*	*Usage*
STORE, E0	0	No compression is performed.
SUPERFAST, E1	1	Compression Method: Deflate32 or Deflate64
FAST, E2	2	Compression Method: Deflate32 or Deflate64
NORMAL, E3	3	Compression Method: Deflate32 or Deflate64
MAXIMUM, E4	4	Compression Method: Deflate32 or Deflate64
E5	5	Compression Method: Deflate32 or Deflate64
E6	6	Compression Method: Deflate32 or Deflate64
E7	7	Compression Method: Deflate32 or Deflate64
E8	8	Compression Method: Deflate32 or Deflate64
E9	9	Compression Method: Deflate32 or Deflate64

Usage Notes:

- Compression levels 1 through 9 all work with Deflate32 and Deflate64 compression methods.
- “Maximum” is retained at level 4 to provide equivalent compression ratios with earlier releases. Higher levels may yield better compression ratios than previously obtained with “Maximum”.
- Compression results are data-stream dependent and produce non-linear results. When configuring a job for high volume processing, benchmarking

results with sample file may be of value to find the best balance between compression ratio and resources (elapsed and processor time).

- In many cases, levels 8 and 9 do not produce significant compression results over level 7.
- When compression level is STORE, or E0, the compression method will be set automatically to store.
- When migrating from earlier releases of ***PKZIP***^{i}, a difference in compression ratio/processor time can be expected for a given data stream and setting. Although internal settings have been tuned to produce similar results across the scale of levels, a specific level setting may not produce faster speeds or better compression for a data stream. If these criteria are of importance, then benchmarking should be performed to achieve the "best" fit results with the new algorithms.

Method (*DEFATE32 |*DEFLATE64 |*STORE |*TERSE)

This option specifies the algorithm to be used when compressing a file during ZIP processing. The method works in conjunction with the compression level option to specify a depth of compression.

STORE performs no compression of the data. Deflate64 (using the same level control) will usually produce better compression with less processor time than Deflate32.

The allowable values are:

***DEFLATE32**	Use the Deflate 32 algorithm.
***DEFLATE64**	Use the Deflate 64 algorithm.
***STORE**	Store Data with no compression.
***TERSE**	Use the IBM Terse algorithm.

Usage Notes:

- When compression method is store is specified, the compression level will be set automatically to *STORE.
- The GZIP specification only supports Deflate32. When GZIP(*YES) is encountered, PKZIP will automatically switch from Deflate64 or STORE to Deflate32.
- Not all non-PKWARE "ZIP compatible" products in the market support the more advanced Deflate64 algorithm. If the intended target systems support Deflate64, then it may be chosen for the best compression/speed performance.

CRTLIST

CRTLIST(*NONE| *path/filename* | *SIMULATE)

Specifies that PKZIP will create an output file with a list of entries that would have been compressed based upon the selection criteria in the FILES and EXCLUDE parameters.

See parameter TYPLISTFL for file system type.

***NONE** Default. No list file will be created.

path/filename
Enter the file path and name of the file to create. The layout depends on which file system you want to create the file in.

Library File System: The format is "library/file(member)".

Integrated File System (IFS):
The format is "path1/path2/../pathn/filename".

***SIMULATE** Will simulate the file selection and show the selection as a printed or message list instead of writing to a list file.

CVTDATA

CVTDATA(*External Program Conversion Extended Data*)

Specifies the extended data that is passed to the external program CVTNAME. When CVTFLAG is not *NONE, the contents of the parameter are passed to provide extended flexibility in controlling how the iSeries names are stored in the archive. The *System Administrator's Guide* contains more information on CVTNAME.

External Program Conversion Extended Data
Specify up to 255 bytes of unedited data which is passed to the exit program CVTNAME to assist in controlling the program logic.

CVTFLAG

CVTFLAG(*NONE| *Conversion Flags*)

Specifies the flags passed to the external program CVTNAME. These are used to control how the iSeries names are stored in the archive. The *System Administrator's Guide* contains more information on CVTNAME.

***NONE** Conversion exit is not active.

Conversion Flags Specify a five-byte flag that is passed to the exit program CVTNAME to control the program logic. If the name passed back is blank, then conversion is referred back to the setting of the CVTTYPE parameter.

CVTTYPE

CVTTYPE(*NONE|*DROP|*SUFFIX)

Specifies how the iSeries library and file names are stored in the archive. Since the length of the library name, file name, and member name can each be up to 10 characters, and MS/DOS format requires a maximum of 8 characters with an optional extension, this option allows name compatibility.

The allowable values are:

***SUFFIX**	This forces any iSeries name with more than 8 characters to create a name of 8 characters and a period(.), followed by characters 9 and 10 to be considered an extension to suffix.
***NONE**	This leaves the iSeries name as the archive name.
***DROP**	This forces any iSeries name with more than 8 characters, to drop characters 9 thru 19.

DATEAB

DATEAB(mmddyyyy)

Used with DATETYPE parameter, DATEAB specifies the date to be used to compare with the files latest modification date for file selection. The format is mmddyyyy, where “mm” is a valid month (01-12), “dd” is valid day of the month, and “yyyy” is the four digits of the year (2001).

DATETYPE

DATETYPE(*NO|*BEFORE|*AFTER)

Specifies if PKZIP should select files based upon a file modification date.

The allowable values are:

***NO**	No date selection will take place.
***BEFORE**	Files with a modification date before the date in DATETYPE will be selected.
***AFTER**	Files with a modification date on or after the date in DATETYPE will be selected.

DBSERVICE

DBSERVICE (*NO|*YES)

Specifies if the iSeries special database extended file attributes describing the database file, fields and keys are to be store in the archives. This will force the

option EXTRAFLD(*YES). The database will also be stored in binary mode. This mode can produce larger archive files.

The allowable values are:

***NO**	Does not store database extended services attributes.
***YES**	Stores the database extended service attributes in the archive file and treat non-SAVF as a database.

DFTARCHREC

DFTARCHREC(132|Record Length)

Specifies the record length to use when creating an archive file in the QSYS library system. If the TYPARCHFL parameter is *DB or *XDB, and the archive file does not exist, the archive file will be created with the record length specified in this parameter.

Note: A large record length will leave a high residual number if only one byte is use in the last record.

The allowable values are:

132	Default is record length of 132 to match previous versions.
Record Length	A decimal number from 50 to 32000.

DELIM

DELIM(CRLF |CR |LF |LFCR)

When compressing a text file (not binary), the DELIM parameter specifies what characters are to be appended at the end of records to serve as delimiters. The delimiter is removed from the record when it is decompressed.

The allowable values are:

CRLF	This is the default selection. Specifies for ***PKZIP^{i}*** to use the default delimiter CR-LF (x'0D0A') at the end of each text record.
CR	Appends an ASCII carriage return (hex 0D).
LF	Appends an ASCII line feed character (hex 0A).
LFCR	Appends an ASCII line feed character (hex 0A0D).

Note that transfers of MS-DOS records uses a CRLF for a delimiter, while UNIX records use a LF.

DIRNAMES

DIRNAMES(*YES|*NO)

Specifies to store directories as an entry. This is valid only for files in IFS.

***YES**	Store the directories as entries in the archive.
***NO**	Do not store directories as an archive entry.

DIRRECRS

DIRRECRS(*NONE|*FULL|*NAMEONLY)

IFS only. Specifies whether to search recursively through directories for file selection, or only search the current, specified directory.

The allowable values are:

***NONE**	Search only the current, specified directory.
***FULL**	Search through all directories by starting with the current, specified directory for selected files. If *FULL is used, and * is for file selections, all files found in all directories below the current directory will be selected.
***NAMEONLY**	To be considered a hit, the full path and file name must match the selection statements exactly.

ENTPREC

Requires SecureZIP

```
Encryption Recipients       :
     LookUp Type  . . . . . .   *DB     *DB, *LDAP, *FILE...
     Recipient  . . . . . . . _____________________________________

     Passphrase (If Private)  . _____________________________________
     Required . . . . . . . . .   *RQD          *RQD, *OPT
               + for more values   _
```

Or

ENTPREC((*MBRSET 'pkwareCertAdmin04.cer' () *RQD))
ENTPREC((*FILE '/yourpath/PKWARE/Cstores/public/pkwareCertAdmin04.cer' () *RQD))
ENTPREC((*FILE '/yourpath/PKWARE/Cstores/public/pkwareCertAdmin04.pfx' ('mypassphrase') *RQD))
ENTPREC((*DB 'EM=bill.Somebody@pkware.com' () *RQD))
ENTPREC((*LDAP 'EM=bill.Somebody@pkware.com' () *RQD))

ENTPREC((*INLIST 'ATEST/INLIST(ENGNEER1)' *N)

The encryption recipient parameter defines one to many Recipients which is to be included for the ZIP process. This parameter allows 1-4 types of certificate searches to take place along with providing the ability for an include file that may contain the recipients.

The specification of this recipient ENTPREC parameter, triggers encryption to take place during ZIP processing utilizing the found recipients along with any passphrase that may be entered.

There are four entries for the ENTPREC parameter (lookup type, recipient, passphrase, and required).

Lookup Type (*NONE |*DB |*LDAP |*FILE |*MBRSET |*SPONSOR |*SAME)

The Lookup type would be the type of recipient search that will be used for the recipient string.

- *DB - The recipient string is defined to search using the Certificate Locator Database to access the digital certificate.
- *LDAP - The recipient string is defined to search using the LDAP server to access the digital certificate.
- *FILE - The recipient string is defined to read a specific file in a specific path in the IFS in order to access the digital certificate.
- *MBRSET - The recipient string is defined to read this specific file from the enterprise public certificate store to access the digital certificate.
- *INLIST - The recipient string defines a specific file that will contain 1 to many recipients.
- *SPONSOR - The recipient string is the encryption recipient file for a sponsoring partner. This applies only for SecureZIP Partner Read mode.

Recipient (The recipient string name)

The recipient string format depends on what was specified for the Lookup type.

- If type is *DB: The recipient string will either be an email address or the common name of the certificate. This depends on the configuration setting in PKCFGSEC parameter CERTDB. To override the default selection mode, you can prefix the string with EM= for email address or CN= for the common name.

  For example:

  ENTPREC((*DB ‘bill.Somebody@pkware.com' () *RQD)
  (*DB ‘CN=bill Somebody' () *RQD)
  (*DB ‘EM=bill.Somebody@pkware.com' () *RQD))

- If type is *LDAP: The recipient string will either be an email address or the common name of the certificate depending on the search mode configuration setting in PKCFGSEC parameter LDAP. To override the default selection mode, you can prefix the string with EM= for email address or CN= for the common name.

For example:

ENTPREC((*LDAP ‘bill.Somebody@pkware.com' () *RQD)
(*LDAP ‘CN=bill Somebody' () *OPT)
(*LDAP ‘EM=bill.Somebody@pkware.com' () *RQD))

- If type is *FILE: The recipient string is defined to read a specific file in a specific path of the IFS. This file should be public-key X.509 file or private-key X.509 certificate file.

  For example:

  ENTPREC((*FILE '/yourpath/PKWARE/Cstores/public/pkwareCertAdmin04.cer' () *RQD))

The digital certificate file ‘pkwareCertAdmin04.cer’ will be in the full path ‘/yourpath/PKWARE/Cstores/public’.

- If type is *MBRSET: The recipient string is defined to read a specific file from public certificate store / private certificate store of the IFS. This file should be public-key X.509 file or private-key X.509 certificate file.

  For example:

  ENTPREC((*MBRSET 'pkwareCertAdmin04.cer' () *RQD))

The digital certificate file ‘pkwareCertAdmin04.cer’ will be in the full path of the public certificate store defined in the Enterprise Security Configuration public store(parameter CSPUB). If a passphrase was included, the file would be searched for in the Enterprise Security Configuration private store (parameter CSPRIV).

- If the type is *INLIST: The recipient string defines a full file name of an input list file that contains records of ENTPREC shortcut parameters. The type of file will in the QSYS library file system if TYPLISTFL(*DB) is set and will be a path file name in the IFS if TYPLISTFL(*IFS) is set. The format of the ENTPREC shortcut parameters are define below in the *INLIST usage section.
- If type is *SPONSOR, the recipient string is the sponsor recipient file stored in the ‘…/Sponsor/Recip’ folder. If the recipient string is all numeric, the name will automatically be formatted as R0000000.p7, assuming that the number is the sponsor ID number.

Passphrase (Private Cert Passphrase)

The passphrase is required only if the certificate that is being selected is a private certificate. This option should be omitted if a public certificate will be utilized.

Required (*RQD|*OPT|*SAME)

If *RQD, then this recipient must be found during the selection, and the certificate must be valid, or the ZIP/UNZIP run will fail.

Usage Notes:

The ZIP process only requires a X.509 public-key format certificate to encrypt files. The UNZIP process requires X.509 private-key format certificate file to decrypt files and this will the input of a passphrase.

For inlist that contains a passphrase to open a private certificate, make sure that the security is sufficient to only allow the owner of the certificate to have read access. Otherwise this would leave a security hole where other users could browse the passphrase.

***INLIST Usage:**

If *INLIST is defined on the ENTPREC parameter, then the recipient field will be a file that SecureZIP will read to include recipients. The format is very similar to the ENTPREC parameter describe above except each line recipient starts with "{RECIPIENT=" and is terminated by the "}" character with the semi-colon ";" as a separator for each entry.

{RECIPIENT=Lookup Type; Recipient; Passphrase; Required}

Lookup Type	See Lookup Type in ENTREC excluding the INLIST
Recipient	See Recipient in ENTREC.
Passphrase	See Passphrase in ENTREC.
Required	See Required in ENTREC, but use RDQ for *RQD and OPT for *OPT.

Examples:

```
{RECIPIENT=MBRSETEM; mypassphrase;RQD}
```

Sample 1: tstpriv_db4.inlist.

```
{RECIPIENT=DB;EM=PKTESTDB4@nowhere.com;PKWARE;RQD}
```

Sample 2: tstpriv_mb3.inlist.

```
{RECIPIENT=MBRSET;pktestdb3.pfx;PKWARE;RQD}
```

Sample 3: tstpubl1.inlist.

```
{RECIPIENT=MBRSET;pktestdb3.crt;;RQD}
{RECIPIENT=MBRSET;pktestdb4.crt;;OPT}
```

Sample 4: tstpubl2.inlist.

```
{RECIPIENT=DB;EM=PKTESTDB3@nowhere.com;;RQD}
{RECIPIENT=DB;CN=PKWARE Test4;;OPT}
```

ENCRYPOL

Requires SecureZIP

```
Encryption Filters:
    Validate Level     . . . . .   *SYSTEM       *VALIDATE, *WARN, *NONE...
       Filters . . . . . . . . .   *SYSTEM       *SYSTEM, *ALL, *NONE...
               + for more values
```

Or

ENCRYPOL(*WARN (*SYSTEM))
ENCRYPOL(*WARN (*ALL *NOTTRUSTED))
ENCRYPOL(*SYSTEM (*ALL *NOTEXPIRED))

This parameter defines the processing options and filters that should apply when the ENTPREC is used to encrypt files with certificate keys.

Validate Level (*VALIDATE |*WARN |*SYSTEM)

The validate level specifies the type of encryption certificate error processing that is used if certificates are specified in ENTPREC. If *SYSTEM is specified, the enterprise setting from PKCFSEC is used. If the enterprise setting is defined as lockdown, then this parameter cannot be revised and a warning will be issued if a change is detected.

- *SYSTEM - Indicates the authentication processing that is set in the environmental setting will be used.
- *VALIDATE – Indicates that when encryption with certificates (ENTPREC parm) takes place and a failure based on the filters occurs, the run will be considered a failure and the message issued at the end will indicate one or more errors during the run.
- *WARN - Indicates that when encryption with certificates (ENTPREC parm) takes place and a failure based on the filters occurs, the failure is only considered a warning. The messages at the end of the run will not consider any failed filters for encryption certificates as errors.

Filters (*SYSTEM |*ALL |*NONE |*TRUSTED |*EXPIRED |*REVOKED |*NOTTRUSTED |*NOTEXPIRED |*NOTREVOKED)

The ENTPREC certificate filter policies settings are defined in the enterprise security file supplied by the SecureZIP administrator (see PKCFGSEC). These global policy settings can be revised with sub-parameter values, but if the enterprise setting is defined as lockdown, this parameter cannot be revised and a warning will be issued if a change is detected. The variables are cumulative from the global setting.

- ***SYSTEM** – All filter policies are from the global settings.
- ***ALL** - This sub-parameter activates all levels of authentication. If followed by negating sub-levels, then all but those negating levels are activated. For example: *ALL, NOTEXPIRED means that expired certificates will not cause an authentication error, but TRUST and REVOKE must both be satisfied.

- ***NONE** – Will negate all the policies.
- ***TRUSTED** – This sub-parameter signifies that the entire certificate authority chain must be validated. This includes locating the root ("self-signed") certificate on the local system.
- ***EXPIRED** – This sub-parameter signifies that certificate date range validation should be performed on the certificates (including the certificate authority chain). Although the term "expired" is used, a certificate that has not yet reached its valid data range specification will fail.
- ***REVOKED** - A certificate owner may request that the issuing certificate authority declare a certificate to be revoked and thereby no longer consider that certificate to be valid. The encryption certificate request will fail if any of the certificates in the trust chain are found to have been revoked or if the revocation status could not be determined.
- ***NOTTRUSTED** – Negates the *TRUSTED filter.
- ***NOTEXPIRED** - Negates the *EXPIRED filter.
- ***NOTREVOKED** – Negates the *REVOKED filter.

ERROPT

ERROPT(*END|*SKIP)

Specifies what action to take if an error occurs while processing (selecting or compressing) a spool file or using the iPSRA SAVCHGOBJ command.

If the SAVCHGOBJ finds no files to archive, then the error AQZ0368 "Save Command Found Nothing to Do" is issued. AQZ0368 follows the rules of the ERROPT parameter.

The allowable values are:

***END**	PKZIP will end without completing the compression of the file. The archive is not updated.
***SKIP**	The program will skip the file with the input error and continue to process all other files to completion. Message AQZ0022 will be issued at the end to indicate that an error occurred.

EXCLFILE

EXCLFILE(*NONE| *path/filename*)

This parameter specifies the file containing the list of files to be excluded. This can be used with or without the EXCLUDE parameter. See parameter TYPLISTFL for file system type information.

***NONE**	No list file will be processed.
path/filename	Enter the file path and name of the file to process. The layout depends on which file system you want the file created.

Library File System:
The format is "library/file(member)".

Integrated File System (IFS):
The format is "path1/path2/../pathn/filename".

EXCLUDE

EXCLUDE(file_specification1, file_specification2,... file_specification *n*)

Specifies the files and file specification patterns that will be excluded from the PKZIP run. One or more names can be specified. Each name should be in the OS/400 file system format, such as, QSYS is library/file(member) and IFS is directory/file, and can include wildcards "*" and "?."

Note: If TYPE(*VIEW) is being used, then the format for these names is the MS/DOS format.

The PKZIP program can also exclude file specifications by using the list file parameter EXCLFILE with a list of names to exclude.

Please refer to "File Selection and Name Processing" in chapter 1 for details of file specification formatting.

The valid parameter values for the FILES keyword are as follows:

'*file_specification1*'

'*file_specification2*'

'*file_specification n*'

EXTRAFLD

EXTRAFLD(*YES|*NO)

Specifies if the basic ***PKZIP***^{i} extended file attributes should be stored in the archive. Some basic file attributes are record size, library text description, file text description, etc.

The allowable values are:

***YES**	Store the basic normal iSeries file attributes. This is the default and will be the same as coding *BOTH.
***NO**	Do not store any extended attributes.
***CENTRAL**	Stores the basic normal iSeries file attributes in only the archive's central directory. This will reduce the overall archive size by only storing the attributes in the Central.
***LOCAL**	Stores the basic normal iSeries file attributes in only the archive's local directory. *Warning:* PKUNZIP only utilizes the central directory for extended data attributes.

***BOTH**	Stores the basic normal iSeries file attributes in both the archive's central directory and local directory.

Migration consideration: if the archive will be processed by an earlier release of PKZIP for OS/400™ and the attributes are required, then *BOTH should be coded.

FILES

FILES(file_specification1, file_specification2,... file_specification *n*)

Specifies the files and file specification patterns that will be selected in the PKZIP process. One or more names can be specified. Each name should be in the OS/400 file system format, such as, QSYS is library/file(member) and IFS is directory/file, and can include wildcard "*" and "?." For the IFS, the path and file name can up to 256 characters and can contain embedded spaces.

If the FILES parameter starts with a question mark (?) or a dash (-), then PKZIP assumes that a Save command is being entered to activate the iPSRA feature. For details on how to enter iPSRA commands, see chapter 6.

The key word "*COPY" as an option of FILES parameter, will copy the files from the input archive to the new archive. This can be used when creating a new archive with a different name and avoid selecting any new files.

Note: If TYPE(*VIEW) or TYPE(*DELETE) is being used, then the format for these names is the MS/DOS format.

The PKZIP program can also have file specifications selections to include by using the list file parameter INCLFILE with a list of names to select.

Files may also be excluded. See the EXCLUDE parameter.

Please refer to "File Selection and Name Processing" in chapter 1 for details of file specification formatting.

The valid parameter values for the FILES keyword are as follows:

'file_specification1'

'file_specification2'...

'file_specification n'

FILESTEXT

FILESTEXT(*NO|*ALL|*NEW|*UPDATE)

Specifies if PKZIP allows the editing (and the type of editing performed) of a file's text comments that are stored in an archive.

The allowable values are:

***NO**	No comment editing (the default).
***ALL**	Add comments or edit comments for all files in the archive.

***NEW**	Add comments only for new files that are added to the archive.
***UPDATE**	Add or edit the comments of files that are added, updated, or freshened in the archive. Only file comments of files that are affected by a change are eligible for editing.

FILETYPE

FILETYPE(*BINARY|*DETECT|*EBCDIC|*FIXTEXT|*TEXT)

Specifies whether the files selected are treated as text or binary data. For text files added to an archive, trailing spaces in each line are removed, the text is converted to ASCII (based on the translation tables) by default, and a carriage return and line feed (CR/LF) are added to each line before the data is compressed into the archive. Binary files are not converted.

The default is *DETECT; where PKZIP attempts to make a determination based on the nature of the data itself. The program will read in a portion of the data, evaluate it, and determine the appropriate process.

Note: This will lower performance time. A message will display the type used when compressing.

Use of text file option is usually faster because PKZIP has to process less data than with *BINARY, but more processing may also take place to perform the translation.

If the file is a SAVF or a database file (with DBSERVICE(*YES)), then the file will be processed as binary regardless of what option is specified.

***BINARY**	Specifies that the files selected are binary files and no translation should be performed.
***DETECT**	The PKZIP program will try to determine the data type of text or binary.
***EBCDIC**	Specifies that the files selected are text files and leaves it in EBCDIC without performing any translation. This is good only if the files are to be used on an iSeries or IBM-type mainframe. If they will be unzipped to a PC file, then a translation from EBCDIC to ASCII is required.
***FIXTEXT**	Specifies that the files selected are text files with a fixed record length based on the iSeries file's record length and translation will be performed using the translate tables specified in the TRAN option. This means the compressed file will contain records with trailing spaces followed by a CR and LF. This is only valid for QSYS library file types as files in the IFS do not contain a record length.
***TEXT**	Specifies that the files selected are text files and translation will be performed using the translate tables specified in the TRAN option.

FNE

Requires SecureZIP

FNE(*YES|*NO *YES|*NO

```
File Name Encryption     :
   Create FNE Archive  . . . . .   *NO           *NO, *YES
   Overwrite In FNE  . . . . . .   *NO           *NO, *YES
```

FNE(*NO *NO)

Specifies the activation and use of the file name encryption feature.

The first option controls the creation of an archive with file name encryption.

***NO** Do not create the new/updated archive as filename-encrypted archive.

***YES** Create the new or updated archive into a filename-encrypted archive. If the archive exists, then the security features will be defined by the inputted FnE archive. If no archive exist, the new filename-encrypted archive will use the encryption method define with ADVCRYPT parameter and the PASSWORD and/or ENTPREC parameters.

The second option controls the overwriting of a filename-encrypted input archive to remove the filename encryption. This option is used with the first option of *NO (do not create a filename-encrypted archive), and with an existing filename-encrypted archive is input for update. Coding this option to *NO indicates that you know that the input archive is filename-encrypted and you want overwrite it to produce an archive that is not filename-encrypted.

***NO** Do not allow an existing filename-encrypted archive to be changed to a non-filename-encrypted archive.

***YES** Allow an existing filename-encrypted archive to be changed to a non-filename-encrypted archive when archive is updated.

FTRAN

FTRAN(*ISO88591 |*INTERNAL| *Member Name*)

Specifies the translation table for use in translating "file names, comments, and passphrase" from the iSeries EBCDIC character set to the character set used in the archive file (normally ASCII character set). A default internal table is predefined. See Appendix D for additional information.

***ISO88591** The predefined internal table for translation. This table provides translation that is consistent with the ISO

8859-1 definitions. This table uses the EBCDIC code page 037 and the ASCII code page 819 for translation.

***INTERNAL** To provide some compatibility to pre V8 version, *INTERNAL will use the internal tables that were the default in V5 PKZIP.

membername Specify the member name in the file PKZTABLES that will be parsed and used to translate "file names and comments" files to the archive character set. The member should have the exact format of member ISO9959_1 in file PKZTABLES. See Appendix D for information on defining translation tables.

GZIP

GZIP(*YES|*NO)

If this option is set to *YES, PKZIP will create a compressed archive in the GZIP format. The GZIP format only allows for one file or member per archive and all text data is stored in ISO 8859- 1 (LATIN- 1) character set. The GZIP format is very different from the ***PKZIP***^{i} archive format, a program that can process ***PKZIP***® archives will not necessarily process a GZIP archive correctly. The GZIP archive created conforms to the GZIP specifications RFC1951 and RFC1952.

Do not use this option if the archive is to be unzipped on another platform where GZIP compatibility is not confirmed.

The allowable values are:

***YES** The PKZIP program will create a compressed archive in the GZIP format.

NO** The PKZIP program will create an archive in the ***PKZIP® format. This is the default.

IFSCDEPAGE

IFSCDEPAGE(*NO| *Code-Page*)

If this option is set to *NO, PKZIP will read IFS files using the code page that is registered for the file. Otherwise, PKZIP will read IFS files with the specified code page.

This parameter also controls the spool file ASCII conversion for *TEST and *PDF documents. When *NO is specified for spool Files, the conversion will use code page 819.

The allowable values are:

***NO** The PKZIP program will read IFS files with the code page registered for the file. If the file is a spool file, the code page 819 will be used. This is the default.

Code-Page The PKZIP program will read IFS files with the specified code page value. If the file is a spool file, it is the code page that a spool file will use for ASCII translation.

INCLFILE

INCLFILE(*NONE| *path/filename*)

This parameter specifies the file containing the list of files to be selected for inclusion. This can be used with or without the FILES parameter. See parameter TYPLISTFL for file system type information.

***NONE** No Include list file will be processed. This is the default.

path/filename Enter the file path and name of the file to process. The layout depends on which file system you want to create the file in.

Library File System:
The format is "library/file(member)".

Integrated File System (IFS):
The format is "path1/path2/../pathn/filename".

MSGTYPE

```
Outlist Details:
        Type . . . . . .   *BOTH          *BOTH, *SEND, *PRINT
        Licence Info.  . . . . .   *NORMAL        *NORMAL, *SHORT, *NONE
```

Or

MSGTYPE (*SEND)
MSGTYPE (*BOTH *SHORT)
MSGTYPE (*BOTH *NONE)
MSGTYPE (*PRINT)

This parameter specifies where displayed output will be outputted and the type of licensing splash screen to provide.

Detail Type (*BOTH |*PRINT |*SEND)

Specifies where the display of messages and information should be shown. The PKZIP program has the ability to send messages that appear on the log and/or the ability to print to stdout and stderr. If working interactively, stdout and stderr will show upon the dynamic screen. If submitted via batch, you can override them to print in an OUTQ or build a CL and save them to an outfile.

***BOTH** Send the information to the log with send message commands and also to stdout and stderr

***PRINT** Send the information to stdout and stderr

***SEND** Send the information to the log with send message commands

License Info (*NORMAL |*SHORT |*NONE)

Specify the amount of detail license/copyrights information pertaining to ***PKZIP***^{i} product that will be displayed.

***NORMAL**	Displays all license and copyright information
***SHORT**	Displays base licensing/copyrights
***NONE**	Displays only registration information

PASSWORD

PASSWORD(Archive Passphrase)

Specifies an encryption passphrase for files being added to an archive. This passphrase may be up to 260 characters in length and is case sensitive. All files selected for archiving will be encrypted using the specified passphrase. If a contingency key is coded for the enterprise, certificate-based encryption is done for the key in addition to the passphrase-based encryption. The length range of the passphrase for SecureZIP is defined for the enterprise settings by PKCFGSEC.

Note: There is no way to extract the passphrase used from the archive data. If the passphrase is forgotten, the file will become inaccessible. If files in an archive need to have different passphrases, PKZIP must be run for each passphrase required.

Since the passphrase is entered in EBCDIC, the translation table referenced in the FTRAN parameter is used to translate it to an ASCII format. In order to be able to use this passphrase on other platforms, care should be taken to assure that there is a correct translation of the EBCDIC character to a valid ASCII character. For this reason, creating a passphrase using standard character set characters is safer for use in other environments. Care should also be taken when using the FTRAN() override with a passphrase encryption. To extract passphrase-protected files, the same FTRAN() override option is required.

If the contents of the PASSWORD() parameter starts with the key word *INLIST;, then the passphrase will be retrieved from the Inlist file defined in the PASSWORD() parameter.

***INLIST Usage**

To utilize the inlist file for a passphrase, the PASSWORD parameter must start with the keyword *INLIST;. If the inlist file is not from the same file system that is set with the TYPLISTFL(*IFS/*DB) parameter, then code a *DB; or *IFS; to describe the file system where the following inlist will reside. Following the *INLIST; or *IFS;/*DB;, the file name should be specified. Using the inlist method for passphrase allows the opportunities to secure a file of a passphrase with the iSeries object authorities. For more information on INLIST see Appendix C.

Examples of PASSWORD() passphrase inlist file coding:

- **PASSWORD('*INLIST;ATEST/PPINLIST(MBR01)')**

  where TYPLISTFL(*DB) and the file is PPINLIST in library ATEST for file member MBR01

- **PASSWORD('*INLIST;*DB;ATEST/PPINLIST(MBR01)')**

  demonstrates overriding the TYPELISTFL

- **PASSWORD('*INLIST;/myroot/PKINLIST/PP01.inl')**

  where TYPLISTFL(*IFS) and the path to the inlist file is '/myroot/PKINLIST/' with stream file name of 'PP01.inl'

- **PASSWORD('*inlist;*IFS;/myroot/PKINLIST/PP01.inl'))**

  demonstrates overriding the TYPELISTFL

The passphrase inlist file must contain "PASSPHRASE={" and be terminated by the "}" character. The passphrase used will be all of the bytes that exist between the {} not including the { or the }. Null bytes and end-of-record bytes are ignored. Therefore the passphrase structure should be only on one record of an inlist file.

Example of contents of an inlist file:

```
PASSPHRASE={x12345678901234x}
```

The passphrase used will be x12345678901234x in EBCDIC. This will then be translated to ASCII using the FTRAN parameter.

Care should be taken that the file is EBCDIC and has a correct code page. It will use all bytes of data between the {}, even non-displayable bytes.

After a passphrase inlist file is set up, it should be tested with both PKZIP and the PKUNZIP command.

PKOVRTAPF

```
New Archive Tape Overrides:
    Tape Device         . . . . .   *TAPF        Tape Devive
    Tape File Label     . . . . .   *TAPF        Tape Header
    Tape Sequence Nbr   . . . . .   *TAPF        1-16777216, *TAPF, *END
    File expiration date . . . .    *TAPF        Date, *NONE, PERM, *TAPF
    End Of Tape Option . . . . .    *TAPF        *TAPF, *REWIND, *UNLOAD...
```

Or

PKOVRTAPF(*TAPF 'ARCHIVE_TEST01' 1 '11/08/2005' *TAPF)
PKOVRTAPF(*TAPF 'ARCHIVE_TEST02' *END *TAPF *LEAVE)
PKOVRTAPF(TAP02 'ARCHIVE_TEST03' *END *PERM *LEAVE)

This parameter defines the override options for the tape device file specified in the archive parameter. These options are active only if TYPARCHFL(*TAP) is set to write the archive directly to a tape. When *TAPF is specified, the value from the current tape device file is used. For more information on these options, refer to the CRTTAPF command.

Tape Device (Tape Device |*TAPF)

Overrides the DEV() parameter of the tape device file. Specifies the name of a tape device used with this device file to perform output data operations.

Tape File Label (label string |*TAPF)

Overrides the LABEL() parameter of the tape device file. Specifies the data file identifier of the data file that is being processed by this tape device file. The data file identifier is defined for standard-labeled tapes and is stored in the header label immediately before the data file that the header describes. For more details on the specification for this 17-byte option, refer to the CRTTAPF command and tape label processing.

Tape Sequence Number (*END |sequence nbr |*TAPF)

Overrides the SEQNBR() parameter of the tape device file.

This specifies the file sequence number of the data file on the tape being processed. For standard-labeled tapes, this four-position file sequence number is read from the first header label of the data file.

- *END - The file is written to the end of the tape or the after the last file.
- Sequence number - Specifies the file sequence number of the file being processed on this tape.

Tape File Expiration Date (Date |*NONE |*PERM |*TAPF)

Overrides the EXPDATE() parameter of the tape device file. This option specifies the expiration date of the data file used by this device file. The data file is protected and cannot be written over until the specified expiration date has past.

- *NONE - No expiration date for the data file is specified. The file is not protected.
- *PERM - The data file is protected permanently. The date written on the tape is 999999.
- Date - Specify the date on which, and beyond which, the data file is no longer protected. The date format should be mm/dd/yyyy such as ‘11/08/2005’.

End of Tape Option (*TAPF |*REWIND |*UNLOAD |*LEAVE)

Overrides the ENDOPT() parameter of the tape device file. This option specifies the operation that is automatically performed on the tape volume after the operation ends. If more than one volume is included, this parameter applies only to the last tape volume used; all other tape volumes are rewound and unloaded when the end of the tape is reached.

- ***REWIND** - The tape is rewound, but not unloaded.
- ***UNLOAD** – The tape is automatically rewound and unloaded after the operation ends.
- ***LEAVE** – The tape does not rewind or unload after the operation ends. It remains at the current position in the tape drive.

SELFXTRACT

SELFXTRACT (*MAINTAIN| WINDOWS| UNIX| LINUX| *REMOVE)

This licensed feature specifies the action to take concerning self extracting archives. The actions are to maintain the current archive as is, create the new archive with a self extracting preamble, or to remove the self extracting preamble if one exists in the archive.

The self extracting programs are held as binary entities in the ***PKZIP***^{i} library in the file PKZIPSFX. The appropriate member is loaded and the executable data copied to the beginning of the archive as a preamble when requested.

The resulting archive can still be processed by ***PKZIP***^{i} as a normal ZIP archive.

The allowable values are:

MAINTAIN	If the current archive contains a self extracting preamble, it will be maintained at the beginning of the updated archive.
WINDOWS	The Windows version of the self extracting preamble will be installed to archive. (Microsoft Windows 95 and later)
AIX	The AIX version of the self extracting preamble will be installed to archive. (IBM AIX Version 4.0 and later)
HP_UNIX	The HP UNIX version of the self extracting preamble will be installed to archive. (HP/UX Version 9.0 and later)
SUN_UNIX	The Sun UNIX version of the self extracting preamble will be installed to archive. (Sun Solaris 2.3 (SunOS 53) and later)
LINUXINTEL	The Linux version of the self extract preamble (if available) will be installed to archive. (LINUX Kernel 2.x for Intel Note:libc-5 must be installed on the target system.)
***REMOVE**	If the current archive contains a self extracting preamble, it will be removed.

SFUSER

SFUSER (*CURRENT|*ALL |User Name List)

Specifies the user names that created spool files that will be selected. This value is ignored if SFJOBNAM is coded.

The allowable values are:

***CURRENT**	Only files created by the user running this command are selected.
***ALL**	Files created by all users are selected.

User Name	Specify up to 10 user names. Only files created by those users are selected.

SFQUEUE

SFQUEUE (*ALL |Name|*LIBS)

Specifies the output queue that will be searched for the spool file selections. If no OUTQ library is specified, it will default to *LIBL.

The allowable values are:

***ALL**	Files on any device-created or user-created output queue are selected.
OUTQ	The OUTQ that will be searched.
OUTQ Library	The library where the OUTQ resides. Defaults to *LIBL.
***LIBS**	*LIBS will search all OUTQ that exist in the specified OUTQ Library. If *LIBS is selected then the library cannot be blank, nor contain *LIBL nor *CURRENT.

SFFORM

SFFORM (*ALL | *STD| Form Type)

Specifies the spool file form type that is on the spool files that will be selected.

The allowable values are:

***ALL**	Files for all form types are selected.
***STD**	Only files that specify the standard form type are selected.
Form Type	Only spool files with this specific form type will be selected.

SFUSRDTA

SFUSRDTA (*ALL| User Data)

The user data tag associated with the spool file to select.

The allowable values are:

***ALL**	Files with any user data tag specified are selected.
User Data	Only spool files with this specific user data tag will be selected.

SFSTATUS

SFSTATUS (*ALL |*READY|*HELD|*CLOSED|*SAVED|*PENDING|*DEFERRED)

Specifies the statuses of the spool files to be selected. Up to four statuses can be selected for one run.

The allowable values are:

***ALL**	All spool file status will be considered for selection.
***READY**	Only spool files with a status of *READY will be selected.
***HELD**	Only spool files with a status of *HELD will be selected.
***CLOSED**	Only spool files with a status of *CLOSED will be selected.
***SAVED**	Only spool files with a status of *SAVED will be selected.
***PENDING**	Only spool files with a status of *PENDING will be selected.
***DEFERRED**	Only spool files with a status of *DEFERRED will be selected.

SFJOBNAM

SFJOBNAM(Blank|*|Spool File Jobname/User/Job Number)

Specifies the job name, user name, and job number that will be used to select spool files. If anything other than blanks is in SFJOBNAM parameter, it will be used as the primary selection criteria. If any of the three fields (job name, user name, and job number) are specified, then all three fields must be entered and be valid.

The allowable values are:

Blank	This is the default selection. This will cause all other selection criteria to be used for spool files.
*****	The * will cause the current job-name/user-name/job number to be used to select spool files.
Job-name	Specify the name of the job to be selected. If no job qualifier is given, all of the jobs currently in the system are searched for the simple job name.
User-Name	Specify the name that identifies the user profile under which the job is run.
Job-Number	Specify the job number assigned by the system.

SFTARGET

SFTARGET (*SPLF|*TEXT|*PDF|*TEXT1|*TEXT2)

Specifies the format of the file that will be stored in the archive.

The allowable values are:

***SPLF** This is the default selection. This will compress the spool file in a spool file format with all of the spool file attributes. This format is only valid on an AS/400. If the archive is extracted, it will take on the latest spool file settings, such as, job name, user, job number, spool file number, etc. The suffix for this selection is SPLF. Parameter SFTGFILE is required to be *GEN1 for SFTARGET(*SPLF).

***TEXT** The spool file will be saved in the archived as an ASCII text document. The suffix for this selection is .TXT. Each new page will have a form feed control character.

***TEXT1** The spool file will be saved in the archived as an ASCII text document. The suffix for this selection is .TXT. Each new page will have a carriage control and line feed control characters.

***TEXT2** The spool file will be saved in the archived as an ASCII text document. The suffix for this selection is .TXT. Each page will have a carriage control and line feed control characters for blanks lines to fill out a page with the number lines required by the spool file attribute.

***TEXTFC** The spool file will be saved in the archive as an ASCII Text document. The suffix for this selection is .TXT. Each new page will have a form feed control character.

***PDF** The spool file will be saved in the archived as a PDF text document. The suffix for this selection is .PDF. The size will be adjusted based upon the width and length of the spool file.

***PDFLETTER** The spool file will be saved in the archived as a PDF text document. The suffix for this selection is .PDF. The size will be adjusted based upon the width and length of the spool file.

***PDFLEGAL** The spool file will be saved in the archived as a PDF text document. The suffix for this selection is .PDF. The size will be adjusted based upon the width and length of the spool file.

SFTGFILE

SFTGFILE (|*GEN1|*GEN2|*GEN1P|File Name)

Specifies the how the file name will be stored in the archive.

The allowable values are:

***GEN1** GEN1 is the default selection. This generates a very specific name using most of the spool file name attributes to form the file name so that it will not be

duplicated. *GEN1 is required if SFTRAGET is *SPLF. The name will be built as follows:

"Job-Name/User-Name/#Job-Number/Spool-File-Name/Fspool-File-Number.Suffix"

"MYJOB/BILLS#152681/INVOICE/F0021.SPLF"

The suffix is dependent on the SFTARGET setting.

***GEN1P** *****GEN1P generates the same file name as *GEN1 except instead of a '/' separator, *GEN1P will use a '.' as name separator.

***GEN2** *****GEN2 uses the spool file name and appends the spool file number followed by the suffix that is depended on the SFTARGET setting. Caution should be taken in that a duplicate file name in the archive could be created. An example of GEN2 is a spool file INVOICE with spool file number of 21 that will be converted to a text file will generate a file name of INVOICE21.TXT.

File Name This parameter should only be used when selecting one specific spool file where you want a specific file name.

SPLFILE

SPLFILE (*ALL| Spool File Name)

Specifies the spool file name that will be selected. This parameter is used along with all the other spool file selection parameters to determine the spool files to select.

The allowable values are:

***ALL** This is the default setting. *ALL indicates that spool file name is not important.

Spool File Name A specific spool file name that will be searched for and selected.

SPLNBR

SPLFILE (*ALL|*LAST| Spool File Number)

Specifies the number of the spool file from the job whose data records are to be selected. If *ALL is coded then all file numbers are considered. This parameter is only valid when the SFJOBNAM parameter or SPLFILE is used. This parameter is used along with all the other spool file selection parameters to determine the spool files to select.

The allowable values are:

***ALL** This is the default setting. *ALL indicates that spool file number is not important.

***LAST** The spooled file with the highest number is used.

Spool File Number A number 1-9999 to specify the number of the spooled file whose data records are to be selected.

SIGNERS

Requires SecureZIP

```
Signing Certificates       :
    File/Archive   . . . . . .   *FILE          *FILE, *ARCHIVE, *ALL
    LookUp Type  . . . . . . .   *DB           *DB, *FILE, *MBRSET, *INLIST
    Signer . . . . . . . . . .   _____________________________________________
    Passphrase (If Private)  . . _____________________________________
    Required . . . . . . . . .   *RQD          *RQD, *OPT
```

Or

```
SIGNERS((*FILE *MBRSET
  'pkwareCertAdmin04.pfx'  (passphrase) *RQD))
SIGNERS((*FILE *FILE
  '/yourpath/PKWARE/Cstores/private/pkwareCertAdmin04.pfx'  (passphrase) *RQD))
SIGNERS((*FILE *FILE
  '/yourpath/PKWARE/Cstores/private/pkwareCertAdmin04.pfx'  ('mypassphrase') *RQD))
SIGNERS((*FILE *DB
  'EM=bill.somebody@pkware.com'  (passphrase) *RQD))
SIGNERS((*FILE *INLIST 'ATEST/INLIST(ENGNEER1)' *N)
```

This parameter identifies the public key certificate with private key that is to be used to digitally sign files to be added to the archive and/or the archive directory. Multiple signing certificates may be applied to the files but only one signer is allowed to sign the archive directory. Signing an archive by signing its central directory enables people who receive the archive to confirm that the archive as a whole is not changed. By contrast, signing only individual files in an archive enables people to confirm that the particular signed files are unchanged but leaves open the possibility that the archive has had files added or removed.

There are five options for SIGNERS.

Signing Type File/Archive (*FILE |*ARCHIVE |*ALL)

The File/Archive selection determines whether the files, archive or both are to be signed during the ZIP run. Only one signer can be specified for an archive. If the lookup type is *INLIST, then this option will be ignored and will pickup from the records in the inlist file.

- *FILE – All new files being compressed in the run will be signed by this private key and a signature entry will be added to the archive.
- *ARCHIVE – The archive directory will be signed by this private key and a signature entry will be added to the archive.
- *ALL – Both *FILE and *ARCHIVE for the signer will be used.

Lookup Type (*DB |*FILE |*MBRSET |*INLIST)

The lookup type would be the type of signer search that will be used for the signer string to lookup the private key.

- *DB - The signer string is defined to search using the Certificate Locator Database to access the digital certificate and private key.
- *FILE - The signer string is defined to read a specific file in a specific path in the IFS in order to access the digital certificate and private key.
- *MBRSET - The signer string is defined to read this specific file from the enterprise private certificate store to access the digital certificate and private key.
- *INLIST- The signer string defines a specific file that will contain one to many signers. The TYPLISTFL parameter must specify the file type for the inlist.

Signer (The signer string name)

The signer string format depends on what was specified for the lookup type.

- If lookup type is *DB, the signer string will either be an email address or the common name of the certificate. This depends on the configuration setting in PKCFGSEC parameter CERTDB. To override the default selection mode, you can prefix the string with EM= for email, or CN= for the common name.

  For example:

  SIGNERS((*FILE *DB ‘bill.somebody@pkware.com' (passphrase) *RQD)
  (*ARCHIVE *DB ‘CN=bill somebody' (passphrase) *RQD)
  (FILE *DB ‘EM=bill.somebody@pkware.com' (passphrase) *OPT))

- If lookup type is *FILE, the signer string is defined to read a specific file in a specific path of the IFS. This file should be public key X.509 with the private key X.509 certificate file.

  For example:

  SIGNERS((*ARCHIVE *FILE '/yourpath/PKWARE/Cstores/private/pkwareCertAdmin04.pfx' (passphrase) *RQD))

  The digital certificate file with the private key ‘pkwareCertAdmin04.pfx’ will be in the full path '/yourpath/PKWARE/Cstores/private’.

- If lookup type is *MBRSET, the signer string is defined to read a specific file from the public certificate store and/or the private certificate store of the IFS. This file should be public key X.509 with the private key X.509 certificate file.

  For example:

  SIGNERS((*ALL *MBRSET 'pkwareCertAdmin04.pfx' (passphrase) *RQD))

  The digital certificate file ‘pkwareCertAdmin04.pfx’ will be in the full path of the private certificate store defined in the enterprise security configuration private store (parameter CSPRV). Since the passphrase was included, the file will be searched for in the enterprise security configuration private store (parameter CSPRIV).

- If lookup type is *INLIST the signer string defines a full file name of an input list file that contains records of SIGNERS shortcut parameters. The type of file will exist in the QSYS library file system if TYPLISTFL(*DB) is set and will be a

path file name in the IFS if TYPLISTFL(*IFS) is set. The format of the SIGNERS shortcut parameters are defined below in the *INLIST usage section.

Passphrase

This designates the passphrase that is *required* for a private key (PKCS#12 file). When a value is specified, the target must be an X.509 PKCS#12 private key certificate.

The PASSWORD value may contain blanks and is delimited by the closing right parenthesis ")" of the signing command.

Required (*RQD|*OPT|*SAME)

If *RQD then this signer MUST be found during the selection and the certificate MUST be a valid certificate with a private key or the run will fail.

Usage Notes:

A NULL file (binary file having zero bytes of data) will be signed. However, note that the digital signature is based on a fixed hash value.

The entire data stream of each file is run through the hash algorithm before compression or encryption. However, file text data is translated before hashing so that the receiving system is able to hash the identical stream after decryption/decompression.

The processor requirement for a file signature is directly related to the size of the file(s) being signed and/or authenticated (see SIGN_HASHALG). Therefore, when processing costs are a consideration, the decision whether to use SIGNERS to sign large files should be based on the business case. Sometimes signers for the archive may be more appropriate. (The directory size is proportional to the number of files in the archive, not the physical size of the file data.)

A separate signing operation is performed for each supplied certificate, for each file. Processor and elapsed time will be impacted in proportion to the number of signatories and files selected.

The number of file signatures that can be held for each file is constrained by a number of factors. These include EXTRAFLD(*YES) and DBSERVICE(*NO), the size of the signatures generated (based on the size of the certificate information), the number of certificates in the authenticating certificate authority chain, the number of different certificate authorities used in association with the signing certificates, if FNE(*YES) is specified, and the number of recipients for certificate-based encryption of files. For planning purposes, typical ZIP operations will support up to 10 file signatories as a rule, although more or fewer may be achieved in practice.

It is important that the passphrase is entered in the correct case. Any variation in case or misspelling will result in a public key certificate access attempt (which will fail for a private key PKCS#12 certificate). Please note that passphrases will be masked out in all output displays.

A local certificate store configuration is required to complete the processing of this command. Even when a direct FILE specification is made to locate the private key certificate, the CS and ROOT certificate store components must be accessible to complete the certificate signing chain within the archive. This information is required

to complete authentication processing on the target system when the local certificate store on that system does not contain the certificate authority chain required to validate TRUST (see PKCFGSEC).

Processing will be terminated if none of the requested certificates can be accessed, regardless of the "R" required flag. If multiple requests are made and at least one signature is found, processing will continue normally.

Signed files are tolerated by prior releases of ***PKZIP for i5/OS*** and ***SecureZIP for i5/OS*** but are not processed for authentication.

For inlist that contains a passphrase to open a private certificate, make sure that the security is sufficient to only allow the owner of the certificate to have read access. Otherwise this would leave a security hole where others could browse the passphrase.

***INLIST Usage:**

If *INLIST is defined on the SIGNERS parameter, then the signer filed will be a file that SecureZIP will read to include the signer. The format is very similar to the SIGNERS parameter described above except each line signer starts with "{SIGNERS=" and is terminated by the "}" character with the semi-colon ";" as a separator for each entry.

{SIGNERS=Signing Type, Lookup Type; Signer; Passphrase; Required}

Signing Type	See Signing Type in SIGNERS
Lookup Type	See Lookup Type in SIGNERS excluding the INLIST
Signer	See Signer in SIGNERS.
Passphrase	See Passphrase in SIGNERS.
Required	See Required in SIGNERS, but use RDQ for *RQD and OPT for *OPT.

Examples:

Sample 1: tstsign_db1.inlist.

```
{SIGNERS=File;DB;EM=PKTESTDB4@nowhere.com;PKWARE;RQD}
```

Sample 2: tstsign_mb2.inlist.

```
{SIGNERS=ARCHIVE;MBRSET;pktestdb3.pfx;PKWARE;RQD}
```

SIGNPOL

Requires SecureZIP

```
Signing Filters:
    Validate Level     . . . . .   *SYSTEM       *VALIDATE, *WARN, *NONE...
       Filters . . . . . . . . .   *SYSTEM       *SYSTEM, *ALL, *NONE...
               + for more values
```

Or

SIGNPOL(*WARN (*SYSTEM))
SIGNPOL(*WARN (*ALL *NOTTRUSTED))
SIGNPOL(*SYSTEM (*ALL *NOTEXPIRED))

This parameter defines the processing options and filters that should take place if the SIGNERS parameter is used to define the file or archive signing certificates.

Validate Level (*VALIDATE |*WARN |*SYSTEM)

The validate level specifies the type of signing processing that should take place if the signer requests encounter an error. If *SYSTEM is specified, the enterprise setting from PKCFSEC is used. If the enterprise setting is defined as Lockdown, then this parameter cannot be revised and a warning will be issued if a change is detected.

- *SYSTEM - Indicates the authentication processing that is set in the environmental setting will be used.
- *VALIDATE - Indicates that when authentication takes place and a failure occurs based on the filters, the run will be considered a failure, and the message issued at the end will indicate one or more errors during the run.
- *WARN - Indicates that when authentication takes place and a failure occurs, the failure is only considered a warning. The messages at the end of the run will not consider any failed filters for signer certificates as errors.

Filters (*SYSTEM |*ALL |*NONE |*TRUSTED |*EXPIRED |*REVOKED |*NOTTRUSTED |*NOTEXPIRED |*NOTREVOKED)

The signing filter policies settings are defined in the enterprise security file supplied by the SecureZIP administrator (see PKCFGSEC). These global policy settings can be revised with sub-parameter values, but if the enterprise setting is defined as lockdown, this parameter cannot be revised and a warning will be issued if a change is detected. The variables are cumulative from the global setting. The setting of these filters defines what certificates are acceptable for signing.

- ***SYSTEM** - All filter policies are from the global settings.
- ***ALL** - This sub-parameter activates all levels of authentication. If followed by negating sub-levels, then all but those negating levels are activated. For example: *ALL, NOTEXPIRED means that expired certificates will not cause an authentication error, but TRUST and REVOKE must both be satisfied.
- ***NONE** - Will negate all the policies.
- ***TRUSTED** - Each end-entity certificate used in the signature must be traced back to a trusted root certificate. The CACA and CSROOT stores on the local system performing the authentication check will be accessed to determine if the entire certificate chain can be trusted. Although the root (“self-signed”) certificate may be included within the archive, it MUST also exist in the CSROOT store to complete the TRUSTED state.
- ***EXPIRED** - The digital certificates used to originally perform the signing operation contain internal date ranges of validity. The signer operation will fail if any of the certificates in the trust chain are not found to be within their stated data range. Note that an end-entity certificate may have expired at the

time that the archive is being accessed, and NOTEXPIRED may be used to continue processing.

- ***REVOKED** - A certificate owner may request that the issuing certificate authority declare a certificate to be revoked and thereby no longer consider that certificate to be valid. The signer operation will fail if any of the certificates in the trust chain are found to have been revoked or if the revocation status could not be determined.
- ***NOTTRUSTED** - Negates the *TRUSTED filter.
- ***NOTEXPIRED** - Negates the *EXPIRED filter.
- ***NOTREVOKED** - Negates the *REVOKED filter.

STOREPATH

STOREPATH(*NO|*YES)

Specifies whether to store the full path and file name in the archive, or to just save the file name. If the file is an IFS file type, the path is all directories, from the current directory, to the directory of the file. In the library system, the path is the library and the file name. The member name is considered to be the archive name.

The allowable values are:

***YES**	Store all paths and the filename in the PKZIP archive.
***NO**	Store only the filename in the PKZIP archive.

TMPPATH

TMPPATH(*CURRENT| *pathname*)

Specifies a directory or library/file in which to build the temporary archive file. While PKZIP is compressing data into an archive, a temporary archive file name is used. The temporary file name is a 10-character name with a prefix of “PZ” followed by a time stamp (PZttttttttt). If this option is *CURRENT, the temporary file is built in the same directory (for library file systems it is same library/file with temporary member) in which the new archive will be stored and is then renamed at the end of the run to the archive name. If an override path is specified, the temporary archive file is built into that specified path, and the file is then copied to its final archive path at the end of the run. The temporary file name and path type will be the same as specified for ARCHIVE. See parameter TYPARCHFL for file system type information. Special libraries (such as QTEMP) are used frequently.

***CURRENT**	Specifies that the current archive path will be used (see ARCHIVE) to build the temporary archive file PZxxxxxxxx.
pathname	Specifies a path name (if using IFS such as /PKZIP/tempdir) or a library/file (if using the library system).

NOTE 1: When using the QSYS library file system and specifying "qtemp" as the TMPPATH, a dynamic file name and member name is created in the library qtemp. At the end of the run, the file and member are removed. If any other combination of names is used, then a dynamic member name is created and only the member is removed.

NOTE 2: When using the QSYS library file system and specifying a TMPPATH, there may be a slight performance degradation because the archive file will have to be copied from one library/file to another library/file. Otherwise, if *CURRENT is used, the file member name will only be renamed.

TRAN

TRAN(*ISO88591 |*INTERNAL| *Member Name*)

Specifies the translation table for use with translating "data" from the iSeries EBCDIC character set to the character set used in the archive file (normally the ASCII character set). A default internal table is predefined (see Appendix D).

***ISO88591** The predefined internal table for translation. This table provides translation that is consistent with the ISO 8859-1 definitions. This table uses the EBCDIC code page 037 and the ASCII code page 819 for translation.

***INTERNAL** To provide some compatibility to pre V8 version, *INTERNAL will use the internal tables that were the default in V5 PKZIP.

Member Name Specifies the member name in the file PKZTABLES that will be parsed and used to translate data files to the archive character set. The member should have the exact format of member ISO9959_1 in file PKZTABLES (see Appendix D for information on defining translation tables).

TYPARCHFL

```
Archive File:
        Type . . . . . .   *DB          *DB, *IFS, *TAP, *XDB
        Check ZIP64  . . . . . .   *NONE        *NONE, *WARN, *FAIL
```

Or

TYPARCHFL (*IFS)
TYPARCHFL (*DB *WARN)
TYPARCHFL (*IFS *FAIL)
TYPARCHFL (*TAP)

This parameter specifies the file system to create the archive and the archive constraints.

Archive Type (*DB |*IFS |*TAP |*XDB)

Specifies the type of file system in which the archive file will exist (see parameters ARCHIVE and TMPPATH for additional information).

***DB**	Archive files are to be in the QSYS library file system. Even though *DB is working with archive files that are in the QSYS library file system, the IFS is utilized for performance and for large file support (ZIP64). To provide an option for archive file creation utilizing exclusively the QSYS library system, use TYPARCHFL(*XDB), which supports OS400 features such as Adopt Authority.
***IFS**	Archive files are to be in the integrated file system (IFS).
***TAP**	Archive file will be written directly to tape. The ARCHIVE parameter MUST be a tape device file or have attribute *TAPF. The tape device file PKTAPEO1 is distributed with PKZIP.
***XDB**	The archive files are to be in the QSYS library file system and will exclusively use the QSYS library file system during processing. This will force the Check ZIP64(*FAIL). This option will not support Large File Support or ZIP64.

Check ZIP64 (*NONE |*WARN |*FAIL)

Specify the severity of message and return code when creating or updating an archive and ZIP64 processing is required.

***NONE**	No action or message when ZIP64 constraint exceeded.
***WARN**	Warning message AQZ0613 will be issued but processing will continue.
***FAIL**	Failure message AQZ0614 will be issued and process will cease without building a new archive.

This feature may be of value when creating archives intended for distribution to systems that may not be able to handle the ZIP64 processing attributes. This may be due to the UNZIP software being used on the target system or the file system for the related OS. (For example, some UNIX or Windows FAT file systems cannot handle file sizes greater than 4 gigabytes).

Triggers for this option include:

- More than 65,535 files are being placed into the archive
- One or more source files are greater than 4 gigabytes in size
- The amount of data written to the archive exceeds 4 gigabytes

TYPFL2ZP

TYPFL2ZP(*DB|*IFS)

Specifies the type of file system that contains files to be zipped. Reflected for files in parameters FILES and EXCLUDE.

***DB**	Files to be zipped are in the QSYS library file system.
***IFS**	Files to be zipped are in the IFS (integrated files system) - Case sensitive selection.
***IFS2**	Files to be zipped are in the IFS (integrated files system) – Non-case-sensitive selection.
***DBA**	Files to be compressed are database files in the QSYS library file system with database mode "DBSERVICE(*YES)", and the records are to be processed in arrival sequence. This is only pertinent for database files containing keys and when it is important to retain the arrival sequence of the data.
***SPL**	Files to be zipped are spool files.

TYPLISTFL

TYPLISTFL(*DB|*IFS)

Specifies the “type of files system” that will be used for the input list file and/or the output list file of selected items.

To use input list files, see parameters INCLFILE (file section list) or EXCLFILE (file exclude list). To create an output list file of the selected file items, see parameter CRTLIST.

***DB**	Files are in the QSYS library file system.
***IFS**	Files are in the IFS(integrated files system).

VERBOSE

VERBOSE(*NORMAL|*NONE| *ALL|*MAX)

Specifies how the detail will be displayed during a PKZIP run.

The allowable values are:

***NORMAL**	Displays most informative message to show PKZIP is processing.
***NONE**	Displays only major exception information.
***ALL**	Displays all messages.
***MAX**	Used only for debugging purposes.

VPASSWORD

VPASSWORD(Archive Verify Passphrase)

Specifies a verification passphrase against the entered passphrase since the PASSWORD is not visible. This parameter is required for all encryption methods except ZIPSTD. VPASSWORD follows all the rules of PASSWORD and must match exactly to the archive passphrase entered in PASSWORD parameter or the run will be terminated. If the PASSWORD() parameter contains an inlist file, the VPASSWORD() parameter is ignored.

8 PKUNZIP Command

PKUNZIP Command Summary with Parameter Keyword Format

If the OS/400 command prompt screen is to be used, the command format is simply: PKUNZIP.

The command prompt screen is displayed when ENTER or PF4 is pressed. The parameter keywords are displayed on this screen together with the available keyword options. The required options can be selected before PF4 is pressed to accept the selections. If the command and parameter keywords are entered together on the command line, the required format is:

PKUNZIP keyword1(option) keyword2(option) . . . keyword*n*(option)

Keywords are delimited by spaces. The keyword “ARCHIVE” is the only positional keyword where the keyword itself is not required. Whenever the word “path” is used, its meaning depends on the file system that is being used. If IFS is used, path refers to the openness true path type. If the library systems or *DB is used, path means library/file, and then the file name refers to the member name.

```
TYPE(       *VIEW                 )
              (*EXTRACT}
              {*NEWER}
              {*TEST}

ARCHIVE(    Archive Zip File name with path      )

AUTHCHK(    Authenticators        )             (SecureZIP Only)
            Authenticate Type           {*FILE}
                                        {*ARCHIVE}
                                        {*ALL}
            Lookup Type                 {*DB }
                                        {*LDAP}
                                        {*FILE}
                                        {*MBRSET}
                                        {*INLIST}
                                        {*SPONSOR} (Read mode Only)
            Recipient                   {Recipient String}
            Passphrase (if Private)     {Certificate passphrase}
            Required                    {*RQD }
                                        {*OPT}

AUTHPOL   (  Authenticate Filters: )            (SecureZIP Only)
Validate Level        {*SYSTEM }
                      {*WARN }
                      {*VALIDATE}
                      {*REQUIRED}
Validate Type         {*NONE }
                      {*ALL }
                      {*ARCHIVE}
                      {*FILE}
Filters               {*SYSTEM }
                      {*ALL}
                      {*NONE}
                      {*TAMPER}
                      {*TRUSTED}
                      {*EXPIRED}
                      {*REVOKED}
                      {*NOTAMPER}
                      {*NOTTRUSTED}
                      {*NOTEXPIRED}
                      {*NOTREVOKED}

CRTLIST(    {*NONE}               )
              path/filename

CVTDATA(    External Pgm Conversion Extended Data)


CVTFLAG(    {*NONE}               )
              External Pgm Conversion Flags

CVTTYPE(    {*NONE}               )
              {*DROP}
              {*SUFFIX}

DFTDBRECLN({132}                  )
              {decimal number}

DROPPATH(   {*NONE}               )
              {*ALL}
              {*LIB}

ENTPREC(    Decryption Recipients )            (SecureZIP Only)
            Lookup Type                 {*DB }
                                        {*FILE}
```

```
                                    {*MBRSET}
                                    {*INLIST}
           Recipient                {Recipient String}
           Passphrase (if Private)   {Certificate passphrase}
           Required                 {*RQD }
                                    {*OPT}
EXCLFILE(  {*NONE}              )
               path/filename

EXCLUDE(   file_specification1,        )
               file_specification2,
               file_specificationn

EXDIR(     {*CURRENT}           )
               path

FILES(     file_specification1,        )
               file_specification2,
               file_specificationn

FILETYPE(  {*TEXT}              )
               {*BINARY}
               {*EBCDIC}
               {*DETECT}

FTRAN(       {*ISO88591}        )
             {*INTERNAL}
              Member Name

IFSCDEPAGE({*NO}                )
               Code-page

INCLFILE(  {*NONE}              )
               path/filename

MSGTYPE(   Outlist Details:     )
  Type         {*BOTH}
               {*PRINT}
               {*SEND}

  Licence Info {*NORMAL}
               {*SHORT}
               {*NONE}

OVERWRITE( {*NO}                )
               {*YES}
               {*PROMPT}

PASSWORD(  Archive Passphrase          )

RSTIPSRA ( Restore Command for iPSRA Files  )

SFQUEUE (  {*DFT}               )
               {Library/Outq }SPLUSRID (

SPLUSRID   {*DFT}               )
               {User ID }

TRAN(        {*ISO88591}        )
             {*INTERNAL}
              Member Name

TYPARCHFL(     {*DB}                  )
               {*IFS}
               {*XDB}
```

```
TYPFL2ZP(       {*DB}                    )
                {*IFS}

TYPLISTFL(      {*DB}                    )
                {*IFS}

VERBOSE(    {*NORMAL}              )
                {*NONE}
                {*ALL}
                {*MAX}

VIEWOPT(    {*NORMAL}              )
                {*DETAIL}
                {*BRIEF}
                {*COMMENT }
                {*FNE}
                {*FNEALL}


VIEWSORT(   {*ASIS}                )
                {*DATE}
                {*DATER}
                {*NAME}
                {*NAMER}
                {*PERCENT}
                {*PERCENTR}
                {*SIZE}
                {*SIZER}
```

PKUNZIP Command Keyword Details

TYPE

TYPE(*EXTRACT|*NEWER|*TEST |*VIEW)

The TYPE keyword specifies the type of action PKUNZIP should perform on the ZIP archive.

The possible actions are:

***VIEW**	Display output information about all files or selected files contained in an archive. This option is performed using PKUNZIP. The sequence (see *VIEWSORT) and type of list (*VIEWOPT) determines what information is displayed.
***EXTRACT**	Extracts files from the archive (please refer to the DROPPATH, CVTTYPE, TO, and EXDIR parameters for controlling the conversion of file names extracted from the archive).
***NEWER**	Extracts files in the archive that have a more recent date and time than the corresponding file on disk. If the files do not exist on disk, they will be extracted as newer. All other files will be skipped.
***TEST**	Tests the integrity of files in the archive by extracting files without writing the data. As each file is extracted, a CRC is calculated. At the end of the file the calculated CRC is compared against the stored CRC in the archive file header to confirm that the data has not been corrupted.

ARCHIVE

ARCHIVE(Archive Zip File name with path)

Specifies the path/file name or the library/file name of the PKUNZIP archive to be processed.

This is a required parameter.

The format depends on whether you will be using the archive file in the library file system or the IFS.

See parameter TYPARCHFL for file system type information.

Library File System: Format is library/file(member). If member is omitted, it will use the file name for the member.

Integrated File System (IFS): Open system path followed by the archive file name. The path and file name can up to 256 characters and may contain embedded spaces.

AUTHCHK

Requires SecureZIP

```
Authenticator Certificates:
    File/Archive   . . . . . .   *FILE          *FILE, *ARCHIVE, *ALL
    LookUp Type  . . . . . . .   *DB            *DB, *LDAP, *FILE, *MBRSET...
    Authenticator  . . . . . .   _____________________________
    Passphrase (If Private)  . . _____________________________
    Required . . . . . . . . .   *RQD           *RQD, *OPT
               + for more values  _
```

Or

AUTHCHK((*FILE *MBRSET
'pkwareCertAdmin04.pfx' (passphrase) *RQD))
AUTHCHK((*ALL *FILE
'/yourpath/PKWARE/Cstores/public/pkwareCertAdmin04.cer' () *RQD))
AUTHCHK((*ARCHIVE *FILE
'/yourpath/PKWARE/Cstores/public/pkwareCertAdmin04.cer' () *RQD))
AUTHCHK((*FILE *DB
'EM=bill.somebody@pkware.com' () *OPT))
AUTHCHK((*FILE *INLIST 'ATEST/INLIST(ENGNEER1)' *N)

This parameter specifies that digital signature authentication processing should be performed for specific signers. Separate authentication processing may be specified for either the archive central directory or files by using multiple commands. Optionally, specific signers may be specified to authenticate against. This parameter is used in conjunction with the AUTHPOL parameters and its settings.

It is possible that more than one certificate may be returned for a single common name or email search. As a result, each one will be added to the list of validating sources.

When no specific certificates are requested, any signatories found in the archive are validated in accordance with the systems or current AUTHPOL Filters policy settings.

There are five options for AUTHCHK.

Authenticator Type File/Archive (*FILE |*ARCHIVE |*ALL)

This designates the type of authentication that is to be performed. Either ARCHIVE, FILE or ALL may be specified on each item, but by using ALL or archive with the *RQD option will result in error since the archive can only have one signatory. If the lookup type is *INLIST, then this option will be ignored and will pickup from the records in the inlist file.

- *FILE – The signed files will be authenticated with this authenticator.
- *ARCHIVE - The archive directory will be authenticated with this authenticator.

- *ALL – Both the signed files and the archive directory will be authenticated with this authenticator.

Lookup Type (*DB |*FILE |*LDAP |*MBRSET |*INLIST |*SPONSOR)

The lookup type would be the type of authenticator search to be used for the authenticator string to look up the public key.

- *DB - The authenticator string is defined to search using the certificate locator database to access the digital certificate.
- *FILE - The authenticator string is defined to read a specific file in a specific path in the IFS in order to access the digital certificate.
- *LDAP - The recipient string is defined to search using the LDAP server to access the digital certificate.
- *MBRSET - The authenticator string is defined to read this specific file from the enterprise public certificate store to access the digital certificate.
- *INLIST- The authenticator string defines a specific file that will contain one to many AUTHCHK. The TYPLISTFL parameter must specify the file type for the inlist.
- *SPONSOR - The authenticator string is the authenticating file for a sponsoring partner. This applies only for SecureZIP Partner Read mode and for *ARCHIVE.

Authenticator (The authenticator string name)

The authenticator string format depends on what was specified for the lookup type.

- If lookup type is *DB, the authenticator string will either be an email address or the common name of the certificate. This depends on the configuration setting in PKCFGSEC parameter CERTDB. To override the default selection mode, you can prefix the string with EM= for email, or CN= for the common name.

  For example:

  AUTHCHK((*FILE *DB ‘bill.somebody@pkware.com' () *RQD)
  (*ARCHIVE *DB ‘CN=bill somebody' () *RQD)
  (FILE *DB ‘EM=bill.somebody@pkware.com' (passphrase) *OPT))

- If lookup type is *FILE, the authenticator string is defined to read a specific file in a specific path of the IFS. This file should be a public key X.509 file or public key X.509 certificate with a private key file.

  For example:

  AUTHCHK((*ARCHIVE *FILE
  '/yourpath/PKWARE/Cstores/public/pkwareCertAdmin04.cer' () *RQD))

  The digital certificate file ‘pkwareCertAdmin04.cer’ will be in the full path '/yourpath/PKWARE/Cstores/public’.

- If type is *LDAP, the authenticator string will either be an email address or the common name of the certificate depending on the search mode configuration setting in PKCFGSEC parameter LDAP. To override the default

selection mode, you can prefix the string with EM= for email address, or CN= for the common name.

For example:

AUTHCHK ((*ARCHIVE *LDAP 'bill.somebody@pkware.com' () *RQD)
(*FILE *LDAP 'CN=bill somebody' () *OPT)
(*FILE *LDAP 'EM=bill.somebody@pkware.com' () *RQD))

- If lookup type is *MBRSET, the authenticator string is defined to read a specific file from the public certificate store and/or the private certificate store of the IFS. This file should be a public key X.509 file or public key X.509 certificate with a private key file.

  For example:

  AUTHCHK((*ALL *MBRSET 'pkwareCertAdmin04.cer' () *RQD))

  The digital certificate file 'pkwareCertAdmin04.cer' will be in the full path of the public certificate store defined in the enterprise security configuration public store (parameter CSPUB). If a passphrase is included, the file is searched for in the enterprise security configuration private store (parameter CSPRIV).

- If lookup type is *INLIST, the authenticator string defines a full file name of an input list file that contains records of AUTHCHK shortcut parameters. The type of file will exist in the QSYS library file system if TYPLISTFL(*DB) is set and will be a path file name in the IFS if TYPLISTFL(*IFS) is set. The format of the AUTHCHK shortcut parameters are defined below in the *INLIST usage section.
- If lookup type is *SPONSOR, the authenticator string is the Sponsor Auth file stored in the '…/Sponsor/Auth' folder. If the authenticator string is all numeric the name will automatically be formatted as A0000000.p7, assuming that the number is the sponsor ID number.

Passphrase

This designates the passphrase that is *required* for a private key certificate with a private key (PKCS#12 file). When a value is specified, the target must be an X.509 PKCS#12 public key certificate with the private key.

The PASSWORD value may contain blanks and is delimited by the closing right parenthesis ")" of the signing command.

Required (*RQD|*OPT|*SAME)

If *RQD, then this authenticator *must* be found during the selection, and the certificate *must* be a valid certificate with a private key, or the ZIP/UNZIP run will fail.

Usage Notes:

Passphrases are masked out in all output displays.

A local certificate store configuration is required to complete the TRUST processing of this command.

Processing is terminated if none of the requested certificates can be accessed, regardless of the “R” required flag. If multiple requests are made and at least one signature is found, processing continues normally.

For inlist that contains a passphrase to open a private certificate, make sure that the security is sufficient to only allow the owner of the certificate to have read access. Otherwise this would leave a security hole where other users could browse the passphrase.

***INLIST Usage:**

If *INLIST is defined on the AUTHCHK parameter, then the authenticator filed will be a file that SecureZIP will read to include the authenticator. The format is very similar to the AUTHCHK parameter described above except that each line authenticator starts with “{AUTHCHK=” and is terminated by the “}” character, with the semi-colon “;” as a separator for each entry.

{AUTHCHK=Authenticator Type, Lookup Type; Authenticator; Passphrase; Required}

Authenticator *Type*	See Authenticator Type in AUTHCHK
Lookup Type	See Lookup Type in AUTHCHK excluding the INLIST
Authenticator	See Authenticator in AUTHCHK.
Passphrase	See Passphrase in AUTHCHK.
Required	See Required in AUTHCHK, but use RDQ for *RQD and OPT for *OPT.

Examples:

Sample 1: tstauth_db1.inlist.

```
{AUTHCHK=FILE;DB;EM=PKTESTDB4@nowhere.com;;RQD}
```

Sample 2: tstauth_mb2.inlist.

```
{AUTHCHK=ARCHIVE;MBRSET;pktestdb3.pfx;PKWARE;RQD}
```

Sample 3: tstauth_mb3.inlist.

```
{AUTHCHK=ALL;MBRSET;pktestdb3.pfx;PKWARE;RQD}
```

AUTHPOL

Requires SecureZIP

```
Authenticate Filters:
    Validate Level     . . . . .   *SYSTEM       *VALIDATE, *WARN, *NONE...
    Validate Type      . . . . .   *ARCHIVE      *ARCHIVE, *NONE
       Filters . . . . . . . . .   *SYSTEM       *SYSTEM, *ALL, *NONE...
               + for more values
```

Or

AUTHPOL(*WARN *ARCHIVE (*SYSTEM))
AUTHPOL(*WARN *FILE (*NOTTRUSTED))
AUTHPOL(*SYSTEM *ALL (*ALL *NOTEXPIRED))

This parameter defines the processing options and filters that should apply if a signed file or signed archive is encountered.

Validate Level (*VALIDATE |*WARN |*REQUIRED |*SYSTEM)

The validate level specifies the type of authentication processing that should take place if a file or archive is encountered. The default is *SYSTEM and, unless it is modified, SecureZIP will use the enterprise setting from PKCFSEC.

- *VALIDATE – Indicates that, when authentication takes place and a failure occurs based on the filters, the run will be considered a failure, and the message issued when the job terminates will indicate one or more errors during the run.
- *WARN - Indicates that when authentication is in place and a failure occurs, the failure is only considered a warning. The messages at the end of the run will not consider any failed authentications as errors.
- *REQUIRED – Indicates that authentication *must* take place and that, if any failure occurs based on the filters, the run will be considered a failure, and the message issued when the job terminates will indicate one or more errors occurred during the run. If the archive or file has not been signed, an error will be issued.
- *SYSTEM – Indicates the authentication processing that is set in the environmental setting will be used.

Validate Type (*ALL |*ARCHIVE |*FILE |*NONE)

The validate type specifies whether the file, archive, all or no authentication will take place if a file or archive has been signed. The default is *NONE, and anything other than *NONE requires the Enhanced Encryption module.

- *ALL - Indicates that authentication will take place for both files and/or the archive has been signed.
- *ARCHIVE - Indicates that only a signed archive will be authenticated.
- *FILE - Indicates that only the signed files will authenticated.
- *NONE - Indicates no authentication will take place even though a file or archive has been signed.

Filters (*SYSTEM |*ALL |*NONE |*TAMPER |*TRUSTED |*EXPIRED |*REVOKED |*NOTAMPER |*NOTTRUSTED |*NOTEXPIRED |*NOTREVOKED)

The authentication filter policies settings are defined in the enterprise security file supplied by the SecureZIP administrator (See PKCFGSEC). These global policy settings can be revised with sub-parameter values. The variables are cumulative from the global setting.

- ***SYSTEM** – All filter policies are from the global settings.

- ***ALL** - This sub-parameter activates all levels of authentication. If followed by negating sub-levels, then all but those negating levels are activated. For example: *ALL NOTEXPIRED means that expired certificates will not cause an authentication error, but TRUST and TAMPERCHECK must both be satisfied.
- ***NONE** – Will negate all the policies.
- ***TAMPER** – This sub-parameter signifies that a verification of the data stream should be done against the digital signature.
- ***TRUSTED** – This sub-parameter signifies that the entire certificate authority chain must be validated. This includes locating the root (self-signed) certificate on the local system.
- ***EXPIRED** – This sub-parameter signifies that certificate date range validation should be performed on the certificates (including the certificate authority chain). Although the term “expired” is used, a certificate that has not yet reached its valid data range specification will fail.
- ***REVOKED** - A certificate owner may request that the issuing certificate authority declare a certificate to be revoked and thereby no longer consider that certificate to be valid. The authentication operation will fail if any of the certificates in the trust chain are found to have been revoked, or if the revocation status could not be determined
- ***NOTAMPER** – Negates the *TAMPER filter.
- ***NOTTRUSTED** – Negates the *TRUSTED filter.
- ***NOTEXPIRED** - Negates the *EXPIRED filter.
- ***NOTREVOKED** – Negates the *REVOKED filter.

CRTLIST

CRTLIST(*NONE| *path/filename*)

Specifies that PKUNZIP will create an output file with a list of entries that will be compressed based upon the selection criteria in the FILES and EXCLUDE parameters. This parameter only works with the TYPE set to *VIEW.

See parameter TYPLISTFL for file system type information.

***NONE**	No list file will be created.
path/filename	Enter the file path and name of the file to create. The layout depends on which file system you want to create the file in.

Library File System:
The format is "library/file(member)".

Integrated File System (IFS):
The format is "path1/path2/../pathn/filename".

CVTDATA

CVTDATA(*External Program Conversion Extended Data*)

Specifies the extended data that is passed to the external program CVTNAME. When CVTFLAG is not *NONE, the contents of the parameter are passed to provide extended flexibility in controlling how the iSeries names are stored in the archive. The *System Administrator's Guide* contains more information on CVTNAME.

External Program Conversion Extended Data
Specify up to 255 bytes of unedited data which is passed to the exit program CVTNAME to assist in controlling the program logic.

CVTFLAG

CVTFLAG(*NONE|*Conversion Flags*)

Specifies the flags passed to the external program CVTNAME. These are used to control how the iSeries names are stored in the archive. The *System Administrator's Guide* contains more information on CVTNAME.

The allowable values are:

***NONE** Conversion exit is not active.

Conversion Flags Specify a 5-byte flag that is passed to the exit program CVTNAME to control the program logic. If the name passed back is blank, then conversion is referred back to the setting of the CVTTYPE parameter.

CVTTYPE

CVTTYPE(*NONE|*DROP|*SUFFIX)

Specifies how the files names in the archive will be converted to a file name in the iSeries library, file, and Member format. In the iSeries QSYS library system, the length of each name in the QSYS format can only be up to 10 characters. In other platforms, the file name formats (including MS/DOS) may have an extension with a period (.) separator which is not valid in the iSeries DB name. The file names in some cases may even exceed the 10-character limit. This parameter gives control over the file name conversion process.

Note: The conversion of file names may result in duplicate file names on the iSeries system. In this case, the rules for overwriting the files are in effect for duplicates (see the OVERWRITE option). If this is the case, using specific file inclusion and exclusion with multiple runs may be required to extract all of the files.

The allowable values are:

***SUFFIX** This forces the removal of the period(.) extension and stores name truncating characters over 10 characters.

***NONE** This leaves the iSeries name as the archive name.

***DROP**	Drops all characters after the period(.) extension separator, and stores the name truncating characters over 10.

DFTDBRECLN

DFTDBRECLN (132|Record Length)

Specifies the record length to use when creating a file in the QSYS library system. If TYPFL2ZP parameter is *DB, and the file being extracted does not exist nor does extended attribute for the record length exist, the file will be created with the record length specified in this parameter.

The allowable values are:

132	Default is record length of 132 to match previous versions.
Record Length	A decimal number from 50 to 32000.

DROPPATH

DROPPATH(*NONE|*ALL| *LIB)

Used to drop the path(s) or libraries of files that are stored in the archives, therefore only using the file names in the archive. This is used along with the keyword EXDIR where the default paths are defined when dropping the paths on files in the archive.

For example, if the file in the archive is “path1/path2/filename” (IFS) or “library/file/member” (QSYS), and if DROPPATH is *ALL, the file being extracted would be “filename” or “member”. If *LIB was used, the file being extracted would be path1/filename” or “file/member”.

See “Example 1 - PKUNZIP Files to a New or Different Library” in Appendix B for an example of using EXDIR and DROPPATH together.

The allowable values are:

***NONE**	Do not remove paths and/or libraries in the archive.
***ALL**	Remove all paths that are stored in the archive, leaving only an IFS file name or member name.
***LIB**	Remove only the first path (which in most cases could be the library).

ENTPREC

Requires SecureZIP

```
Encryption Recipients       :
     LookUp Type  . . . . . . . .   *DB     *DB, *FILE...
     Recipient  . . . . . . . . . _____________________________________

     Passphrase               . . _____________________________________
     Required . . . . . . . . . .   *RQD          *RQD, *OPT
               + for more values    _
```

Or

ENTPREC((*MBRSET 'pkwareCertAdmin04.p12' (pw) *RQD))
ENTPREC((*FILE
'/yourpath/PKWARE/Cstores/private/pkwareCertAdmin04.p12' (pw) *RQD))
ENTPREC((*FILE
'/yourpath/PKWARE/Cstores/private/pkwareCertAdmin04.pfx' ('mypassphrase') *RQD))
ENTPREC((*DB
'EM=bill.Somebody@pkware.com' (pw) *RQD))
ENTPREC((*LDAP
'EM=bill.Somebody@pkware.com' (pw) *RQD))
ENTPREC((*INLIST 'ATEST/INLIST(ENGNEER1)' *N)

The decryption recipient parameter defines one to many recipients which is to be included for UNZIP process. This parameter allows 1-4 types of certificate searches to take place along with providing the ability for an include file that may contain the recipients.

The specification of this recipient ENTPREC parameter triggers decryption to take place during UNZIP processing utilizing the found recipients along with passphrases that were entered to access the private certificates.

There are four options.

Lookup Type (*NONE |*DB |*FILE |*MBRSET |*SAME)

The Lookup type is the type of recipient search to be used for the recipient string.

- *DB - The Recipient string is defined to search using the Certificate Locator Database to access the digital certificate.
- *FILE - The recipient string is defined to read a specific file in a specific path in the IFS in order to access the digital certificate.
- *MBRSET - The recipient string is defined to read this specific file from the enterprise public certificate store to access the digital certificate.
- *INLIST- The recipient string defines a specific file that will contain one to many recipients.

Recipient (The recipient string name)

The recipient string format depends on what was specified for the Lookup type.

- If type is *DB - The recipient string will either be an email address or the common name of the certificate. This depends on the configuration setting in PKCFGSEC parameter CERTDB. To override the default selection mode, you can prefix the string with EM= for email or CN= for the common name.

  For example:

  ENTPREC((*DB ‘bill.Somebody@pkware.com' (pw) *RQD)
  (*DB ‘CN=bill Somebody' (pw) *RQD)
  (*DB ‘EM=bill.Somebody@pkware.com' (pw) *RQD))

- If type is *FILE - The recipient string is defined to read a specific file in a specific path of the IFS. This file should be Public-key X.509 file or private-key X.509 certificate file.

  For example:

  ENTPREC((*FILE '/yourpath/PKWARE/Cstores/private/pkwareCertAdmin04.p12' (pw) *RQD))

  The digital certificate file ‘pkwareCertAdmin04.cer’ will be in the full path '/yourpath/PKWARE/Cstores/private.

- If type is *MBRSET - The recipient string is defined to read a specific file from private certificate store of the IFS. This file should be a private-key X.509 certificate file.

  For example:

  ENTPREC((*MBRSET 'pkwareCertAdmin04.p12' (pw) *RQD))

  The digital certificate file ‘pkwareCertAdmin04.p12’ will be in the full path of the private certificate store defined in the enterprise security configuration private store(parameter CSPRIV).

- If type is *INLIST- The recipient string defines a full file name of an input list file that contains records of ENTPREC shortcut parameters. The type of file will in the QSYS library file system if TYPLISTFL(*DB) is set and will be a path file name in the IFS if TYPLISTFL(*IFS) is set. The format of the ENTPREC shortcut parameters are define below in the *INLIST Usage section.

Passphrase

The passphrase is required to access private certificates.

Required (*RQD|*OPT|*SAME)

If *RQD, then this recipient MUST be found during the selection and the certificate MUST be valid or the ZIP/UNZIP run will fail.

Usage Notes:

The UNZIP process requires a X.509 private-key format certificate file to decrypt files and thus requires an inputted passphrase.

For an inlist that contains a passphrase to open a private-key certificate, make sure that the security is sufficient to allow read access only to the owner of the certificate. Otherwise other users can browse the passphrase.

***INLIST Usage:**

If *INLIST is defined on the ENTPREC parameter, then the recipient filed will be a file that SecureZIP will read to include recipient. The format is very similar to the ENTPREC parameter describe above except each line recipient starts with "{RECIPIENT=" and is terminated by the "}" character with the semi-colon ";" as a separator for each entry.

{RECIPIENT=Lookup Type; Recipient; Passphrase; Required}

Lookup Type	See Lookup Type in ENTREC excluding the INLIST
Recipient	See Recipient in ENTREC.
Passphrase	See Passphrase in ENTREC.
Required	See Required in ENTREC, but use RDQ for *RQD and OPT for *OPT.

Examples:

```
{RECIPIENT=MBRSE;EM; mypassphrase;RQD}
```

Sample 1: tstpriv_db4.inlist.

```
{RECIPIENT=DB;EM=PKTESTDB4@nowhere.com;PKWARE;RQD}
```

Sample 2: tstpriv_mb3.inlist.

```
{RECIPIENT=MBRSET;pktestdb3.pfx;PKWARE;RQD}
```

Sample 3: tstpubl.inlist.

```
{RECIPIENT=MBRSET;pktestdb3.p12;pw;RQD}
{RECIPIENT=MBRSET;pktestdb4.p12;pw;OPT}
```

Sample 4: tstpubl2.inlist.

```
{RECIPIENT=DB;EM=PKTESTDB3@nowhere.com;pw;RQD}
{RECIPIENT=DB;CN=PKWARE Test4;pw;OPT}
```

EXCLFILE

EXCLFILE(*NONE| *path/filename*)

This parameter specifies the file containing the list of files to be excluded. This can be used with or without the EXCLUDE parameter. See parameter TYPLISTFL for file system type information.

***NONE**	No list file will be processed.

path/filename Enter the file path and the name of the file to process. The layout depends on which file system you want the file created.

Library File System:
The format is "library/file(member)".

Integrated File System (IFS):
The format is "path1/path2/../pathn/filename".

EXCLUDE

EXCLUDE(file_specification1, file_specification2,... file_specification *n*)

Specifies the files and file specification patterns that will be excluded from the PKUNZIP run. One or more names can be specified. Each name should be in the OS/400 file system format, such as, QSYS is library/file(member) and IFS is directory/file, and can include wildcards "*" and "?".

Note: If TYPE(*VIEW) is being used, then the format for these names is the MS/DOS format.

The PKUNZIP program can also exclude file specifications by using the list file parameter EXCLFILE with a list of names to exclude.

Please refer to "File Selection and Name Processing" in chapter 1 for details of file specification formatting.

The valid parameter values for the FILES keyword are as follows:

'file_specification 1'

'file_specification 2'...

'file_specification n'

EXDIR

EXDIR(*CURRENT| *path*)

If there are no paths stored in the archive file name, EXDIR specifies the default path to store the files being extracted. The path definition depends on the "file system type" in parameter TYPFL2ZP. This will happen when the files come from a PC or if the files were compressed with ***PKZIP[i]*** using the STOREPATH(*NO) parameter.

If the "file system type" is IFS, EXDIR will be the paths defined for your iSeries open systems and the default path will be the current directory settings (issue the command DSPCURDIR to see the current directory settings).

If the "file system type" is the library file system, the path will be either a library or a library/filename. The default is *CURLIB/UNZIPPED and if the file UNZIPPED does not exist, then it is created with a record length of 132. It is best to create a default file with the record length of your choice, because if a text file is extracted with a record length greater than the file's record length, the record will be truncated to fit the record length.

If EXDIR is coded with keyword MBR and the file system is the QSYS library system, PKUNZIP will use the member name for the file name. For example: EXDIR('newlib/MBR') and DROPPATH(*ALL) parameters are coded and the file name in archive is "mylib/myfile/mymbr", the file will be extract to the file "newlib/mymbr(mymbr)". This is only valid for TYPFL2ZP(*DB) files.

EXDIR is also used when the archive file is a GZIP archive and there is no file name stored in the archive. In this case, EXDIR becomes a required field.

***CURRENT** Current directory for IFS or *CURLIB/UNZIPPED for the QSYS library file system.

path Enter the path or path/path/.. in which to extract. The layout depends on the file system in which the file is to be created.

Library File System:
The format is "library/file".

Integrated File System (IFS):
The format is "path1/path2/../pathn".

FILES

FILES(file_specification1, file_specification2,... file_specification *n*)

Specifies the files and file specification patterns that will be selected in the PKUNZIP process. One or more names can be specified. Each name should be in the OS/400 file system format, such as, QSYS is library/file(member), and IFS is directory/file, and can include wildcard "*" and "?".

Note: If TYPE(*VIEW) is being used then the format for these names is the MS/DOS format.

The PKUNZIP program can also have file specification selections to include by using the list file parameter INCLFILE with a list of names to select.

Files may also be excluded. See the EXCLUDE parameter.

Please refer to "File Selection and Name Processing" in chapter 1 for details of file specification formatting.

The valid parameter values for the FILES keyword are as follows:

'file_specification 1'

'file_specification 2'

'file_specification n'

FILETYPE

FILETYPE(*TEXT|*BINARY|*EBCDIC|*DETECT)

Specifies whether the files selected are treated as text or binary data. For text files added to an archive, trailing spaces in each line are removed, the text is converted to ASCII (based on the translation tables) by default, and a carriage return and line

feed (CR/LF) are added to each line before the data is compressed into the archive. Binary files are not converted at all.

There are attributes which indicate how a file was compressed (TEXT, BINARY, or a SAVF) in the archive headers. The default setting (and recommended) is *DETECT, which analyzes the header to determine the file type. To view the attribute settings of a file, use the VIEWOPT(*DETECT).

If the file is a SAVF, then it will be processed as BINARY, regardless of any option that you select.

***DETECT**	Uses the attribute setting that is stored in the archive to determine the file type.
***TEXT**	Specifies that the files selected are text files and translation will be performed using the translate tables specified in the TRAN option.
***BINARY**	Specifies that the files selected are binary files and no translation should be performed.
***EBCDIC**	Specifies that the files selected are text files and leaves it in EBCDIC without performing any translation. This is good only if the files are to be used on an iSeries or IBM-type mainframe. If they are unzipped to a PC file, then a translation from EBCDIC to ASCII is required.

FTRAN

FTRAN(*ISO88591 |*INTERNAL| *Member Name*)

Specifies the translation table for use in translating "file names, comments, and passphrase" from the iSeries EBCDIC character set to the character set used in the archive file (normally ASCII character set). A default internal table is predefined. See Appendix D for additional information.

***ISO88591**	The predefined internal table for translation. This table provides translation that is consistent with the ISO 8859-1 definitions. This table uses the EBCDIC code page 037 and the ASCII code page 819 for translation.
***INTERNAL**	To provide some compatibility to pre V8 version, *INTERNAL will use the internal tables that were the default in V5 PKZIP.
membername	Specify the member name in the file PKZTABLES that will be parsed and used to translate "file names and comments" files to the archive character set. The member should have the exact format of member ISO9959_1 in file PKZTABLES. See Appendix D for information on defining translation tables.

IFSCDEPAGE

IFSCDEPAGE(*NO | *Code-Page*)

If this option is set to *NO, PKUNZIP will write IFS files with the code page that is registered for the file, or will use the default job code page if no code page is set in the file attributes. Otherwise, PKUNZIP will write IFS files with the specified code page.

Note: If files are to be extracted to a case sensitive file system, the case sensitive format of file names must be used before they can be selected.

The allowable values are:

***NO**	The PKUNZIP program will read IFS files with the code page registered for the file. This is the default.
Code-Page	The PKUNZIP program will write the IFS files with the specified code page value.

INCLFILE

INCLFILE(*NONE| *path/filename*)

This parameter specifies the file containing the list of files to be selected for including. This can be used with or without the FILES parameter. See parameter TYPLISTFL for file system type information.

***NONE**	No include list file will be processed. This is the default.
path/filename	Enter the file path and name of the file to process. The layout depends on which file system you want the file created.

Library File System:
The format is "library/file(member)".

Integrated File System (IFS):
The format is "path1/path2/../pathn/filename".

MSGTYPE

```
Outlist Details:
        Type . . . . . .   *BOTH          *BOTH, *SEND, *PRINT
        Licence Info.  . . . . .   *NORMAL       *NORMAL, *SHORT, *NONE
```

Or

MSGTYPE (*SEND)
MSGTYPE (*BOTH *SHORT)
MSGTYPE (*BOTH *NONE)
MSGTYPE (*PRINT)

This parameter specifies where displayed output will be outputted and the type of licensing splash screen to provide.

Detail Type (*BOTH |*PRINT |*SEND)

Specifies where the display of messages and information should be shown. The PKZIP program can send messages that appear on the log and/or can print to stdout and stderr. If you are working interactively, stdout and stderr will show up on the dynamic screen. If working via batch, you can override stdout and stderr to print in an OUTQ or build a CL and save to an outfile.

***BOTH**	Send the information to the log with send message commands and also to stdout and stderr.
***PRINT**	Send the information to stdout and stderr.
***SEND**	Send the information to the log with send message commands.

License Info (*NORMAL |*SHORT |*NONE)

Specify the amount of detail license/copyrights information pertaining to ***PKZIP**^{i}* product that will be displayed.

***NORMAL**	Displays all license and copyright information.
***SHORT**	Displays base licensing/copyrights.
***NONE**	Displays only registration information.

OVERWRITE

OVERWRITE(*NO|*YES|*PROMPT}

Controls how PKUNZIP reacts to files that are being extracted and the file already exists. To help prevent accidental overwriting of files, the default is *PROMPT.

The allowable values are:

***YES**	Always overwrite files. If the file exists, the file will be overwritten with no message or prompting.
***NO**	Never overwrite files. If the file already exists then the archive file will be skipped and not extracted. This is the default.
***PROMPT**	When a file being extracted already exists, PKUNZIP will issue the warning message AQZ0262 and prompt the user for the required action.

PASSWORD

PASSWORD(Archive Passphrase)

Specifies a decryption passphrase to be used for files that were encrypted to the archive with a passphrase. This passphrase may be up to 260 characters in length and is case-sensitive. All files selected for archiving will be checked for encryption using the specified passphrase. Files in the archive may have different passphrases. If so, PKUNZIP must be run once for each passphrase.

Since the passphrase in entered in EBCDIC, the translation table referenced in the FTRAN parameter is used to translate it to ASCII. Care should be taken when using the FTRAN override and when using a passphrase. To use passphrase-protected files, the same FTRAN override option is required.

If the contents of the PASSWORD() parameter starts with the key word *INLIST;, then the passphrase will be retrieved from the Inlist file defined in the PASSWORD() parameter.

***INLIST Usage**

To utilize the inlist file for a passphrase, the PASSWORD parameter must start with the keyword *INLIST;. If the inlist file is not from the same file system that is set with the TYPLISTFL(*IFS/*DB) parameter, then code a *DB; or *IFS; to describe the file system where the following inlist will reside. Following the *INLIST; or *IFS;/*DB;, the file name should be specified. Using the inlist method for passphrase allows the opportunities to secure a file of a passphrase with the iSeries object authorities. For more information on INLIST see Appendix C.

Examples of PASSWORD() passphrase inlist file coding:

- **PASSWORD('*INLIST;ATEST/PPINLIST(MBR01)')**

  where TYPLISTFL(*DB) and the file is PPINLIST in library ATEST for file member MBR01

- **PASSWORD('*INLIST;*DB;ATEST/PPINLIST(MBR01)')**

  demonstrates overriding the TYPELISTFL

- **PASSWORD('*INLIST;/myroot/PKINLIST/PP01.inl')**

  where TYPLISTFL(*IFS) and the path to the inlist file is '/myroot/PKINLIST/' with stream file name of 'PP01.inl'

- **PASSWORD('*inlist;*IFS;/myroot/PKINLIST/PP01.inl'))**

  demonstrates overriding the TYPELISTFL

The passphrase inlist file must contain "PASSPHRASE={" and be terminated by the "}" character. The passphrase used will be all of the bytes that exist between the {} not including the { or the }. Null bytes and end-of-record bytes are ignored. Therefore the passphrase structure should be only on one record of an inlist file.

Example of contents of an inlist file:

```
PASSPHRASE={x12345678901234x}
```

The passphrase used will be x12345678901234x in EBCDIC. This will then be translated to ASCII using the FTRAN parameter.

Care should be taken that the file is EBCDIC and has a correct code page. It will use all bytes of data between the {}, even non-displayable bytes.

After a passphrase inlist file is set up, it should be tested with both PKZIP and the PKUNZIP command.

RSTIPSRA

RSTIPSRA (For iPSRA files enter a restore command)

If an iPSRA file is to be restored, RSTIPSRA should contain the appropriate restore command for the objects. To view the save command that was used to create the iPSRA file, do a TYPE(*VIEW) VIEWOPT(*ALL). This parameter should contain the restore command with no surrounding quotes. When the cursor is position to a restore command entered in the RSTIPSRA parameter, it can be prompted. If the restore command cannot pass the command pre-processor, an error will show for the restore command. Valid restore commands are: RST, RSTLIB, RSTOBJ, and RSTDLO.

SFQUEUE

SFQUEUE (*DFT |Name)

Specifies the output queue that will be used as an override when extracting spool files. If no OUTQ library is specified, it will default to *LIBL.

The allowable values are:

***DFT**	The output queue that are in the spool file attributes will be used when extracting files.
OUTQ	The specific OUTQ that will used when the spool file is extracted. It must be a valid output queue.
OUTQ Library	The library where the OUTQ resides.

SPLUSRID

SPLUSRID (*DFT| User ID)

The user ID to use when extracting a spool file. If *DFT is used the user ID belonging to the spool file will be used when building the spool file.

The allowable values are:

***DFT**	Use user ID associated with spool file in the archive.
User ID	Specify a valid user ID that the new extracted spool file will belong to. It must be a valid user ID on the OS/400.

Note on extracting Spool Files: To create or extract a spool file with PKUNZIP, the user must have *USE authority to the API QSPCRTSP. The normal setting for the API QSPCRTSP is authority PUBLIC(*EXCLUDE). The API authority is set this way so that system administrators can control the use of this API. This API has security implications because you can create a spooled file from the data of another spooled file. To allow user to extract spool files change the API authority on a need basis.

TRAN

TRAN(*ISO88591 |*INTERNAL| *Member Name*)

Specifies the translation table for use with translating "data" from the iSeries EBCDIC character set to the character set used in the archive file (normally the ASCII character set). A default internal table is predefined (see Appendix D).

***ISO88591**	The predefined internal table for translation. This table provides translation that is consistent with the ISO 8859-1 definitions. This table uses the EBCDIC code page 037 and the ASCII code page 819 for translation.
***INTERNAL**	To provide some compatibility to pre V8 version, *INTERNAL will use the internal tables that were the default in V5 PKZIP.
Member Name	Specifies the member name in the file PKZTABLES that will be parsed and used to translate data files to the archive character set. The member should have the exact format of member ISO9959_1 in file PKZTABLES (see Appendix D for information on defining translation tables).

TYPARCHFL

TYPARCHFL(*DB |*IFS |*XDB)

Specifies the type of file system in which the archive file will exist (see parameters ARCHIVE and TMPPATH for additional information).

***DB**	Archive files are to be in the QSYS library file system. Even though *DB is working with archive files that are in the QSYS library file system, the IFS is utilized for performance and for large file support (ZIP64). To provide an option for archive file reading utilizing exclusively the QSYS library system, use TYPARCHFL(*XDB), which will support OS400 features such as Adopt Authority.
***IFS**	Archive files are to be in the integrated files system (IFS).
***XDB**	The archive file will be read exclusively utilizing the QSYS library file system during processing. This option will not provide large file support or ZIP64 features.

TYPFL2ZP

TYPFL2ZP(*DB|*IFS)

Specifies the type of file system that contains the files to be unzipped. Reflected for files in parameters FILES and EXCLUDE.

***DB**	Files to be unzipped are in the QSYS library file system.
***IFS**	Files to be unzipped are in the IFS (integrated files system).

TYPLISTFL

TYPLISTFL(*DB|*IFS)

Specifies the "type of files system" that will be used for the input list file and/or the output list file of selected items.

To use input list files, see parameters INCLFILE (file section list) or EXCLFILE (file exclude list). To create an output list file of the selected files items, see parameter CRTLIST.

***DB**	Files are in the QSYS library file system.
***IFS**	Files are in the IFS(integrated file system).

VERBOSE

VERBOSE(*NORMAL|*NONE| *ALL|*MAX)

Specifies how the detail will be displayed during a PKUNZIP run.

The allowable values are:

***NORMAL**	Displays most informative messages to show PKUNZIP is processing.
***NONE**	Displays only major exception information.
***ALL**	Displays all messages.
***MAX**	Used only for debugging purposes.

VIEWOPT

VIEWOPT(*NORMAL|*DETAIL|*BRIEF|*COMMENT|*FNE|*FNEALL)

Specifies the level of information produced when viewing the archive.

The allowable values are:

***NORMAL**	Shows the original file length, compression method, compressed size, compression ratio, file date and time, 32-bit CRC value, and file name for each file in the archive.
***DETAIL**	Shows very detailed technical information about each file in the archive. It will also show all extended attribute (extra data fields) information that was stored in the archive produced by PKZIP (only if the PKZIP keywords EXTRAFLD(*YES) or DBSERVICE(*YES) were specified).

***BRIEF**	Shows the original file length, file date and time, and file name for each file in the archive.
***COMMENT**	Same as the *NORMAL option, but also shows any file comments stored on a separate line after its details.
***FNE**	Shows the archive's file name encryption properties.
***FNEALL**	Shows the archive's file name encryption detail properties including the allowable recipients.

VIEWSORT

VIEWSORT(*ASIS|*DATE|*DATER|*NAME|*NAMER|*PERCENT|*PERCENTR| *SIZE|*SIZER))

Specifies the sequence of the viewing display.

The allowable values are:

***ASIS**	List the files in the sequence in which they are stored in the archive, such as, as is.
***DATE**	List the files in ascending order of the file's date & time as stored in the archive.
***DATER**	List the files in descending order of the file's date & time as stored in the archive.
***NAME**	List the files in ascending order of the file name as stored in the archive.
***NAMER**	List the files in descending order of the file name as stored in the archive.
***PERCENT**	List the files in ascending order of the compression percentage as stored in the archive.
***PERCENTR**	List the files in descending order of the compression percentage as stored in the archive.
***SIZE**	List the files in ascending order of the uncompressed file size as stored in the archive.
***SIZER**	List the files in descending order of the uncompressed file size as stored in the archive.

9 PKQRYCDB "Query Cert Database" Command

PKQRYCDB Requires SecureZIP

PKQRYCDB Command Summary with Parameter Keyword Format

PKQRYCDB is a utility command to query the certificate locator database files or a certificate file in the IFS.

Keywords are demarcated by spaces. In many cases there are multiple entries for a parameter where each entry is again demarcated by spaces. For more information about the command process reference the IBM home page for your version of the operating system.

```
                    SecureZIP Query Cert Db  (PKQRYCDB)
Type choices, press Enter.
Processing Type  . . . . . .   *SUMMARY      *SUMMARY, *LEVEL1, *ALL
 File Type  . . . . . . . . .   *DB           *FILE, *DB, *P7B
 Certificate Type . . . . . .   *ALL          *PUBLIC, *PRIVATE, *ALL
 Selection Name . . . . . . . _____________________________________________
_____________________________
 Cert Passphrase. . . . . .
 Logging Level  . . . . . .   *LOG          *NOLOG, *LOG, *MAXLOG
```

PKQRYCDB Command Keyword Details

RUNTYPE - Processing Type

RUNTYPE (*SUMMARY|*LEVEL1 |*SELECT|*ALL)

The processing type determines the amount of details that PKQRYCDB will display. The possible type codes are:

*SUMMARY - Shows only one line per selected item and is based on the selection type (CN= or EM=)

*LEVEL1 - Displays the common name, email address and the certificate path and file name

*SELECT - Displays a display file of certificates based on the selection type. The items can be browsed or selected for a detail display of the certificates. If the certificate dates have expired, the dates will be highlighted.

*ALL - Displays a complete set of details for each certificate; could be 20-40 lines per file

FTYPE - File Type

FTYPE (*FILE| *DB | *P7B)

The file type determines the type of path/file name in the parameter FNAME.

If *DB is selected, PKQRYCDB will search the database based on the contents of the FNAME. For example, CN=Bill* will search for all certificates with a common name that starts with Bill regardless of upper case or lower case.

If *FILE is selected, then FNAME should be a very specific certificate file (full path included).

*P7B will read a specific file that should be in a P7B format. It will then do a detailed display for the contents of the P7B certificate store.

CTYPE - Certificate Type

CTYPE (*ALL| *PUBLIC | *PRIVATE)

CTYPE specifies the type of certificates, private or public, that will be processed in this run.

*ALL will process both public and private certificates.

*PUBLIC specifies that only public certificates should be processed. No passphrase should be supplied.

*PRIVATE indicates that only private-key certificates should be processed and requires that a passphrase be entered.

FNAME - File Name

FNAME (Path/File name)

If FTYPE is *DB, the FNAME contents will be the selection criteria for the certificate locator database. It should contain the prefix of the field to select, such as CN= for common name and EM= for email address. Selection is not case-sensitive. If the selection ends in an asterisk (*), a generic selection is made for all certificates starting with the selection criteria.

If FTYPE is *FILE, the contents of FNAME contains the IFS file that will used to query the certificate contents. Specify the full path and file name of the specific certificate file.

PASSWORD - Certificate Passphrase

PASSWORD (Certificate Private Key Passphrase)

Processing the private key certificate with RUNTYPE(*ALL) requires the passphrase used when the certificate was exported to open and gather the contents. The passphrase is used only to open the certificate to gather the database data; it is not stored or saved. The certificate is not altered in any way.

LOGLVL - Logging Level

LOGLVL (*LOG|*NOLOG |*MAXLOG)

Specifies the level of logging (printing/viewing) used during a PKQRYCDB run. LOGLVL(*NOLOG) shows only a minimal amount of information. LOGLVL(*MAXLOG) shows more details, with some detail useful only for problem determination.

Sample Displays

Request RUNTYPE(*SUMMARY) to generate and display a report containing additional information about the certificate.

➔ PKQRYCDB RUNTYPE(*SUMMARY) FNAME('cn=will*')

```
PKQRYCDB QUERY SecureZIP Cert DataBase starting------2004/11/16 07:37:28
PKQRYCDB Start Search Summary for <cn=will*>
 Public Key CN=William S. Somebody
 Public Key CN=William Somebody
 Public Key CN=William Somebody
Private Key CN=William Somebody
 Public Key CN=William Somebody


PKQRYCDB Run Totals:
   Total Records In Error  =0
   Total Records Processed =5
PKQRYCDB Scan ending------
```

Request RUNTYPE(*Level1) to generate and display a report containing additional information about the certificate.

➔ PKQRYCDB RUNTYPE(*LEVEL1) FNAME('cn=will*')

```
PKQRYCDB QUERY SecureZIP Cert DataBase starting------2004/11/16 07:39:34
PKQRYCDB Start Search Level 1 for <cn=will*>
 Public Key CN=William S. Somebody
            EM=sombody@worldnet.att.net
            FN=William S. Somebody
            File </yourpath/PKWARE/Cstores/public/williamsSomebody.cer>
 Public Key CN=William Somebody
            EM=bill.Somebody@pkware.com
            FN=William Somebody
            File </yourpath/PKWARE/Cstores/public/billSomebody03.cer>
 Public Key CN=William Somebody
            EM=bill.Somebody@pkware.com
            FN=William Somebody
            File </yourpath/PKWARE/Cstores/public/bill_Somebody2003.cer>
```

```
Private Key CN=William Somebody
            EM=bill.Somebody@pkware.com
            FN=William Somebody
            File </yourpath/PKWARE/Cstores/private/billSomebody03.pfx>
 Public Key CN=William Somebody
            EM=bSomebody@pkware.com
            FN=William Somebody
            File </yourpath/PKWARE/Cstores/public/billSomebody.cer>


  PKQRYCDB Run Totals:
     Total Records In Error  =0
     Total Records Processed =5
  PKQRYCDB Scan ending------
```

Request RUNTYPE(*ALL) to generate and display a report containing additional information about the certificate.

➔ **PKQRYCDB RUNTYPE(*ALL) FTYPE(*FILE) FNAME('/yourpath/PKWARE/Cstores/public/billSomebody03.cer')**

```
PKQRYCDB QUERY SecureZIP Cert DataBase starting------2004/11/16 07:43:50


 ----------------------------------------------------------
 Public Key Found File </yourpath/PKWARE/Cstores/public/billSomebody03.cer>
            CN=William Somebody
            EM=bill.Somebody@pkware.com
            FN=William Somebody

--- Certificate ---
William Somebody
Subject:
   O=VeriSign, Inc.
   OU=VeriSign Trust Network
   OU=www.verisign.com/repository/RPA Incorp. by Ref.,LIAB.LTD(c)98
   OU=Persona Not Validated
   OU=Digital ID Class 1 - Microsoft Full Service
   CN=William Somebody
   E=bill.Somebody@pkware.com
Issuer:
   O=VeriSign, Inc.
   OU=VeriSign Trust Network
   OU=www.verisign.com/repository/RPA Incorp. By Ref.,LIAB.LTD(c)98
   CN=VeriSign Class 1 CA Individual Subscriber-Persona Not Validated
SerialNumber:
   3F55 2A91 2B5A 9F9B 46E0 D8A0 96DB DDAB
NotBefore:
   Mon Jul 21 19:00:00 2003
NotAfter:
   Wed Jul 21 18:59:59 2004
SHA-1 Hash of Certificate:
   D5 CE FF A5 72 EF B6 53 EA 75 F7 CA 2E 01 85 7B
   65 7C B8 E7
Public Key Hash:
   6E 16 CF EF FA A0 99 25 2B 79 DE E6 23 C7 D7 42
   80 82 F3 E4
End Entity


PKQRYCDB Run Totals:
   Total Records In Error  =0
   Total Records Processed =1
PKQRYCDB Scan ending------
```

The following table explains the fields of the certificate details in the display.

Heading	*Description*
Subject	Information about the entity to whom the certificate was issued
Issuer	Information about the entity that issued the certificate
Serial Number	Serial number of the certificate
NotBefore/NotAfter	Date range for which the certificate is valid
SHA-1 Hash of Certificate	The SHA-1 algorithm hash, or “thumbprint,” of the certificate
Public Key Hash	The hash, or “thumbprint,” of the public key
Key Usage	Key usage flags that determine how the certificate was intended to be used

The public key hash value is the prime key used in the local certificate store index.

The *Issuer* fields are composed of several x.509 subfields. The exact set varies. The following table describes some of the most commonly used.

Code	*Description*
O	Organization
OU	Organizational Unit
CN	Common Name
E or **EM**	Email address
C	Country
ST	State or Province
L	Locality or City

The common name (CN) and email (E) fields can be searched to identify recipients.

Request RUNTYPE(*SELECT) to generate a browse screen containing additional information about the certificate. This provides the ability to fold and unfold for more information. To display details as shown above, enter a 5.

➔ **PKQRYCDB RUNTYPE(*SELECT) FNAME('cn=P*')**

Folded

```
  4/06/05 08:20:04    Query Certificate Database                    PKQCD01D
                      *CN=PKWARE Test9
 Type option - Press Enter.
   5-View     8-Verify
 Option    Document
_  CN=PKWARE Test1
_  CN=PKWARE Test3
_  CN=PKWARE Test3
_  CN=PKWARE Test4
_  CN=PKWARE Test4
_  CN=PKWARE Test9

F3-Exit                           F9-Fold/UnFold     F12-Return
```

F9 to Unfold

```
 4/06/05 08:20:04    Query Certificate Database                      PKQCD01D
                     *CN=PKWARE Test9
Type option - Press Enter.
  5-View     8-Verify
Option    Document
  CN=PKWARE Test1
  Public   04/14/2004-04/13/2024  NOTTRUSTED NOTREVOKED  Code= CES
  EM=PKTESTDB1@nowhere.com
  File=/yourpath/testroot/CStore/Public/pktestdb1.cer


  CN=PKWARE Test3
  Public   12/20/2004-12/13/2024  TRUSTED    NOTREVOKED  Code= E
  EM=PKTESTDB3@nowhere.com
  File=/yourpath/testroot/CStore/Public/pktestdb3.crt

                                                                        +
F3-Exit                           F9-Fold/UnFold     F12-Return


 4/06/05 08:20:04    Query Certificate Database                      PKQCD01D
                     *CN=PKWARE Test9
Type option - Press Enter.
  5-View     8-Verify
Option    Document
  CN=PKWARE Test3
  Private  12/20/2004-12/13/2024  TRUSTED    NOTREVOKED  Code= E
  EM=PKTESTDB3@nowhere.com
  File=/yourpath/testroot/CStore/Private/pktestdb3.p12


  CN=PKWARE Test4
  Public   12/20/2004-12/13/2024  TRUSTED    NOTREVOKED  Code= E
  EM=PKTESTDB4@nowhere.com
  File=/yourpath/testroot/CStore/Public/pktestdb4.crt


                                                                       +
F3-Exit                           F9-Fold/UnFold     F12-Return


 4/06/05 08:20:04    Query Certificate Database                      PKQCD01D
                     *CN=PKWARE Test9
Type option - Press Enter.
  5-View     8-Verify
Option    Document
  CN=PKWARE Test4
  Private  12/20/2004-12/13/2024  TRUSTED    NOTREVOKED  Code= E
  EM=PKTESTDB4@nowhere.com
  File=/yourpath/testroot/CStore/Private/pktestdb4.p12


  CN=PKWARE Test9
  Private  02/08/2005-12/14/2024  TRUSTED REVOKED        Code= E
  EM=PKTESTDB9@nowhere.com
  File=/yourpath/testroot/CStore/Private/pktestdb9.pfx


F3-Exit                           F9-Fold/UnFold     F12-Return
```

10 Processing with GZIP

Introduction to GZIP (GNU zip)

GZIP (GNU zip) is a compression utility designed to use a different standard for handling compressed data in an archive. Its main advantages over other compression utilities are much better compression and freedom from patented algorithms. It has been adopted by the GNU project and is now relatively popular on the Internet. GZIP was written by Jean-Loup Gailly (jloup@gzip.org) and Mark Adler (the decompression code).

GZIP (GNU zip) utility program (available on a number of platforms including MVS, UNIX, and PC) can be used like ***PKZIP***^{i} to compress and extract data. ***PKZIP***^{i} in producing GZIP archives implements two GZIP standard specifications:

RFC 1952: GZIP file format specification Version 4.3, which documents the GZIP specifications and the format of a GZIP archive file.

RFC 1951: DEFLATE Compressed Data Format Specification Version 1.3, which documents the compression algorithm used by GZIP processing.

The RFC is a process to promote specifications and standards throughout the Internet community and can be found at www.faqs.org/rfcs. Both RFC 1952 and RFC 1951 specifications are platform-independent; therefore, data that was compressed on one platform, for example, UNIX, may be decompressed on another platform, for example, iSeries or MVS.

The one significant advantage of GZIP archive files over ZIP archive files is the ability to handle larger (greater than 4 GB) file sizes. The standard ZIP archive format restricts processing to uncompressed files that are less than 4 GB and cannot create an archive containing multiple files that would meet or exceed the 4 GB limit. These restrictions are due to the size of the specified fields (4 byte fields) that contain the file size information within the archive. The GZIP archive format (see RFC 1952) can process files of any size. This format does not maintain a 'directory' of information for individual files and allows sizes to 'wrap' at 4 GB, so it does not suffer a size restriction.

GZIP Archive Files Used By *PKZIP/SecureZIP for i5/OS*

The term GZIP archive file is used to describe the file that holds data that has been compressed by one of the GZIP programs and meets the specifications of RFC 1952. At the end of the GZIP archive is a trailer that contains the file's compressed size, uncompressed size, and a CRC value for the file (which is used to verify that the decompressed data is identical to the data that was originally compressed).

A GZIP archive file can be transferred from one platform to another and can be decompressed by a GZIP-compatible application which is running on that platform. The internal format of a GZIP archive is identical, no matter what platform compressed the file.

***PKZIP**i* (by default) creates new archives as members of PF-DTA files with 132-byte records. The archive file is given a text field of 'file created by PKZIP iSeries.' The archive member is given a text field of 'Member created by PKZIP iSeries.' If you wish to create your own archive (perhaps because a larger record size would be convenient), then you can do so, but consider the following:

When creating the file, do not create any members in it.

After creating the file, change the MAXMBRS parameter for the file from 1 to *NOMAX.

A GZIP archive holds files internally in either text or binary format, both of which are compatible with other platforms supported by GZIP. Because information held in a GZIP archive is defaulted for binary processing, ***PKZIP**i* uses the parameter FILETYPE for text or binary processing. When transporting archives between machines that use different character sets for text, for example, EBCDIC and ASCII, the binary format may not be appropriate. Specifying that the file is to be compressed as FILETYPE(*TEXT) will allow ***PKZIP**i* to perform EBCDIC-to-ASCII conversion, as required. Specifying FILETYPE(*TEXT) may also be useful when PKUNZIP is used on the iSeries to extract data that has been compressed on an ASCII system.

A GZIP archive is similar to a ZIP archive but normally only contains one compressed file or member. GZIP archives, like ZIP archives, use the Lempel-Ziv algorithm (inflate) to compress and decompress data. Unlike a ZIP archive, GZIP archives do not hold a lot of information in various information blocks throughout the archive. Instead, they contain only one information block at the beginning of the archive and locates size information at the end of the archive. Some GZIP text data, for example, file name and comments, use the ISO 8859-1 (LATIN-1) character set and therefore will be converted to and from LATIN-1 as required. When a GZIP archive is created, an information block is placed in the archive before the compressed version of the file. This information block includes the following information about the file:

- The compression method used on the file.
- The date and time of the last update to the file.
- A flag to indicate optional extended data exists (these fields are usually operating system-dependant and may be ignored if identification code is unrecognized).
- The name of the file that was compressed.
- An archive text comment.

- Compressed data followed by the GZIP trailer at the end of the archive. The trailer includes a CRC value and the original size of the uncompressed file.

Cross Platform Compatibility

Since GZIP archive files adhere to RFC 1952, the files are compatible across all GZIP-supported platforms. If executable files and other platform-dependent objects are compressed on one platform and then decompressed on another, it is unlikely that they will work on the new platform. The same can be said about EBCDIC vs. ASCII. Because the extra information is platform dependent, most likely it will be ignored by another platform.

A major consideration for cross platform processing is when building the archive in the QSYS library file system you may end up with pad bytes at the end of the archive due to files have record lengths and the end of the archive will be padded to the record length. Some GZIP products cannot handle the extra pad bytes at the end of the archive. In this case, the archive should be stored in the IFS where the archive will be a true stream file with no pad bytes at the end of the archive.

GZIP Restrictions

Filename encryption can not be used with GZIP.

Special Note on GZIP Passphrases

GZIP standard processing (RFC 1952) does not normally allow a passphrase to be placed on a GZIP archive. ***PKZIP[i]*** does allow this feature, but its use may cause compatibility issues with other platforms. ***PKZIP for MVS*** does use the same passphrase standard, so GZIP archives with passphrases can be exchanged between ***SecureZIP for i5/OS***, ***SecureZIP for zSeries***, and ***PKZIP for MVS***. Because GZIP archives that are created with a passphrase with ***PKZIP[i]*** or ***PKZIP for MVS™*** are not part of the GZIP standards, these files will probably appear to be corrupt on other platforms.

Processing GZIP Archives

PKZIP[i] can create and extract information from GZIP format archives similar to how it can be used to create and extract information from a ZIP archive. The creation of a GZIP archive and other parameters is exactly like all other processes in ***PKZIP[i]***, including use of extended attributes. To create a GZIP archive file, code the parameter GZIP(*YES). The difference is that the archive can only have one (1) file, and the archive cannot be updated. The PKUNZIP program will identify the GZIP archive and process it accordingly.

The following are the specific GZIP Restrictions that pertain to the PKZIP and PKUNZIP programs:

GZIP Compressing

The code used by ***PKZIP[i]*** follows the standards specified in the two applicable RFC's. Specifically, these are RFC 1952 (GZIP file format specification Version 4.3) and RFC 1951 (DEFLATE Compressed Data Format Specification Version 1.3). ***PKZIP[i]*** should always be able to create a GZIP compatible compressed file and extract data from a GZIP compressed file where the GZIP utility matches these two specifications.

- Parameter COMPRESS(*NO) cannot be used because all GZIP archives must contain compressed data.
- Parameter COMPRESS(*TERSE) cannot be used because terse compression is a non-standard compression method for GZIP.
- Parameter FTRAN is not valid for GZIP because filenames have to be held in the ISO 8859-1(LATIN-1) character set.
- Parameter TYPE (type of processing to be performed) cannot be specified because *DELETE, *UPDATE, *FRESHEN, *MOVEF, or *MOVEU as a GZIP archive cannot be updated once created.

Only one file is supported per GZIP archive. When creating an archive, this means that only one file or member can be identified for inclusion in the archive.

If ***PKZIP[i]*** is used to create and encrypt a GZIP archive using a passphrase, other platforms may not be able to decrypt the data. The encryption algorithm used by ***PKZIP[i]*** in a GZIP archive is similar to that used by PKZIP, but it is not supported as part of the specifications for GZIP.

Once an iSeries SAVF has been zipped into a GZIP archive, the archive will extract on another platform but will not be available as a SAVF. It will just be a binary file and of no use on another platform.

The file name stored in an archive created by ***PKZIP[i]*** will typically contain library, file, and member names (directory components) which relate to the qualifiers of the original iSeries name. According to the GZIP specifications, the file name stored should be the original name of the file being compressed, with any directory components removed. Most GZIP utilities support directory components, and the default ***PKZIP[i]*** processing will include library, file, and member names in the output file name. **Note:** It is not possible to create a GZIP archive using ***PKZIP[i]*** that does not have the file name stored in the archive.

GZIP Extracting

If a file name is present in the GZIP archive, it will be held in the ISO 8859-1 (LATIN-1) character set.

If there is no file name in the archive, one will use the default paths defined in the EXDIR parameter.

If there is more than one compressed file in the archive, only the first file can be processed.

When zipping a file into an archive, the archive cannot already exist. It is not possible to merge or update a GZIP archive.

VIEW processing may not show all of the details from the archive, since some information is not stored (or not stored in convenient locations) in the GZIP archive. For example, the compressed and uncompressed file sizes will typically be shown as zero.

To match the GZIP specifications, the time stored in the archive header will be in universal time format (seconds since 01/01/1970). Because of manipulation during processing, this time will have only a two-second accuracy (will always be divisible by 2) and therefore could be one second off from the original file time.

There is no standard iSeries method to set the creation date of a file. As a result, the time in the GZIP archive is ignored when creating the iSeries output file. It may be viewed by specifying the parameter TYPE(*VIEW).

Sample GZIP Processing

Compressing a file

The following example shows how to compress a file into a GZIP archive. The PKZIP command is used to compress data into an archive. To select the GZIP format for the resulting archive, you must use the GZIP(*YES) option with the PKZIP command:

PKZIP ARCHIVE('MYLIB1/MYARCHFIL(GZ01)') FILES('TESTLIB1/FILE1TXT') TYPE(*ADD) GZIP(*YES)

The command above will compress the text file TESTLIB1/FILE1TXT into the archive MYLIB1/MYARCHFIL(GZ01) in GZIP format. The archive must not already exist or an error message will be generated and the operation will fail.

The output from the above command should look like the following:

```
File MYARCHFIL created in library MYLIB1.

Scanning files for match  ...
File MYARCHFIL in library MYLIB1 with member GZ01 not found.
Found  1 matching files
Member GZ01 added to file MYARCHFIL in MYLIB1.
Member GZ01 removed from file MYARCHFIL in MYLIB1.
Member PZ3AF2447F added to file MYARCHFIL in MYLIB1.
Compressing TESTLIB1/FILE1TXT(FILE1TXT) in TEXT mode
Add  TESTLIB1/FILE1TXT/FILE1TXT  --   Deflating (31%)
Member PZ3AF2447F renamed to member GZ01.
Member GZ01 file MYARCHFIL in MYLIB1 changed.
PKZIP Compressed 1 files in GZIP Archive MYLIB1/MYARCHFIL(GZ01)
PKZIP Completed Successfully
```

11 PKWARE PartnerLink: SecureZIP Partner

This chapter applies only to participants in the PKWARE PartnerLink program. Other readers may skip this section.

PKWARE PartnerLink enables a *sponsor* organization to give *partner* organizations that may not have ***SecureZIP for i5/OS*** the SecureZIP Partner application so that sponsor and partner can use ***SecureZIP for i5/OS*** to securely exchange ZIP archives.

About *SecureZIP Partner for i5/OS*

SecureZIP Partner for i5/OS is a special version of ***SecureZIP for i5/OS***. It provides most of the functionality of the full program but works only with archives created by (or for) a sponsor.

SecureZIP Partner has two modes of operation:

- **Read mode**: Read mode enables SecureZIP functionality to extract files from a ZIP archive signed by a sponsor. In this mode, the program can decrypt and decompress files and authenticate digital signatures.

  In Read mode, the program only extracts; it does not add files to a new or existing archive and does not compress, encrypt, or sign files. SecureZIP Partner extracts only archives digitally signed by a sponsor.

- **Write mode**: Write mode enables SecureZIP functionality for adding files to a ZIP archive, including commands to compress, encrypt, and digitally sign files.

  In Write mode, the program can create and update archives, but only for a designated PartnerLink sponsor and only if the sponsor provides certificates for SecureZIP Partner to use to encrypt. New or updated archives are automatically encrypted for sponsor recipients: only those recipients can decrypt and read the files.

  SecureZIP Partner only does certificate-based encryption. It does not do passphrase-based encryption.

A single copy of the SecureZIP Partner software can process ZIP archives from multiple sponsors.

See the chapter relating to PartnerLink in the *SecureZIP for i5/OS System Administrator's Guide* for a description of administration and configuration activities unique to the SecureZIP Partner product.

If You Are a Sponsor: Sign the Central Directory

A sponsor organization uses SecureZIP as usual to work with archives for, or from, a partner. There is just one special requirement when creating an archive for a partner: In order for the partner to be able to extract the archive, you must sign the central directory of the archive using a certificate included in the Sponsor Distribution Package. A Sponsor Distribution Package is a package that PKWARE assembles for a sponsor to configure partners of that sponsor.

Terms and Acronyms Used in This Chapter

The PKWARE PartnerLink program introduces some new concepts and terminology:

- **Sponsor** – An installation responsible for initiating and defining a PartnerLink sponsor-partner relationship with one or more other installations. A aponsor uses the full-featured SecureZIP product; a partner uses the special ***SecureZIP Partner for i5/OS*** version.
- **Partner** – An installation configured using a particular sponsor's Sponsor Distribution Package (see below) to be a partner of that sponsor. A partner uses ***SecureZIP Partner for i5/OS*** to work with archives from, or for, the sponsor.
- **Sponsor Distribution Package** – A configuration package distributed to a partner on behalf of a sponsor to define the authorization requirements and provide the certificates needed to process ZIP archives from, or for, the sponsor. The package is digitally signed using a PKWARE-assigned certificate.
- **Sponsor File** – A component file in a Sponsor Distribution Package
- **Sponsor Imprint** – A unique digital representation of a registered sponsor-partner relationship within the PKWARE PartnerLink program. This may represent the unique identification of Distribution Package components or of ZIP archives being read.
- **Sponsor/Partner Registration ID** – A unique registration number that identifies a particular sponsor-partner relationship
- **Read mode** – The mode of ***SecureZIP Partner*** UNZIP processing that extracts archives from (and only from) a PartnerLink sponsor configured on the partner's system
- **Write mode** – The mode of ***SecureZIP Partner*** ZIP processing that creates an encrypted ZIP archive for a particular configured PartnerLink sponsor
- **FF** – Acronym for *full-featured* SecureZIP operations, as distinct from those of SecureZIP Partner

PKWARE PartnerLink Program: Overview

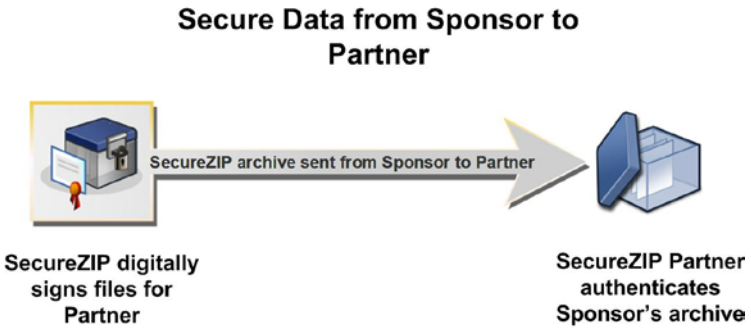
The PKWARE PartnerLink program provides a straightforward, secure way for an organization to exchange sensitive information with outside partners.

A PartnerLink *sponsor* organization establishes a PartnerLink *partner* relationship with another organization. As a PartnerLink partner, the external organization receives the ***SecureZIP Partner for i5/OS*** application to use to decrypt and extract archives created by the sponsor using the full SecureZIP program. The partner can also use the program to create archives for the sponsor that only the sponsor can decrypt.

The SecureZIP Partner program used by a PartnerLink partner extracts archives only *from a sponsor* and creates and encrypts archives only *for a sponsor*.

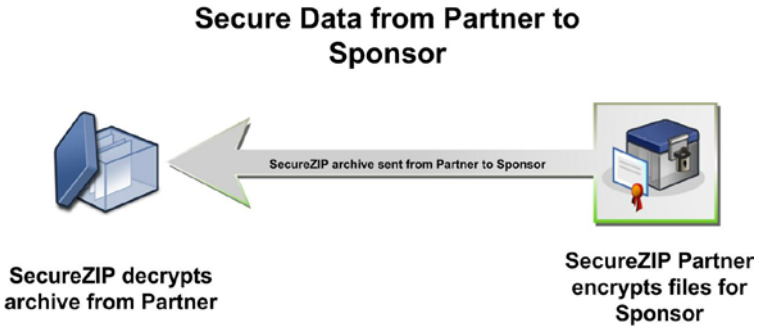
Decrypting and Extracting Sponsor Data (Read Mode)

When SecureZIP Partner is installed at a partner location, a sponsor can create, digitally sign, and encrypt SecureZIP secure containers (ZIP archives) for the partner. In Read mode, the SecureZIP Partner program verifies that the data file received has the appropriate signature from the sponsor and that the signature is valid. This confirms that the data is from the expected sender and that no tampering has occurred. The partner can then decrypt and extract the data.



Creating an Archive for a Sponsor

If a sponsor has provided an encryption key, a partner can also use SecureZIP Partner (Write mode) to create encrypted ZIP archives for the sponsor. SecureZIP Partner automatically encrypts any data placed in an archive. The archive can then be transferred to media or transmitted to the sponsor electronically.



Requirements

License

A license key is provided with the installation package for the system administrator to use to activate the ***SecureZIP Partner for i5/OS*** product.

Operating Environment

SecureZIP Partner for i5/OS requires the same operating environment as full-featured ***SecureZIP for i5/OS***.

Configuring as a Partner for a Sponsor

In order to fully process ZIP Archives, the system administrator for ***SecureZIP Partner for i5/OS*** must install one or more Sponsor Distribution Packages and provide the corresponding run-time configuration information for the ZIP and UNZIP jobs to use. The installed Sponsor Distribution Package determines which archive signatures are approved for Read mode extract processing and defines the list of sponsor recipients for whom SecureZIP Partner encrypts new archives.

Functional Overview

SecureZIP Partner for i5/OS enable a PartnerLink partner to exchange ZIP archives with a sponsor. A Sponsor Distribution Package provides the partner installation with qualifying controls for processing ZIP archives received from or created for a sponsor. Multiple sponsor profiles with unique processing requirements can be configured to support exchanges with multiple PKWARE PartnerLink sponsors.

A given sponsor profile defines the UNZIP and ZIP capabilities for a partner. In a given sponsor-partner relationship, a partner operates in Read mode to extract archives and in write mode to create archives (if write mode functionality is licensed).

See the *SecureZIP for i5/OS System Administrator's Guide* for information on installing Sponsor Distribution Packages.

General Restrictions

Although many features of full-featured ***SecureZIP for i5/OS*** are also available to ***SecureZIP Partner for i5/OS***, some limitations apply for these products.

- ***SecureZIP Partner for i5/OS*** (Read mode) can only open a ZIP archive that has been digitally signed by a qualified and configured sponsor, as specified in the Sponsor Distribution Package.
- ***SecureZIP Partner for i5/OS*** (Write mode) can only encrypt a ZIP archive for a sponsor-designated set of certificate-based recipients.

Attempts to use features that require operational characteristics outside of the bounds set above are rejected or ignored.

SecureZIP Partner IVP Examples

In the distributed SecureZIP library, there is a CL program named PLIVPZIP that runs an initial test with the test distributed package from PKWARE with a Sponsor Id number of 0. The following two examples excerpt steps from the CLP.

Read mode example: Step EXTRACT will read in the signed archive by sponsor 0 and will extract the files to a file TMPTEST. To authenticate the signed archive, AUTHCHK((*ARCHIVE *SPONSOR 0)) is required to read in the sponsor ID number "0" sponsor authentication file.

```
PKUNZIP ARCHIVE('PKW90051L/PLIVPZIP(PLIVPZIP)')
TYPE(*EXTRACT) EXDIR('PKW90051L/TMPTEST')
DROPPATH(*ALL) PASSWORD('PKWARE, Inc.')
AUTHCHK((*ARCHIVE *SPONSOR 0))
```

Sample Results of Step EXTRACT:

```
Digital Certificate Request List:Archive Authenticator
Rqrd    Pub  *SPONSOR                    - a0000000.p7
Archive Authenticator List----------1 processed:
UNZIP Archive: PKW90051L/PLIVPZIP(PLIVPZIP)
Archive Comment:"SecureZIP for zSeries by PKWARE"
Searching Archive PKW90051L/PLIVPZIP(PLIVPZIP)  for files to extract
Archive was signed by "PKWARE PartnerLink TEST Signing Certificate" and verified
Extracting file SECZIP/READER/README.TXT
Inflating *DB:PKW90051L/TMPTEST(READMETXT) Text
SecureUNZIP  extracted     1 files
SecureUNZIP   Completed Successfully
```

Write mode example: Step SLNKZIP will read the file that was extracted above and create a new archive by selecting files TMPTEST(READMETXT) for compression with AES256 encryption. The encryption will use the pubic certificates from the Sponsor ID number "0" recipient file with the parameter ENTPREC((*SPONSOR 0)) or ENTPREC((*SPONSOR 'R0000000.p7')).

```
PKZIP ARCHIVE('PKW90051L/PLIVPZIP(NEWTESTZ)')
FILES('PKW90051L/TMPTEST(READMETXT)') ADVCRYPT(AES256)
ENTPREC((*SPONSOR 0))
```

Sample Results of Step SLNKZIP:

```
Scanning files in *DB for match  ...
Digital Certificate Request List:Encryption Recipients
Rqrd    Pub  *SPONSOR                    -
/yourpath/PKWARE/PLstore/Sponsor/RECIP/r
0000000.p7
Encryption Recipients List----------1 processed:
CN=PKWARE PartnerLink TEST Encryption Certificate
EMail=PKWAREPartnerLinkCA@pkware.com
Found  1 matching files
Compressing PKW90051L/TMPTEST(READMETXT) in TEXT mode
Add  PKW90051.L/TMPTEST/READMETX.T  --  Deflating (69%)  encrypt(BSAFE AES
256Key)
SecureZIP Compressed 1 files in Archive PKW90051L/PLIVPZIP(NEWTESTZ)
SecureZIP Completed Successfully
```

Read Mode (UNZIP) Processing

The following features are provided in Read mode:

- An AUTHCHK(Archive) is automatically performed whenever a ZIP archive is opened, except in the following cases:
  - An AUTHCHK(ARCHIVE) is requested manually
  - Any form of View action
  - A TEST action without any form of AUTHCHK request
- A TAMPERCHECK policy will always be enforced for authentication, regardless of the SecureZIP configuration policy settings.
- The certificate authority trust chain will automatically be honored from the installed and configured Sponsor Distribution Package during archive authentication even if the trusted root certificate is not installed in the local certificate ROOT store.
- If the sponsor also signed files in an archive with the same certificate used to sign the archive central directory, the same certificate authority trust chain used to authenticate the archive signature is used to authenticate signatures on the files.

Restrictions

The following limitations or special behavior applies when ***SecureZIP for i5/OS*** is run in Read/Write mode:

- Archive types (such as GZIP) that do not support signing the archive central directory are not available
- Unsigned archives are rejected for processing

Archive Authentication Settings

The archive authentication that is automatically performed when a ZIP archive is opened for Read mode extract processing uses one or more Sponsor Authentication Configuration Settings to reference an installed Sponsor Authentication File in the certificate store. This is accomplished by including one or more AUTHCHK ((*ARCHIVE *SPONSOR x) ((*ARCHIVE *SPONSOR y)) parameters where x and y are sponsor ID numbers.

- At least one AUTHCHK((*ARCHIVE *SPONSOR x)) command is required to access a ZIP archive for extract processing.
- If more than one Sponsor Authentication Configuration Setting command is provided, then the archive authentication will accept an archive from any of the represented sponsors.

Example: Unzipping and authenticating an archive from sponsor 0:

```
➔ PKUNZIP ARCHIVE('PKW90051L/PLIVPZIP(PLIVPZIP)')
    TYPE(*EXTRACT)
    PASSWORD('PKWARE, Inc.') OVERWRITE(*YES)
    EXDIR('PKW90051L/TMPTEST') DROPPATH(*ALL)
    AUTHCHK((*ARCHIVE *SPONSOR 0) )
```

Sample Results:

```
Digital Certificate Request List:Archive Authenticator
Rqrd    Pub  *SPONSOR                    - a0000000.p7
Archive Authenticator List----------1 processed:
UNZIP Archive: PKW90051L/PLIVPZIP(PLIVPZIP)
Archive Comment:"SecureZIP for zSeries by PKWARE"
Searching Archive PKW90051L/PLIVPZIP(PLIVPZIP)  for files to extract
Archive was signed by "PKWARE PartnerLink TEST Signing Certificate" and verified
Extracting file SECZIP/READER/README.TXT
Inflating *DB:PKW90051L/TMPTEST(READMETXT) Text
SecureUNZIP  extracted     1 files
SecureUNZIP   Completed Successfully
```

Decryption Certificate Selection

RECIPIENT private-key/certificate selection follows the rules for full-featured ***SecureZIP for i5/OS*** local certificate store administration and operations.

File Signature Authentication Certificate Selection

In addition to supporting AUTHCHK *FILES with implicit reference to the AUTHCHK *ARCHIVE certificate validation, separate and distinct file signatory validation can be performed outside of the configured Sponsor Distribution Package. However, this operation is allowed only for files in a sponsor-provided data archive that have signatures for which certificates are not included in the Sponsor Distribution Package.

Public-key certificate files supporting file signature authentication can be supplied through the full-featured ***SecureZIP for i5/OS*** CER certificate types in the local certificate store.

Write Mode (ZIP) Processing

With Write mode, a sponsor-authorized partner can generate a ZIP archive for the sponsor. Data files placed in the created archive are encrypted for a sponsor-designated set of certificate-based recipients. The following special features are provided by Write mode:

- Unless otherwise specified, a minimum encryption method of AES128 is set for newly encrypted files.
- All recipients defined in the sponsor-defined recipient package (as configured from the Sponsor Distribution Package) are included in the encryption request.
- Recipients identified in the sponsor-defined recipient package are subject to the SecureZIP ENCRYPOL policy settings in the certificate store configuration. Individual recipients not passing the designated policy attributes are eliminated from encryption processing.
- The certificate authority trust chain from the installed and configured Sponsor Distribution Package is automatically honored for the designated recipients even if the trusted root certificate is not installed in the local certificate store ROOT. A trusted root is included in the sponsors authentication package.

- When a sponsor-created ZIP archive is used as input to create a new target archive, the same features in effect for Read mode are activated for the input archive. In particular, a signed archive is validated with AUTHCHK.
- When a sponsor-source ZIP archive is used as input to create a new target archive, files copied from the original archive are retained in their original form.
- Newly created archives may be Viewed in accordance with SecureZIP functionality.

Restrictions

The following features are not available or have limitations for ***SecureZIP Partner for i5/OS***:

- GZIP output is not available.
- Self-extracting archives cannot be created.
- An encryption method for supported recipient-based encryption must be used ("Standard" is not supported).
- Passphrase-based encryption for new archives is not available.
- Encryption is only permitted for sponsor-provided keys.
- All archive creation actions require a qualified response recipient configuration as provided by the Sponsor Distribution Package.
- Directory Integration with LDAP access to private-key certificates for decryption and related command settings is not available.
- An archive can be created and encrypted only for recipients associated with a single sponsor: an ENTPREC request must target a configured sponsor, and an archive cannot be created for multiple sponsors. Note, however, that multiple public-key certificates can be included by a given sponsor in one Sponsor Distribution Package. This implementation rules out the use of DB: and LDAP: request formats for the ENTPREC command.
- An output archive with FNE(*YES) can be created in accordance with the qualified sponsor recipient keys. However, because Write mode can create and encrypt archives only for a sponsor, a partner cannot update a filename-encrypted archive *from* a sponsor *for* the partner.

Encryption Certificate Selection

ENTPREC public-key/certificate selection is predefined by the Sponsor Distribution Package. The ***SecureZIP for i5/OS*** local certificate store is extended to support sponsor-provided encryption keys with a lookup type of *SPONSOR. The Write mode ENTPREC command is limited to access only those public-keys supplied in the PartnerLink Sponsor Distribution Package.

Sponsor encryption is accomplished by including the ENTPREC((*SPONSOR x)) parameters, where x is the sponsor ID number or sponsor recipient file (R000000x).

One ENTPREC((*SPONSOR x)) is required encrypt the files for the sponsor.

Example: Encrypting files into an archive for sponsor 0:

```
➔ PKZIP  ARCHIVE('PKW90051L/PLIVPZIP(NEWTESTZ)')
    FILES(&FILES1)
    ADVCRYPT(AES256)
    ENTPREC((*SPONSOR 0))
```

Sample Results:

```
Scanning files in *DB for match  ...
Digital Certificate Request List:Encryption Recipients
Rqrd    Pub  *SPONSOR         -
/yourpath/PKWARE/PLstore/Sponsor/RECIP/r0000000.p7
Encryption Recipients List----------1 processed:
CN=PKWARE PartnerLink TEST Encryption Certificate
EMail=PKWAREPartnerLinkCA@pkware.com
Found  1 matching files
Compressing PKW90051L/TMPTEST(READMETXT) in TEXT mode
Add  PKW90051.L/TMPTEST/READMETX.T  --  Deflating (69%)  encrypt(BSAFE AES
256Key)
SecureZIP Compressed 1 files in Archive PKW90051L/PLIVPZIP(NEWTESTZ)
SecureZIP Completed Successfully
```

12 1Step2Tape Archive Tape Processing

With the 1Step2Tape feature, ***PKZIP***^{i} can create archive files directly to tape by using a *tape device file* (file type *TAPF). Writing the archive files directly to tape eliminates the need to provide disk space for temporary archive files that must then be copied to tape.

The archiving process has two steps: Define the tape attributes in a tape device file, and then specify the tape device file as the ARCHIVE() parameter with the TYPARCHFL(*TAP) option.

The PKZIP command parameter PKOVRTAPF has five options that can override the current tape device file when TYPARCHFL(*TAP) is set. The PKOVRTAPF parameter defaults to the current settings of the *TAPF. The PKOVRTAPF options are:

```
New Archive Tape Overrides:
    Tape Device        . . . . .   *TAPF      _     Tape Device
    Tape File Label    . . . . .   *TAPF            Tape Header
    Tape Sequence Nbr  . . . . .   *TAPF            1-16777216, *TAPF, *END
    File expiration date . . . .   *TAPF            Date, *NONE, *PERM, *TAPF
    End Of Tape Option . . . . .   *TAPF            *TAPF, *REWIND, *UNLOAD...
```

To extract files from an archive on tape:

- Copy the archive from tape to disk
- Run PKUNZIP using the disk copy of the file

Setting Up or Changing a Tape Device File for PKZIP

To write an archive directly to tape, the input parameter for ARCHIVE should be a tape device file that defines the tape output for the archive. Do not confuse tape device files with data files on the tape volumes. For processing volumes which contain data files, the tape device files provide a link between the application program and the tape device.

Included as part of the distribution library is the tape device file named PKTAPEO1. You can view the contents by doing a CHGTAPF pkziplib/PKTAPEO1 and pressing the F4 key. You can tailor this file for your environment with the CHGTAPF command, or you can create and use any other TAPF object that you want. For more information, refer to the CRTTAPF, CHGTAPF, OVRTAPF, and DLTTAPF commands in the IBM CL

programmers guide, or reference the IBM site for iSeries: http://publib.boulder.ibm.com/html/as400/infocenter.html.

In the PKTAPEO1 tape device file, the LABEL parameter specifies the data file identifier or tape header label of the archive file on tape. The PKZIP command provides overrides to the DEV, SEQNBR, LABEL, EXPDATE, and the ENDOPT parameters.

The distributed PKWARE TAPF object was created with the following command:

➔ **CRTTAPF FILE(pkziplib/PKTAPEO1) DEV(TAP01) VOL(*NONE) REELS(*SL) SEQNBR(*END) LABEL('PKZIP.ARCHIVE') FILETYPE(*DATA) TEXT('SecureZIP Archive Tape File') RCDBLKFMT(*FB) CODE(*EBCDIC) EXPDATE(*PERM) ENDOPT(*REWIND)**

The contents of the distributed tape device file are shown below:

```
                         Change Tape File (CHGTAPF)

File . . . . . . . . . . . . . . . FILE          PKTAPEO1
  Library  . . . . . . . . . . .                   PKZIPLIB
Tape device  . . . . . . . . . . . DEV             TAP01
                          + for more values
Volume identifier  . . . . . . . . VOL           *NONE
                          + for more values
Tape reels specifications:         REELS
  Label processing type  . . . .                 *SL
  Number of reels  . . . . . . .                 1
Sequence number  . . . . . . . . . SEQNBR        *END
Tape label . . . . . . . . . . . . LABEL         'PKZIP.ARCHIVE    '
Text 'description' . . . . . . . . TEXT          'SecureZIP Archive Tape File'
                         Additional Parameters

Record length  . . . . . . . . . . RCDLEN        *CALC
Block length . . . . . . . . . . . BLKLEN        *CALC
Buffer offset  . . . . . . . . . . BUFOFSET      0
Record block format  . . . . . . . RCDBLKFMT     *FB
Extend:                            EXTEND
  Extend file  . . . . . . . . .                 *NO
  Check file . . . . . . . . . .
Tape density . . . . . . . . . . . DENSITY       *DEVTYPE
Data compaction  . . . . . . . . . COMPACT       *DEVD
Code . . . . . . . . . . . . . . . CODE          *EBCDIC
Creation date  . . . . . . . . . . CRTDATE       *NONE
File expiration date . . . . . . . EXPDATE       *PERM
End of tape option . . . . . . . . ENDOPT        *REWIND
User label program . . . . . . . . USRLBLPGM     *NONE
  Library  . . . . . . . . . . .
User specified DBCS data . . . . . IGCDTA        *NO
Maximum file wait time . . . . . . WAITFILE      *IMMED
Share open data path . . . . . . . SHARE         *NO
```

Requirements for Writing PKZIP Archives to Tape

- The TAPF device file type must be data or FILETYPE(*DATA) or the archive data will become corrupted
- The tape device file record length (RCDLEN) MUST not exceed 32,764 (the default for PKZIP)

- If the tape device file block length (BLKLEN) is changed, it must be changed by a multiple of 32,764 in the CHGTAPF or CRTTAPF command. On the OVRTAPF command, the BLKLEN must be a multiple of the RCDLEN if the RCDLEN is also overridden.
- The record block format must be fixed block or RCDBLKFMT(*FB)
- The record labels processing must be standard labels or REELS(*SL)
- The archive file type must be *TAP or ARCHTYPFL(*TAP)
- The ARCHIVE() file name must be the name of the tape device file

Notes and Suggestions for Writing Archives to Tape

- To reduce run time, pre-initialize the tapes by defining the tape drive, new volume name and the tape density. For example:

  ➔ **INZTAP DEV(TAP01) NEWVOL(PKZIP) DENSITY(*QIC525)**
- The tape device file should be tailored for your environment.
- If there are files already on the tape, the open may take much longer depending on the TAPF sequence number (SEQNBR).
- If SEQNBR is set to *END, ***PKZIP**^{i} adds the new archive as the last file on the tape. If SEQNBR is set to 1, ***PKZIP**^{i} starts at the beginning and overwrites any files currently on the tape. The checking of current files on a tape is done using the normal IBM file checking based on expiration date, label processing, and so on.
- If you are writing several archives to the same tape, it will run faster if you use the *LEAVE option for the end-of-tape option.
- If you are using a tape that contains multiple files, you should perform a volume listing with “DSPTAP DEV(TAP01) OUTPUT(*PRINT)” to verify what files and sequences are currently on the tape.
- Tape Compression/Compaction: When creating an archive to tape in a non-store mode, overall performance will improve by turning off the tape compaction feature. After compression and/or encryption, data is usually so random that it cannot be further compacted, but the system will test each buffer anyway and sometimes will try. Also, most tape system compaction inserts bytes even when not able to compact.

  To turn off tape compaction, you can either change the tape device file for COMPACT(*NO) or issue a tape override prior to PKZIP. For example:

  ➔ **OVRTAPF FILE(PKTAPEO2) COMPACT(*NO)**
- Support for Optimum Blocks: If you have a tape drive and tape format that supports Optimum Block, you can define a new tape device file to improve performance.

  First determine the maximum optimum block size for your tape device and density. Then set up a tape device file to define the BLKLEN (not to exceed the tape drive maximum size). Define the BLKLEN as a multiple of the default record length of 32,764. For example, if the tape drive model is a SLR60 drive, and the format of the tape is *SLR60, then Optimum Block is supported

with a maximum optimum block size of 256K (262,144 bytes). Overall performance improves with a large block size because fewer tape writes are necessary.

The following examples use a block size of 262,112 (a multiple of the 32,764 record size):

➔ **CRTTAPF FILE(pkziplib/PKTAPEO2) DEV(TAPxx) VOL(*NONE)**
REELS(*SL) SEQNBR(*END) LABEL('PKZIP.ARCHIVE')
FILETYPE(*DATA)
BLKLEN(262112) RCDBLKFMT(*FB)
TEXT('SecureZIP Archive Tape File Optimum Block')
CODE(*EBCDIC) EXPDATE(*PERM) ENDOPT(*REWIND)

- To use a smaller record size than the default of 32,764, you must issue an OVRTAPF command prior to each PKZIP run. Changing the RECLEN in the tape device file does not change record size.

  The following example sets the record size to 8192 and the block size to a multiple of this. The block size must be a multiple of the record length.

  ➔ **OVRTAPF FILE(PKTAPEO2) RCDLEN(8192) BLKLEN(262144)**
  RCDBLKFMT(*FB)

Usage Notes:

Archives written directly to tape by ***PKZIP*^i^** use the ZIP64 data descriptor records format. This ZIP file structure is documented in the ZIP File Format Specification (APPNOTE) published by PKWARE. Not all ZIP-compatible products support data descriptors or ZIP64, and you may experience problems reading these archives if you need to process them for any reason outside of your iSeries environment. We recommend using only PKWARE, Inc. products to ensure the successful recovery of data from your tape archives.

***PKZIP*^i^** issues a tape override for the tape device file at the activation group scope level. If other overrides were made with OVRTAPF previously or at the job level for DEV(), LABEL(), SEQNBR(), EXPDATE(), and ENDOPT() parameters, then ***PKZIP*^i^** will not be able to override the earlier override.

For example, if an earlier override was OVRTAPF FILE(PKTAPEO1) LABEL('My Header'), and the PKZIP PKOVRTAPF contained (*TAPF 'ARCHIVE_TEST24' *END *TAPF *TAPF), then the label written to the tape will be 'My Header'.

Sample - Creating an Archive Directly to Tape

The following examples show the steps to create two archives directly to tape. Most tape processing will normally be performed with a CL program, but it can also be done interactively, as in these examples. For another CLP example, refer to member ZIPEXPL20 in the QCLSRC source file distributed with ***PKZIP*^i^**.

First, make sure the tape has been properly initialized with standard labels. This can be done using the INZTAP command with appropriate parameters for your environment.

➔ **INZTAP DEV(TAP01) NEWVOL(PKZIP)**

In this case, we want the archive to be the first sequence file on the tape. We want the label to be ARCHIVE_TEST01 and an expiration date of 11/08/2005:

➔ **PKZIP ARCHIVE('PKW90051S/PKTAPEO1') FILES('testlib/myfile')**
TYPARCHFL(*TAP)
PKOVRTAPF(*TAPF 'ARCHIVE_TEST01' 1 '11/08/2005' *TAPF)

```
Scanning files in *DB for match  ...
Found  1 matching files
Tape Archive PKW90051S/PKTAPEO1 being created
Compressing TESTLIB/MYFILE(MYMBR) in TEXT mode
Add  TESTLIB/MYFILE/MYMBR  --  Deflating (67%)
Archive <ARCHIVE_TEST01> created using tape device file PKW90051S/PKTAPEO1.
SecureZIP Compressed 1 files in Archive PKW90051S/PKTAPEO1
SecureZIP Completed Successfully
```

The next example creates an archive at the end of the tape. We give the header the name ARCHIVE_TEST02. This time we specify VERBOSE(*MAX) to show the overrides processed:

➔ **PKZIP ARCHIVE('PKW90051S/PKTAPEO1') FILES('testlib/myfil*')**
TYPARCHFL(*TAP)
PKOVRTAPF(*TAPF 'ARCHIVE_TEST02' *END *TAPF *REWIND)
VERBOSE(*MAX)

```
Scanning files in *DB for match  ...
Found  4 matching files
Tape Archive PKW90051S/PKTAPEO1 being created
Archive Tape override <OVRTAPF FILE(PKTAPEO1) TOFILE(PKW90051S/PKTAPEO1) LABE
L('ARCHIVE_TEST02') SEQNBR(*END) ENDOPT(*REWIND)>
Compressing TESTLIB/MYFILE(MYMBR) in TEXT mode
Add  TESTLIB/MYFILE/MYMBR  --  Deflating (67%)
Compressing TESTLIB/MYFILEGER(MYMBR) in TEXT mode
Add  TESTLIB/MYFILEGE.R/MYMBR  --  Deflating (10%)
Compressing TESTLIB/MYFILETEXT(MYMBR) in TEXT mode
Add  TESTLIB/MYFILETE.XT/MYMBR  --  Deflating (19%)
Compressing TESTLIB/MYFILE273(MYMBR) in TEXT mode
Add  TESTLIB/MYFILE27.3/MYMBR  --  Deflating (10%)
Archive <ARCHIVE_TEST02> created using tape device file PKW90051S/PKTAPEO1
SecureZIP Compressed 4 files in Archive PKW90051S/PKTAPEO1
SecureZIP Completed Successfully
```

The following command lists the files on the tape after several runs of PKZIP.

➔ **DSPTAP DEV(TAP01) OUTPUT(*PRINT)**

```
*...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8.
...+....9....+....0....+....1....+....2....+....3.
 5722SS1 V5R3M0 040528                TAPE VOLUME INFORMATION         PKZIP
 Device . . . . . :   TAP01                Volume . . . . . :   PKZIP
 Owner ID . . . . :                        Density  . . . . :   *QIC5010
 Type . . . . . :   *SL                    Code . . . . . :   *EBCDIC
                                                          Record
                File       Block  Recg Record Block  File       Mvol  Mvol
Date      Expiration System
Data File Label  Sequence   Format Tech Length Length Length     Ind   Sequence
Created   Date       Where Created

ARCHIVE_TEST01   0000000001 *F     P    32764  032764 0000000001       0000000001
11/08/05  11/08/05   IBMOS400
```

```
ARCHIVE_TEST02   0000000002 *F     P    32764  032764 0000000001       0000000001
11/08/05  *NONE      IBMOS400

ARCHIVE_TEST03   0000000003 *F     P    32764  032764 0000000001       0000000001
11/08/05  *NONE      IBMOS400

ARCHIVE_TEST04   0000000004 *F     P    32764  032764 0000000001       0000000001
11/08/05  *PERM      IBMOS400
                        * * * * *   E N D   O F   L I S T I N G   * * * * *
```

Sample - Extracting Files from an Archive Written Directly to Tape

PKUNZIP and PKZIP cannot use an archive on tape as an input archive. For PKZIP/PKUNZIP to work with an archive on tape, the archive must first be copied from tape to disk. This sample shows the steps

1. To extract an archive on tape, first identify the archive.

   ➔ **DSPTAP DEV(TAP01) OUTPUT(*PRINT)**

```
*...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8.
...+....9....+....0....+....1....+....2....+....3.
 5722SS1 V5R3M0 040528                TAPE VOLUME INFORMATION          PKZIP
 Device . . . . . :   TAP01               Volume . . . . . :   PKZIP
 Owner ID . . . . :                       Density  . . . . :   *QIC5010
 Type . . . . . :   *SL                   Code . . . . . :   *EBCDIC
                                                          Record
                File       Block  Recg Record Block  File       Mvol  Mvol
Date      Expiration System
Data File Label  Sequence   Format Tech Length Length Length     Ind   Sequence
Created   Date       Where Created

ARCHIVE_TEST01   0000000001 *F     P    32764  032764 0000000001       0000000001
11/08/05  11/08/05   IBMOS400

ARCHIVE_TEST02   0000000002 *F     P    32764  032764 0000000001       0000000001
11/08/05  *NONE      IBMOS400

ARCHIVE_TEST03   0000000003 *F     P    32764  032764 0000000001       0000000001
11/08/05  *NONE      IBMOS400

ARCHIVE_TEST04   0000000004 *F     P    32764  032764 0000000001       0000000001
11/08/05  *PERM      IBMOS400
                        * * * * *   E N D   O F   L I S T I N G   * * * * *
```

2. Create a file in a library where the file from tape will be copied. The record length must be the same as the record length on the tape file.

   ➔ **CRTPF FILE(MYLIB/TEMPZIP) RCDLEN(32764) FILETYPE(*DATA) MBR(*NONE) MAXMBRS(*NOMAX)**

3. Run the CPYFRMTAP command to copy the tape file to disk.

   ➔ **OVRTAPF FILE(PKTAPEO1) TOFILE(*LIBL/PKTAPEO1) DEV(TAP01) SEQNBR(2) LABEL(ARCHIVE_TEST02)**

   ➔ **CPYFRMTAP FROMFILE(*LIBL/PKTAPEO1) TOFILE(MYLIB/TEMPZIP) TOMBR(MYNBR1) FROMENDOPT(*REWIND) MBROPT(*ADD)**

```
File PKTAPEO1 overridden to PKTAPEO1 in PKZIPLIB.
RCDLEN, BLKLEN, RCDBLKFMT, BUFOFSET values assumed.
Member or label overridden to ARCHIVE_TEST01.
Member MYNBR1 added to file TEMPZIP in ATEST.
47 records copied from member ARCHIVE_TEST01.
```

4. Now run PKUNZIP/PKZIP using the file that was copied.

➔ **PKUNZIP ARCHIVE('MYLIB/TEMPZIP/MYNBR1') TYPE(*VIEW)**

```
Archive:  MYLIB/TEMPZIP(MYNBR1), 32764 bytes, 4 files, 1 Segment
Length  Method    Size  Ratio  Date   Time  CRC-32    Name
------- ------  ------- -----  ----   ----  ------    ----
  55271 Defl:S    18146  67% 10-26-05 07:48 13abbd41 TESTLIB/MYFILE/MYMBR
     48 Defl:S       43  10% 10-26-05 07:48 1b277b62 TESTLIB/MYFILEGE.R/MYMBR
    259 Defl:S      210  19% 10-26-05 07:48 b5dbf80c TESTLIB/MYFILETE.XT/MYMBR
     48 Defl:S       43  10% 10-26-05 07:48 1b277b62 TESTLIB/MYFILE27.3/MYMBR
-------         -------  ----                            -------
 55626            18442  67%                              4 files
SecureUNZIP  extracted     0 files
SecureUNZIP   Completed Successfully
```

A Performance Considerations

This appendix lists a few performance considerations when running ***PKZIPi***. Most performance related issues can be controlled by the PKZIP/PKUNZIP parameters. However, it should be noted that PKZIP data compression is CPU intensive by its very nature, and that PKZIP/PKUNZIP parameters can only help to a limited degree. Therefore, it should be expected that a *reasonable* amount of CPU resources will be needed for such operations.

Interactive Performance

When compressing large size files, PKZIP will sometimes use as much CPU resources as the system will allow. With this in mind, processing very large files may perform best as a submitted job. However, some iSeries environments have constraints on running interactive jobs. If those interactive jobs run for a long time and use a high amount of CPU resources, the system will slow down and may issue the message CPI1479 "Interactive activity approaching capacity of installed feature." In this case, review the details of this message. This usually means that the interactive systems are using more resources than the iSeries was configured to use.

Compression Type Performance

Selecting a compression method is one way to get the smallest compressed file with the relationship to the CPU usage and run times. Sometimes, to get the best results, you may have to run several tests with the data to balance the compression ratio to the length of the run time. Running with *MAX will usually get the best compression ratio but will also run the longest. In most of our test cases, *MAX would run 30%-40% longer than *NORMAL and might only gain less than 1% better ratio. This is why we recommend using SUPERFAST (the default) unless your testing implies otherwise.

To minimize the overhead needed to ZIP, the best thing (and the easiest) is to select a compression method other than *MAX. PKZIP's default compression method is SUPERFAST.

When using the compression method of Maximum, you are only compressing the data by another 1-8% over a job that might use the SUPERFAST compression method. The archive file size change is minimal. However, the time difference

between a maximum and a SUPERFAST job can be measured in hours if the file is big enough!

You may read more about the compression levels by prompting the Compression Level parameter (F1).

```
Compression Level  . . . . . *SUPERFAST    *FAST, *NORMAL, *MAX...
```

Data Type Selection

Getting the best performance from your iSeries machine with regards to a PKZIP job can truly depend on the parameters you have selected for the job. In many cases, the compressed size of a file depends on the type of data (Binary vs. Text), and the compression type selected. Text will usually compress more since it has a higher probability of repeated characters.

Knowing the target platform of the data will help you resolve how PKZIP is to treat the data during the compression process. However, PKZIP treatment of data defaults to *DETECT. *DETECT means that PKZIP will scan the data (up to 97% of the input file) to determine whether the data that it is going to compress should be treated as TEXT or BINARY. This can be an especially painful process if you are selecting large files for compression. However, to get around the *scanning* overhead, if you know you are sending the archive or ZIP file to a PC or to a UNIX machine, you know that the data will need to be converted to TEXT (or ASCII). Therefore, you should select file Types(*TEXT). If the data is targeted for another iSeries machine, then you should select *BINARY. *DETECT should only be used when you do not know the nature of the data.

You may read more about the data types by prompting the file Types parameter (F1).

```
File Types . . . . . .   *DETECT  *DETECT *TEXT *BINARY ....
```

Archive Placement (IFS or in a Library)

For best performance try to store the archives in the IFS. By placing the archive in the IFS instead of in a library/file reduces the overall CPU usage and in some cases can reduce the run times as much as 30%-40%.

It is recommended when using the ZIP process for large files that the ZIP archive be stored in the IFS. This method provides the best performance and makes the most efficient use of storage space for both ZIP archives and ZIP temporary files.

ZIP64 Processing Considerations

When processing very large files or high volumes of files, the processing characteristics of PKZIP may vary depending on the phase of processing involved. Some common processing phases and their run-time characteristics are:

- ZIP file selection: When selecting a very large number of files through many directories and/or libraries, the initial selection requires IO time and memory per file to analyze and manage each of the file's properties. The more files to

select, the more memory and initial startup overhead. Each site will have to discover their practical limits based on their environments and resources.

- Archive directory read processing: When updating an existing archive that contains a very large number of files, time and memory again are used to manage the archives and its directory. Or when using PKUNZIP to view the files, the more files in the archive, the more memory that is required and the more time that is involved when sorting the files in archive properties before displaying or printing the contents.
- Archive updating: When updating a large archive with large file sizes, there will be overhead to copy the files from the previous archive, before adding or updating new files to the archive. For example, if you have a 10 GB archive with 5 files that are each compressed, down to 2 GB, overhead will be required to copy the compressed files from the old archive to the new archive. This is another reason for storing the archive in the IFS, which can help reduce resources rather than storing the archive in a file in a library.
- When compressing large size files, PKZIP will sometimes use as much CPU as the system will allow. With this in mind, processing very large files may perform best as a submitted job. Some iSeries systems have constraints on running interactively, and if interactive jobs run a long time and use high amounts of CPU resources, their system will start slowing down and may issue the message CPI1479 "Interactive activity approaching capacity of installed feature." In this case they should review the details of this message, which usually means that their interactive systems are using more resources than the iSeries was configured to use.

Encryption Performance

When using advanced encryption versus no encryption, there will be a slight increase in the overall size of the archive that contains the AES overhead (approximately 300 bytes per file in archive). The increase in size will be same whether you use AES 128, AES 192, or AES 256.

AES 256 being the most secure encryption algorithm, will also consume the most CPU usage. AES 128 on average could use around 9% more CPU than running with no encryption. AES 256 averages about 3.4% more usage when compared with AES 128 (or around 12.5% versus no encryption).

Extended Attributes Selections

The extended attributes naturally contribute some overhead to the archive but it is minimal, unless you are compressing a database file in the QSYS library file system with the parameter DBSERVICE(*YES). This size then depends on the definitions of the database (fields, headings, etc), but also is very important in rebuilding a DB2 database where it does not exist.

These extended attributes can be stored in two places, called the local header and central header directories. ***PKZIP***^{i} 8.2 and other current PKWARE products now only use the extended attributes from the central directory. PKZIP for OS/400 5.5 required some of the attributes in the local header.

To help reduce the archive overhead the parameter EXTRAFLD in ***PKZIP**[i]* has been expanded to select where you want to store the attributes. By using EXTRAFLD(*Central), you reduce the size of each file in the archive by the size of the extended attributes. **Caution: If the attributes are required on another iSeries not running on 5.0 or above, use the option EXTRAFLD(*BOTH) or EXTRAFLD(*YES).**

B Examples

Example 1 - PKUNZIP Files to a New or Different Library

To extract the files in the archive to a new library. An example of the files in the archives are:

```
PKUNZIP ARCHIVE('atest/qz/tstchg')
Archive:  ATEST/QZ(TSTCHG)   88572 bytes   6 files
 Length  Method   Size  Ratio   Date   Time  CRC-32    Name
-------- ------ ------- -----   ----   ----  ------    ----
      259  Defl:F      178  01-16-01 08:24 b5dbf80c TESTLIB1/BEN/BEESON
   449664  Defl:F    48720  01-16-01 14:46 94c7506c
TESTLIB1/MYSPLFTM.P/AQZIP123.4X
   205693  Defl:F    29687  01-16-01 14:46 e2473ea4
TESTLIB1/MYSPLFTM.P/LISTDBOB.X
    27488  Defl:F     6771  01-16-01 14:46 c264817a TESTLIB1/MYSPLFTM.P/QPRINT1X
     3352  Defl:F      800  01-16-01 14:46 3e485445
TESTLIB1/MYSPLFTM.P/T4RSTLIB.XY
      256  Stored      256  01-16-01 15:22 29058c73 TESTLIB1/TEST1/HEX
 --------          -------  ---                            -------
   686712            86412                              6 files
```

To extract to a new library, use the keyword EXDIR and DROPPATH

```
PKUNZIP ARCHIVE('atest/qz/tstchg') TYPE(*EXTRACT) EXDIR(mynewlib) DROPPATH(*LIB)

     PKUNZIP Archive: ATEST/QZ(TSTCHG)
     Searching Archive ATEST/QZ(TSTCHG)  for files to extract
     Extracting file TESTLIB1/BEN/BEESON                       NOTE path
     Library MYNEWLIB created.                                 NOTE Library
created
     File BEN created in library MYNEWLIB.
     Member BEESON added to file BEN in MYNEWLIB.
     Member BEESON file BEN in MYNEWLIB changed.
     Inflating: MYNEWLIB/BEN(BEESON)     Text                 NOTE new path
     Extracting file TESTLIB1/MYSPLFTM.P/AQZIP123.4X
     File MYSPLFTMP created in library MYNEWLIB.
     Member AQZIP1234X added to file MYSPLFTMP in MYNEWLIB.
     Member AQZIP1234X file MYSPLFTMP in MYNEWLIB changed.
     Inflating: MYNEWLIB/MYSPLFTMP(AQZIP1234X)     Text
```

Since the library mynewlib did not exist, it was created.

Example 2 - CLP with Override for Stdout and Stderr to an OUTQ

The following is an example of overriding the PKZIP and PKUNZIP program output, and then redirecting the output to an OUTQ. This also provides an example of using mixed file systems, such as having the archive file in the IFS and selecting files from the QSYS library file system.

```
 ZIPEXPL01:   PGM        PARM(&OUTQ)
/* Program:   ZIPEXPL01            Example                      */
/*************************************************************/
/* Abstract: This is a example CL program has on parameter for   */
/*    the OUTQ for the processing of PKZIP for i5/OS.  If it is */
/*    *none or *NONE no overriding to QPRINT will take place.     */
/*    1. Will add the PKZIP for i5/OS Library to Library List  */
/*       If it is already part of LIBL then note so as to not    */
/*        remove at the end.                                     */
/*    2. Set the Current Library (only required if parameters of */
/*        PKZIP leaves out the Library and a default is needed)  */
/*    3. An example of setting the current directory is IFS      */
/*    4. Check input OUTQ.  If none keep processing              */
/*    5. Override the Stdout and Stderr to input outq            */
/*        This is where PKZIP will send messages when the        */
/*        MSGTYPE is (*PRINT) or (*BOTH).                        */
/*    6. Run Test01 of PKZIP                                     */
/*    7. Run Test02 of PKZIP with archive in IFS                 */
/*    8. Run Test03 of PKZIP with IFS system                     */
/*    9. Run Test04 of PKUNZIP  to view archive                  */
/*   10. If the PKZIP Library was not present at the beginning   */
/*       remove it from *LIBL.                                   */
/*                                                               */
/*************************************************************/

 OUTQ:       DCL        VAR(&OUTQ) TYPE(*CHAR) LEN(10)
 PKZIPLIB:   DCL        VAR(&PKZIPLIB) TYPE(*CHAR) LEN(10) +
                          VALUE(PKW90051S)
   /* if PKZIP library is in Libl do not remove it at the end */
 LIBLCHG:    DCL        VAR(&LIBLCHG) TYPE(*CHAR) LEN(1) VALUE('Y')
 CURLIB:     DCL        VAR(&CURLIB) TYPE(*CHAR) LEN(10) VALUE(MYLIB)
 ZIPDIR:     DCL        VAR(&ZIPDIR) TYPE(*CHAR) LEN(10) +
                          VALUE('/mydir')
     /* Add the PKZIP for i5/OS  library to library List  */
             ADDLIBLE   LIB(&PKZIPLIB)
             MONMSG     MSGID(CPF2103) EXEC(CHGVAR VAR(&LIBLCHG) +
                          VALUE('N'))
     /* Set Current Directory to MYLIB  */
                /*(not Required for this test Just an example)*/
             CHGCURLIB  CURLIB(&CURLIB)
     /* Set Current Directory to zip     */
                /*(not Required for this test Just an example)*/
              CD         DIR(&zipdir)

     /* Check input outq to see overides required   */
CHKOUTQ:     IF         COND((&OUTQ *EQ '*none') *OR (&OUTQ *EQ +
                          '*NONE')) THEN(GOTO CMDLBL(NOOUTQ))
  /* change Stdout and Stderr to my outq  */
             OVRPRTF    FILE(STDOUT) TOFILE(*LIBL/QSYSPRT) OUTQ(&OUTQ)
             OVRPRTF    FILE(STDERR) TOFILE(*LIBL/QSYSPRT) OUTQ(&OUTQ)
NOOUTQ:




     /* Test basic PKZIP  */
 TEST01:     PKZIP      ARCHIVE('ATEST/PKZ2(MYSL04)') +
                          FILES('TESTLIB/MYSPLF(*ALL)') +
                          EXCLUDE('TESTLIB/MYSPLF(Q*)')

     /* Test basic PKZIP with archive in IFS and print only messages  */
```

```
TEST02:     PKZIP      ARCHIVE('/mydir /tmpsave/itest01') +
                         FILES('TESTLIB/TEST') FILETYPE(*BINARY) +
                         TYPARCHFL(*IFS) MSGTYPE(*PRINT)

    /* Test PKZIP with all files in IFS  */
TEST03:     PKZIP      ARCHIVE('/mydir/tmpsave/itest02.zip') +
                         FILES('/mydir/test1/basetest') +
                         TYPARCHFL(*IFS) TYPFL2ZP(*IFS)

    /* Test PKUNZIP view of archive from test02  */
TEST04:     PKUNZIP    ARCHIVE('/mydir/tmpsave/itest01') +
                         TYPARCHFL(*IFS)  MSGTYPE(*PRINT)

    /* If PKZIP Library was added to LIBL then remove it */
ENDPGM:     IF         COND(&LIBLCHG *EQ 'Y') THEN(RMVLIBLE +
                         LIB(&PKZIPLIB))
            ENDPGM
```

Example 3 - Creating an Archive in Personal Folders (QDLS)

The following is an example of creating and processing the archive in the Document library Services file system (QDLS). First, assume a folder in QDLS with a name of MYFOLDER where the archives will be stored. To view the folders, issue the command WRKLNK '/QDLS/*' (you could use WRKDOC and WRKFLR, but WRKLNK is better to use since PKZIP will be using /QDLS).

```
                          Work with Object Links

Directory  . . . . :   /qdls

Type options, press Enter.
  3=Copy   4=Remove   5=Next level   7=Rename   8=Display attribu
  11=Change current directory ...

Opt   Object link            Type             Attribute    Text
      .                      FLR
      ..                     FLR
      MYFOLDER               FLR
      QBKBOOKS               FLR
```

Run the PKZIP command:

PKZIP ARCHIVE('/QDLS/MYFOLDER/MYARCH1.ZIP') FILES('testlib/ben') TYPARCHFL(*IFS)

The suffix .ZIP was added to help identify the file as an archive file.

```
Scanning files for match  ...
Found  1 matching files
Compressing TESTLIB/BEN(BEESON) in TEXT mode
Add  TESTLIB/BEN/BEESON  --   Deflating (32%)
PKZIP Compressed 1 files in Archive /QDLS/MYFOLDER/MYARCH1.ZIP
PKZIP Completed Successfully
Press ENTER to end terminal session.
```

To see the file in the folders, run WRKLNK '/QDLS/MYFOLDER/*'

```
                          Work with Object Links

Directory  . . . . :   /QDLS/MYFOLDER

Type options, press Enter.
  3=Copy   4=Remove   5=Next level   7=Rename   8=Display attributes
  11=Change current directory ...

Opt   Object link            Type             Attribute    Text
      .                      FLR
      ..                     FLR
      MYARCH1.ZIP            DOC
```

Next, to view the contents, run:

PKUNZIP ARCHIVE('/QDLS/MYFOLDER/MYARCH1.ZIP') TYPARCHFL(*IFS)

```
Archive:  /QDLS/MYFOLDER/MYARCH1.ZIP   551 bytes   1 file
  Length  Method    Size  Ratio  Date   Time  CRC-32    Name
 -------- ------  ------- -----  ----   ----  ------    ----
      259 Defl:F      177  32% 11-27-00 15:32 b5dbf80c TESTLIB/BEN/BEESON
 --------         -------  ----                          -------
     259              177  32%                           1 file
PKUNZIP extracted     0 files
PKUNZIP Completed Successfully
Press ENTER to end terminal session.
```

Example 4 - Processing Archive on a CD (QOPT)

The following is an example of processing an archive that exists on a CD and using PKUNZIP to view or extract. Because the archive file is on a CD, and the file system QOPT controls the CD, this archive basically exists in the IFS.

First, check and ensure the archive is on the CD by doing a WRKLNK (you can use WRKOPTDIR, but using WRKLNK will show the actual paths required). Remember, the volume of the CD is also a directory in QOPT file system. If the file names are longer than eight characters, the file name will be changed, much like you see in DOS systems. It will contain a tilda (~) followed by a number for files found with excessive name lengths.

WRKLNK '/QOPT/*'

```
                          Work with Object Links

Directory  . . . . :   /QOPT

Type options, press Enter.
  3=Copy   4=Remove   5=Next level   7=Rename   8=Display attributes
  11=Change current directory ...

Opt   Object link            Type             Attribute    Text
      MYTESTLABEL            DDIR
```

The above screen shows that the volume label of the CD is “MYTESTLABEL”. Using the “5” for the next level option, you can navigate through the directories. You will then see the files and directories on the root of the CD. For example:

```
                          Work with Object Links

 Directory  . . . . :   /QOPT/MYTESTLABEL

 Type options, press Enter.
   3=Copy   4=Remove   5=Next level   7=Rename   8=Display attributes
   11=Change current directory ...

 Opt   Object link            Type             Attribute    Text
       ARCHIVE.ZIP            DSTMF
       GZIPPW.GAR             DSTMF
       OS_400~3.DOC           DSTMF
       PKZCVT~2.DOC           DSTMF
       PKW90~1.SAV           DSTMF
       PKW90~1.ZIP           DSTMF
```

To view the archive PKW90~1.ZIP (which is really the long name PKW90051S.ZIP) contents, use PKUNZIP with *VIEW.

Use the command:

PKUNZIP ARCHIVE('/QOPT/MYTESTLABEL/PKW90~1.ZIP') TYPARCHFL(*IFS) TYPE(*VIEW)

```
Archive:  /QOPT/MYTESTLABEL/PKW90~1.ZIP   1373026 bytes   1 file
  Length  Method    Size  Ratio  Date   Time  CRC-32    Name
 -------- ------  ------- -----  ----   ----  ------    ----
  6044544 Defl:N  1372902  77% 08-16-01 21:17 d73f09cf PKW90051S.sav
 --------         -------  ----                          -------
 6044544          1372902  77%                           1 file
PKUNZIP extracted     0 files
PKUNZIP Completed Successfully
```

Example 5 - Compressing files from a CD (QOPT)

Using the document (.DOC) files on the CD shown in Example 4, we can compress the files and store them in the archive in my archive library ATEST under the file V509 archives with an archive file member named CDTEST01.

PKZIP ARCHIVE('atest/v509/cdtest01')
FILES('/QOPT/MYTESTLABEL/OS_400~3.DOC'
'/QOPT/MYTESTLABEL/PKZCVT~2.DOC') TYPFL2ZP(*IFS)

```
Scanning files for match  ...
Found  2 matching files
Compressing /QOPT/MYTESTLABEL/OS_400~3.DOC in BINARY mode
Add  /QOPT/MYTESTLABEL/OS_400~3.DOC  --   Deflating (77%)
Compressing /QOPT/MYTESTLABEL/PKZCVT~2.DOC in BINARY mode
Add  /QOPT/MYTESTLABEL/PKZCVT~2.DOC  --   Deflating (79%)
PKZIP Compressed 2 files in Archive ATEST/V509(CDTEST01)
PKZIP Completed Successfully
```

Because you would not be able to extract them to the CD, you may want to use the parameter STOREPATH(*NO) so that only file names OS_400~3.DOC and PKZCVT~2.DOC are stored in the archive.

Example 6 - Compressing CL with MSG Checking

The following brief example demonstrates using PKZIP in a CL passing the archive's library, file, and member as variables and then monitoring for errors from the PKZIP run.

```
ZIPEXPL03:  PGM        PARM(&ZIPLIB &ZIPFILE &ZIPMBR)

/* Program:   ZIPEXPL03            Example                      */
/*************************************************************/
/* Abstract: This is a example CL program that has 3 paramters   */
/*    that specifies the archive's Library, File and Member.     */
/*    1. Will add the PKZIP for i5/OS  Library to Library List */
/*       If it is already part of LIBL then note so as to not    */
/*        remove at the end.                                     */
/*    2. Build the archive file namefor PKZIP by concatenating   */
/*       the inputted library, file and member names.            */
/*    3. Compress all files in TESTLIB with PKZIP Command        */
/*    4. Monitor for error messages from PKZIP.                  */
/*        If errors send message.                                */
/*    5. If the PKZIP Library was not present at the beginning   */
/*       remove it from *LIBL.                                   */
/*                                                               */
/*************************************************************/
 /* &PKZIPLIB contains the current PKZIP library            */
            DCL        VAR(&PKZIPLIB) TYPE(*CHAR) LEN(10) +
                         VALUE(PKW90051S)
 /* if PKZIP library is in Libl do not remove it at the end */
            DCL        VAR(&LIBLCHG) TYPE(*CHAR) LEN(1) VALUE('Y')
 /* &ZIPLIB is Library where archive will be stored  */
            DCL        VAR(&ZIPLIB)  TYPE(*CHAR) LEN(10)
 /* &ZIPFILE is File for the archive                */
            DCL        VAR(&ZIPFILE) TYPE(*CHAR) LEN(10)
 /* &ZIPMBR is Member of the archive file           */
            DCL        VAR(&ZIPMBR)  TYPE(*CHAR) LEN(10)
 /* Archive file for PKZIP built with concatenation  */
            DCL        VAR(&ZARCHF)  TYPE(*CHAR) LEN(36)
 /* Add the PKZIP for i5/OS  library to library List  */
            ADDLIBLE   LIB(&PKZIPLIB)
            MONMSG     MSGID(CPF2103) EXEC(CHGVAR VAR(&LIBLCHG) +
                         VALUE('N'))

 /*Concatenate the libraries, files and members for PKZIP of the archive*/
            CHGVAR     VAR(&ZARCHF) VALUE(&ZIPLIB *TCAT '/' *TCAT +
                         &ZIPFILE *TCAT '/' *TCAT &ZIPMBR)

 /* Compress all files in the library TESTLIB  and     */
 /* store them in the archive specified in the         */
 /* calling of the CLP program                         */
 /* If messages AQZ0022 "PKZIP Completed with Errors." */
 /* or AQZ0012 "PKZIP ending with Nothing to do"       */
 /* are returned                                       */
 /* from PKZIP, send message indicating an error       */
 /* Occured.                                           */
            PKZIP      ARCHIVE(&ZARCHF) FILES('TESTLIB/*all(*all)') +
                         COMPRESS(*NORMAL) ARCHTEXT('This the text +
                         of the archive for example 3')
            MONMSG     MSGID(AQZ0022) EXEC(SNDPGMMSG MSG('PKZIP +
                         Ended with MONMSG for AQZ0022'))
            MONMSG     MSGID(AQZ0012) EXEC(SNDPGMMSG MSG('PKZIP +
                         Ended with MONMSG for AQZ0012'))

    /* If PKZIP Library was added to LIBL then remove it */
 ENDPGM:    IF         COND(&LIBLCHG *EQ 'Y') THEN(RMVLIBLE +
                         LIB(&PKZIPLIB))
 EOJ:       ENDPGM
```

Example 7 – Compressing Spool Files Samples

The following are several samples demonstrating the selection of spool file for compression.

Sample 1: Select a specific spool file (MYSPLFFILE) for the specific job (jobname-WSSSPL, User-WSS and job number 11) in all output queues (the default of *SFQUEUE*) and convert the spool file to a PDF format *SFTARGET(*PDFLETTER)* to fit a letter format. store the archive in the IFS with *TYPARCHFL(*IFS)* .

```
PKZSPOOL ARCHIVE('/yourpath/bills/splftest01.zip') TYPARCHFL(*IFS)
         SPLFILE(MYSPLFILE) SFUSER(*ALL) SFJOBNAM(11/WSS/WSSSPL)
         SFTARGET(*PDFLETTER) SFTGFILE(*GEN1P)
```

Sample 2: Select all spool files belonging to users WSS and TAIT (*SPLUSERID*) that resides in the OUTQ QPRINTS (*SFQUEUE*) and compress them as spool files with *SFTARGET(*SPLF)*. This might be done to save the spool files for later review since this OUTQ is purged on a regular basis.

```
PKZSPOOL ARCHIVE('/yourpath/bills/splftest02.zip') TYPARCHFL(*IFS)
         SFUSER(WSS TAIT) SFQUEUE(QPRINTS) SFTARGET(*SPLF) SFTGFILE(*GEN1)
```

Sample 3: Using the archive from Sample 2, we want to restore or extract the spool files in order to print them again. Except in this case we want them to belong to the user MAS with *SPLUSERID* and place the spool files in the OUTQ MASQ (*SFQUEUE*) located in the library DEVPLIB.

```
PKUNZIP ARCHIVE('/yourpath/bills/splftest02.zip') TYPARCHFL(*IFS)
        TYPE(*EXTRACT) SPLUSRID(MAS) SFQUEUE(DEVPLIB/MASQ)
```

Sample 4: Select the spool file QPRINTS (*SPLFILE*), spool file number 17 (*SPLNBR*), user MAS (*SFUSER*) and convert the file to a TEXT file with *SFTARGET(*TEXTFC).* In this case, the file is needed to read into a PC program and the user wants the ANSI control characters in position 1 of each line.

```
PKZSPOOL ARCHIVE('/yourpath/bills/splftest04.zip')  TYPARCHFL(*IFS)
    SPLFILE(QPRINTS) SFUSER(MAS) SPLNBR(17)
    SFTARGET(*TEXTFC) SFTGFILE(*GEN1P)
```

Sample 5: Now we want to extract the text file created in Sample 4 to one of our shared drives areas ('/yourpath/PCFILES') that our PCs can access. In this case the normal extraction would identify the file as a text file and would convert it to EBCDIC. Since the file will be used by a PC program that is expecting the data to be in ASCII, we will have to extract the file as binary since the internal file is already in ASCII. By specifying FILETYPE(*BINARY), this ensures that no translation of the data takes place.

```
PKUNZIP ARCHIVE('/yourpath/bills/splftest04.zip') TYPARCHFL(*IFS) TYPFL2ZP(*IFS)
TYPE(*EXTRACT) FILETYPE(*BINARY)
EXDIR('/yourpath/PCFILES') DROPPATH(*ALL)
```

Example 8 – PKZSPOOL The Last Spool File of Current Job

The following brief CLP example demonstrates using PKZSPOOL to compress to a PDF, only the last spool file that was written out by the current job.

```
 ZIPEXPL07:  PGM
/* Program:   ZIPEXPL07            Example                      */
/*************************************************************/
/* Abstract: This is an example CL program that perform several */
/* task that prints reports.  Then compresses only the LAST     */
/* spool file created to a PDF file in a archive                */
/*                                                               */
/*************************************************************/


/* display the properties of the files start with "i" in QIBM folder   */
             DSPLNK     OBJ('/QIBM/i*') OUTPUT(*PRINT) +
                          DETAIL(*EXTENDED) DSPOPT(*ALL)


 /* display all libraries that start with Q and print              */
             DSPOBJD    OBJ(*LIBL/Q*) OBJTYPE(*LIB) DETAIL(*BASIC) +
                          OUTPUT(*PRINT)



/* Compress the ONLY the last spool file created to a PDF file       */
             PKZSPOOL   ARCHIVE('/yourpath/bills/PKZdata1.zip') +
                          SFJOBNAM(*) SPLNBR(*LAST) +
                          SFTARGET(*PDFLETTER) SFTGFILE(*GEN1P) +
                          TYPARCHFL(*IFS)
 ENDOFJOB:   ENDPGM
```

Example 9 - CL to Compress All Spool Files for a Job to a PDF

The following brief example demonstrates using PKZIP in a CL to pass the archive's library, file, and member as variables and then monitor for errors from the PKZIP run.

```
 ZIPEXPL08:  PGM
/* Program:   ZIPEXPL08            Example                      */
/*************************************************************/
/* Abstract: This is an example CL program that perform several */
/* task that prints reports.  Then submits a job at the end of   */
/* the job that will compress all spool files to PDF files. The  */
/* reason the job was submitted was to also compress the job log */
/* to a PDF                                                      */
/*                                                               */
/*************************************************************/
         /*    Current Job Name     */
             DCL        VAR(&MYJOBNM) TYPE(*CHAR) LEN(10) VALUE(' ')
         /*    Current Job User     */
             DCL        VAR(&MYJUSER) TYPE(*CHAR) LEN(10) VALUE(' ')
         /*    Current Job Number   */
             DCL        VAR(&MYJNBR) TYPE(*CHAR) LEN(10) +
                          VALUE('000000')
/* retrieve the job name, user, and job number for later use       */
             RTVJOBA    JOB(&MYJOBNM) USER(&MYJUSER) NBR(&MYJNBR)
/* this job to get a full job log                                  */
             CHGJOB     LOG(4 00 *SECLVL) LOGCLPGM(*YES)

/* display the properties of the files start with c in my folder   */
             DSPLNK     OBJ('/yourpath/bills/c*') OUTPUT(*PRINT) +
```

```
                          DETAIL(*EXTENDED) DSPOPT(*ALL)
             DSPAUT OBJ('/yourpath/BILLS') OUTPUT(*PRINT)


/* create an archive with one file                                    */
             PKZIP      ARCHIVE('/yourpath/bills/PKZtest1.zip') +
                        FILES('/yourpath/bills/chartest2.zip') +
                        TYPARCHFL(*IFS) TYPFL2ZP(*IFS) STOREPATH(*NO)

/*  display detail attributes for the new archive created  */
             DSPLNK     OBJ('/yourpath/bills/PKZtest1*') OUTPUT(*PRINT) +
                          DETAIL(*EXTENDED) DSPOPT(*ALL)

/* submit a job to create all spool Files including the job log into a */
/*   PDF file and place them in  archive  PKZdata1                     */
             SBMJOB     CMD(PKZSPOOL +
                          ARCHIVE('/yourpath/bills/PKZdata1.zip') +
                          SFJOBNAM(&MYJNBR/&MYJUSER/&MYJOBNM) +
                          SFTARGET(*PDFLETTER) SFTGFILE(*GEN1P) +
                          TYPARCHFL(*IFS)) JOB(&MYJOBNM) +
                          INLLIBL(*CURRENT)
 ENDOFJOB:   ENDPGM
```

Example 10 - Compress File with Public Digital Certificates

Requires SecureZIP

The first ZIP test will use both of the public certificates and 256-bit AES algorithm to encrypt and compress one file to an archive in the folder that was created earlier. This test will use the *MBRSET and *FILE types for the selection of the certificates.

➔PKZIP ARCHIVE('/myroot/pkware/CStore/Testzips/TestC01.zip')
FILES('PKW90051s/$CONTACT') ADVCRYPT(AES256)
TYPARCHFL(*IFS) TYPFL2ZP(*DB)
ENTPREC((*MBRSET pktestdb3.crt)
(*FILE '/myroot/pkware/CStore/Public/pktestdb4.crt'))

```
Scanning files in *DB for match  ...
Total Recipients processed 2
Archive Recipient List:
CN=PKWARE Test4 EMail=PKTESTDB4@nowhere.com
CN=PKWARE Test3 EMail=PKTESTDB3@nowhere.com
Found  1 matching files
Compressing PKW90051S/$CONTACT($CONTACT) in TEXT mode
Add  PKW90051.S/$CONTACT/$CONTACT  --  Deflating (80%)  encrypt(BSAFE AES 256
Key)
SecureZIP Compressed 1 files in Archive /myroot/pkware/CStore/Testzips/TestC0
1.zip
SecureZIP Completed Successfully
```

The second ZIP test will use both of the public certificates and AES256 algorithm to encrypt and compress one file to an archive in the folder. This test will use the *DB with email and common name for the selection of the certificates.

➔PKZIP ARCHIVE('/myroot/pkware/CStore/Testzips/TestC02.zip')
FILES('PKW90051s/$CONTACT') ADVCRYPT(AES256)
TYPARCHFL(*IFS) TYPFL2ZP(*DB)
ENTPREC((*DB 'EM=PKTESTDB3@nowhere.com')
(*DB 'CN=PKWARE Test4'))

```
Scanning files in *DB for match  ...
Total Recipients processed 2
Archive Recipient List:
CN=PKWARE Test4 EMail=PKTESTDB4@nowhere.com
CN=PKWARE Test3 EMail=PKTESTDB3@nowhere.com
Found  1 matching files
Compressing PKW90051S/$CONTACT($CONTACT) in TEXT mode
Updating:PKW90051.S/$CONTACT/$CONTACT    Deflating (80%)  encrypt(BSAFE AES 2
56Key)
SecureZIP Compressed 1 files in Archive /myroot/pkware/CStore/Testzips/TestC0
2.zip
SecureZIP Completed Successfully
```

Example 11 - Decrypting File with Private Key Certificates

Requires SecureZIP

In order to decrypt the file you will need to provide at least one valid private certificate with the passphrase that matches a recipient on the archive.

```
➔PKUNZIP ARCHIVE('/myroot/pkware/CStore/Testzips/TestC01.zip')
  TYPE(*TEST)
  TYPARCHFL(*IFS)
  ENTPREC((*DB 'CN=PKWARE Test4' ('PKWARE')))
```

Example 12 - Sign Files and Archive with Private Keys

Requires SecureZIP

Create an archive and sign the files in the archive by two signers and then sign the archive directory. Note that signing requires the private key.

```
➔PKZIP ARCHIVE('/myroot/pkware/CStore/Testzips/TestC03.zip')
  FILES('PKW90051s/$CONTACT') ADVCRYPT(AES256)
  TYPARCHFL(*IFS) TYPFL2ZP(*DB)
  ENTPREC((*DB 'EM=PKTESTDB3@nowhere.com')
  (*DB 'CN=PKWARE Test4'))
  SIGNERS((*FILE *MBRSET 'pktestdb3.p12' (PKWARE) )
  (*ALL *MBRSET 'pktestdb4.p12' (PKWARE)) )
```

```
Scanning files in *DB for match  ...
2  Encryption Recipients processed
Encryption Recipients List:
--CN=PKWARE Test3 EMail=PKTESTDB3@nowhere.com
--CN=PKWARE Test4 EMail=PKTESTDB4@nowhere.com
2  File Signers processed
File Signers List:
--CN=PKWARE Test4 EMail=PKTESTDB4@nowhere.com
--CN=PKWARE Test3 EMail=PKTESTDB3@nowhere.com
1  Archive Signer processed
Archive Signer List:
--CN=PKWARE Test4 EMail=PKTESTDB4@nowhere.com
Found  1 matching files
Compressing PKW900XXS/$CONTACT($CONTACT) in TEXT mode
Add  PKW900XX.S/$CONTACT/$CONTACT  --  Deflating (80%)  encrypt(BSAFE AES 256Key)
```

```
SecureZIP Compressed 1 files in Archive
/myroot/pkware/CStore/Testzips/TestC03.zip
SecureZIP Completed Successfully
```

Example 13 - Authenticate Signed Files and Archive

Requires SecureZIP

When doing a basic view of the newly signed archive, notice that only the archive directory signatures are validated. To validate the signature of the files would require a TYPE(*TEST).

➔ **PKUNZIP ARCHIVE('/myroot/pkware/CStore/Testzips/TestC03.zip') TYPE(*VIEW) TYPARCHFL(*IFS) TYPFL2ZP(*DB) AUTHCHK((*ARCHIVE *MBRSET 'pktestdb4.crt')) AUTHPOL(*WARN (*ALL))**

```
1  Archive Signer processed
Archive:  /myroot/pkware/CStore/Testzips/TestC03.zip   7053 bytes   1 file

  Length  Method    Size  Ratio  Date   Time  CRC-32    Name
 -------- ------  ------- -----  ----   ----  ------    ----
     5451 Defl:F     1702  69% 01-11-05 13:34 f091572d
!PKW900XX.S/$CONTACT/$CONTACT
 --------         -------  ----                            -------
    5451             1702  69%                             1 file
Archive has been Digitally Signed.
Archive was signed by "PKWARE Test4" and verified
SecureUNZIP  extracted     0 files
SecureUNZIP   Completed Successfully
```

When the files in the archive are tested or extracted, the archive signature is validated first and then, after each file has been tested, the file's signatures are tested. If no AUTHCHK parameter is entered, all signatures are validated.

➔ **PKUNZIP ARCHIVE('/myroot/pkware/CStore/Testzips/TestC03.zip') TYPE(*TEST) TYPARCHFL(*IFS) TYPFL2ZP(*DB) ENTPREC((*DB 'CN=PKWARE Test3' 'PKWARE'))**

```
1  Encryption Recipients processed
UNZIP Archive: /myroot/pkware/CStore/Testzips/TestC03.zip
Searching Archive /myroot/pkware/CStore/Testzips/TestC03.zip  for files to
extract
Archive was signed by "PKWARE Test4" and verified
Testing: PKW900XX.S/$CONTACT/$CONTACT
File was signed by "PKWARE Test4" and verified
File was signed by "PKWARE Test3" and verified
PKW900XX.S/$CONTACT/$CONTACT tested OK
No errors detected in compressed data of
/myroot/pkware/CStore/Testzips/TestC03.zip.
SecureUNZIP   Completed Successfully
```

Example 14 - Encryption Using LDAP Search for Recipients

Requires SecureZIP

➔ **PKZIP ARCHIVE('/yourpath/aVXXTest/test013.zip')**
FILES('/yourpath/aVXXTest/recp/Test cases.txt')
TYPARCHFL(*IFS) TYPFL2ZP(*IFS) TYPLISTFL(*IFS)
STOREPATH(*NO) ADVCRYPT(AES256)
ENTPREC((*LDAP 'EM=bill.Somebody@pkware.com' *N *RQD))

Displayed output from example.

```
Scanning files in *IFS for match  ...
Total Recipients processed 2
Archive Recipient List:
CN=PKWCADMIN EMail=none
CN=William Somebody EMail=bill.Somebody@pkware.com
Found  1 matching files
Compressing /yourpath/aVXXTest/recp/Test cases.txt in BINARY mode
Add  Test cases.txt  --  Deflating (81%)  encrypt(BSAFE AES 256Key)
SecureZIP Compressed 1 files in Archive /yourpath/aVXXTest/test013.zip
SecureZIP Completed Successfully
```

-------- new begin

Example 15 - Using IPSRA for Multiple Libraries with 1Step2Tape

Requires SecureZIP

Example 15 source member ZIPEXPL51 can be found in the QCLSRC file of the distributed library. This example demonstrates how to save multiple libraries directly to tape utilizing iPSRA and 1Step2Tape features. It simulates the SAVLIB command with multiple libraries by writing each library's saved archive directly to tape using the tape label name as the library name. It also creates a LOG file for each saved library as the member name. This file contains the contents of the save operation.

This example can be used to simulate using a *NONSYS with a SAVLIB command.

This CL displays of library type objects, with output going to a temporary file. The CL then loops through the output file, performs a PKZIP with iPSRA options for each selected library, and creates the archive directly to tape using the tape label for the library name. Encryption could be added to make the save files secure.

Noted in italics is code that should be changed or removed if the CL is used.

For the CLP to compile, the file DSPOBJD01 must exist in QTEMP. You can create the file there by executing the following command prior to compiling:

➔ **DSPOBJD OBJ(*IBM) OBJTYPE(*LIB) DETAIL(*BASIC)**
OUTPUT(*OUTFILE) OUTFILE(QTEMP/DSPOBJD01)
OUTMBR(*FIRST *REPLACE)

The CL has three parameters:

- A tape device that will be used to create an archive directly to tape
- The library where the log will reside
- The name of the save log file

Example call

➔ **CALL ZIPEXPL51 ('TAP01' 'ATESTLIB' 'ZSLOG01')**

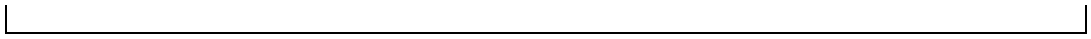
This call creates all archives on tape TAP01 and builds the log members in file ZSLOG01 on library ATESTLIB. If the file ZSLOG01 does not exist, it is created.

Example ZIPEXPL51 Source Listing

```
            PGM        PARM(&TAPDEVN &LOGLIB &LOGFIL)
/* Program:   ZIPEXPL51            Example Only                  */
/**************************************************************/
/*  This CLP contains programming source code for your        */
/*  consideration.  This example has not been thoroughly tested  */
/*  under all conditions.  PKWARE, Inc, therefore, cannot     */
/*  guarantee or imply reliability or functionality of this   */
/*  program.  The program contained herein is provided to     */
/*  you "AS IS".                                              */
/**************************************************************/
/* Abstract:This example CL demonstrates one way to save all or  */
/*    selected libraries directly to tape utilizing PKZIP's   */
/*    1-Step-2Tape and iPSRA solutions.                       */
/*    This example CL program has 3 parameters.               */
/*    Parameter 1 specifies a tape device that will be used to   */
/*    create an archive directly to tape.                     */
/*    Parameter 2 specifies the library where the log will reside*/
/*    Parameter 3 specifies the name of the save log file.    */
/*                                                            */
/*    1. Override any tape attributes requires such as for    */
/*         maximum optimum tape size.                         */
/*    2. Do DSPOBJD with selection of Objects for type *LIB   */
/*        This can be modified for selections.                */
/*    3. Loop thru the outfile of the DSPOBJD for each library   */
/*       to process, when message CPF0864 responds then we are   */
/*       at end of file.                                      */
/*    3.1. Check library and bypass if required               */
/*          This version has hard coding selection so as not  */
/*          select all libraries of an environment.           */
/*    3.2. Change variables and build the SAVLIB command      */
/*    3.3. Issue the PKZIP command for iPSRA SAVLIB direct to    */
/*         tape.  Add the file to the end of the tape.        */
/*         Use *LEAVE for performance.                        */
/*         If this requires encryption add appropriate controls. */
/*                                                            */
/*    4. Optional - Print list of files on tape with DSPTAP   */
/*                                                            */
/*    5. Send completion message                              */
/*                                                            */
/*    This program includes some hard coding for testing      */
/*      purpose only.                                         */
/*                                                            */
/*  To compile this CLP, you will need to have the file       */
/*  DSPOBJD01 from the DSPOBJD command. Run the following     */
/*  prior to compiling this CL.                               */
/*  ==>DSPOBJD OBJ(QUSRTEMP) OBJTYPE(*LIB) DETAIL(*BASIC)     */
/*             OUTPUT(*OUTFILE) OUTFILE(QTEMP/DSPOBJD01)      */
/*             OUTMBR(*FIRST *REPLACE)                        */
/*                                                            */
/* Example Call:                                              */
```

```
/* call zipexpl51 ('TAP01' 'PKW52151S' 'ZSLOG01')                */
/*****************************************************************/

/* *****************declare variables*********************    */
/* &TAPDEVN The inputted Tape device that will be written on  */
             DCL        VAR(&TAPDEVN) TYPE(*CHAR) LEN(10)

/* Inputted Library and file name where the save log files exist */
/*      will be stored                                           */
             DCL        VAR(&LOGLIB) TYPE(*CHAR) LEN(10)
             DCL        VAR(&LOGFIL) TYPE(*CHAR) LEN(10)

/* &USELIB is the Library Variable that will be processed        */
             DCL        VAR(&USELIB) TYPE(*CHAR) LEN(10)

             DCL        VAR(&DEVNM)  TYPE(*CHAR) LEN(14)
/* &SAVCMD is where the SAVLIB command will be built             */
/* the model would be:                                           */
/* -SAVLIB LIB(&USELIB) DEV(&DEVNM) OUTPUT(*OUTFILE)             */
/*     OUTFILE(&LOGLIB/&LOGFIL) OUTMBR(&USELIB *REPLACE)         */
             DCL        VAR(&SAVCMD) TYPE(*CHAR) LEN(100)
/*  SAVBLDx are variable to build SAVLIB command                 */
             DCL        VAR(&SAVBLD1) TYPE(*CHAR) LEN(12) +
                         VALUE('-SAVLIB LIB(')
             DCL        VAR(&SAVBLD2) TYPE(*CHAR) LEN(6) +
                         VALUE(') DEV(')
             DCL        VAR(&SAVBLD3) TYPE(*CHAR) LEN(27) +
                         VALUE(') OUTPUT(*OUTFILE) OUTFILE(')
             DCL        VAR(&SAVBLD4) TYPE(*CHAR) LEN(9) +
                         VALUE(') OUTMBR(')
             DCL        VAR(&SAVBLD5) TYPE(*CHAR) LEN(10) +
                         VALUE(' *REPLACE)')

/*  MSGOUT1 are variable to build SAVLIB command                 */
             DCL        VAR(&MSGOUT1) TYPE(*CHAR) LEN(47) +
                          VALUE('lllllllll Is now being saved +
                          ..............')

/*  LIBCNT is the number of libraries were processed           */
             DCL        VAR(&LIBCNT) TYPE(*DEC) LEN(5) VALUE(0)
             DCL        VAR(&MSGOUT2) TYPE(*CHAR) LEN(32) +
                          VALUE('Number of Libraries Saved were ')
             DCL        VAR(&TSTNOCHAR) TYPE(*CHAR) LEN(5) +
                         /* holds "-99999" to "99999" */

/* &TSEQ is the Tape Sequence Variable that will be processed    */
/*      Could change &TSEQ each time with a specific sequence    */
/*      number if desired                                        */
             DCL        VAR(&TSEQ) TYPE(*CHAR) LEN(10) VALUE('*END')

/* &TDISP is the Tape disposition Variable that will be processed */
/*      When processing mutli tape files *LEAVE will perform     */
/*      better than rewind.  But if you know the last file you   */
/*      can change &TDISP to *REWIND or *UNLOAD                  */
             DCL        VAR(&TDISP) TYPE(*CHAR) LEN(10) VALUE('*LEAVE')

/* Declare file containing results from the display objects command for LIB */
             DCLF       FILE(QTEMP/DSPOBJD01)

/* Step  1. Override any tape attributes requires such as for   */
/*          maximum optimum tape size.                          */
             OVRTAPF    FILE(PKTAPE01) DEV(&TAPDEVN)
/*                      BLKLEN(262112) RCDBLKFMT(*FB)           */

/* Step  2. Do DSPOBJD with selection of Objects for type LIB    */
/*       Doing a SAVLIB LIB(*IBM) and then doing a SAVLIB       */
/*       LIB(*ALLUSR) saves the same libraries as a SAVLIB      */
/*       LIB(*NONSYS)                                           */
             DSPOBJD    OBJ(*IBM) OBJTYPE(*LIB) DETAIL(*BASIC)     +
                          OUTPUT(*OUTFILE) OUTFILE(QTEMP/DSPOBJD01) +
                          OUTMBR(*FIRST *REPLACE)
```

```
            DSPOBJD    OBJ(*ALLUSR) OBJTYPE(*LIB) DETAIL(*BASIC)   +
                         OUTPUT(*OUTFILE) OUTFILE(QTEMP/DSPOBJD01) +
                         OUTMBR(*FIRST *ADD)


/* Step  3. Loop thru the outfile of the DSPOBJD for each library*/
/*       to process, when message CPF0864 responds then we are at*/
/*       end of file.                                            */
/*                                                               */
 loop:       RCVF
             MONMSG     MSGID(CPF0864) EXEC(GOTO CMDLBL(LASTSTEP))
  /* set the library to process   */
             CHGVAR     VAR(&USELIB) VALUE(&ODOBNM)

/* Step  3.1. Check library and bypass if required               */
/*          This version has hard coding selection so as not     */
/*          select all libraries of an environment.              */
/* add logic to by pass any library here                         */
             IF         COND(&ODODMN *EQ '*U') +
                         THEN(GOTO CMDLBL(loop))
  /* testing only select only library that start with PKW0  */
             IF         COND(%SST(&USELIB 1 4) *NE 'PKW0')     +
                         THEN(GOTO CMDLBL(LOOP))


/* Step  3.2. Change variables and build the SAVLIB command        */
/* Build the &DEVNM whihc will be used in the file-in-archive name */
             CHGVAR     VAR(&DEVNM) VALUE(&USELIB *TCAT '_ZIP')
/* Build the proper SAVLIB command for PKZIP                       */
             CHGVAR     VAR(&SAVCMD) VALUE(&SAVBLD1 *TCAT &USELIB +
                          *TCAT &SAVBLD2 *TCAT &DEVNM *TCAT +
                          &SAVBLD3 *TCAT &LOGLIB *TCAT '/' *TCAT +
                          &LOGFIL *TCAT &SAVBLD4 *TCAT &USELIB +
                          *TCAT &SAVBLD5)

/* (Optional) send message to which library is being saved                */
             CHGVAR     VAR(%SST(&MSGOUT1 1 10)) VALUE(&USELIB)
       /* SNDMSG     MSG(&MSGOUT1) TOUSR(*REQUESTER) MSGTYPE(*INFO) */
             SNDPGMMSG  MSGID(AQZ0000) MSGF(*LIBL/PKZIPMSG) +
                          MSGDTA(&MSGOUT1) TOPGMQ(*PRV) MSGTYPE(*INFO)


            /*  DMPCLPGM  Dump clp parameters*/

/* Step  3.3. Issue the PKZIP command for iPSRA SAVLIB direct to */
/*         tape.  Add the file to the end of the tape.           */
/*         Use *LEAVE for performance.                           */
/*         If this requires encryption add appropriate controls. */
             PKZIP      ARCHIVE(PKTAPE01) FILES(&SAVCMD) +
                          TYPARCHFL(*TAP) TYPFL2ZP(*DB) +
                          PKOVRTAPF(*TAPF &USELIB &TSEQ *TAPF &TDISP)

             CHGVAR     VAR(&LIBCNT) VALUE(&LIBCNT + 1)
             GOTO       CMDLBL(loop)


 LASTSTEP:
/* Step  4. Print list of files on tape with DSPTAP                  */
/*            Display The Tape File Contents  (Testing Purpose Only) */
             DSPTAP     DEV(&TAPDEVN) OUTPUT(*PRINT) ENDOPT(*REWIND)

 ENDPGM:
/* Step  5. Send completion message                                  */
             /* Char <- Dec auto conversion */
             CHGVAR VAR(&TSTNOCHAR)  VALUE(&LIBCNT)
             CHGVAR     VAR(&SAVCMD) VALUE(&MSGOUT2 *CAT &TSTNOCHAR)
             SNDPGMMSG  MSGID(AQZ0000) MSGF(*LIBL/PKZIPMSG) +
                          MSGDTA(&SAVCMD) TOPGMQ(*EXT) MSGTYPE(*COMP)

                  ENDPGM
```



---------- new end

C List Files

The list file capabilities provided in the PKZIP and PKUNZIP commands can be a powerful tool for maintaining detailed selection criteria and to exclude files. PKZIP and PKUNZIP commands also allow creating a list of files that are located in a particular archive.

Creating List Files

Both PKZIP and PKUNZIP can create a text format file of file names that meet criteria entered within the FILES and EXCLUDE parameters. In PKZIP, the output files contain the names of all files in the OS/400 format, depending on if the files are from the QSYS file system or IFS. The PKUNZIP program will produce a list of names in the format of the archive. To create an output list file, place the output file name in the parameter CRTLIST(). The default value is CRTLIST(*NONE).

Depending on the value of the TYPLISTFL parameter, the output file can be put in either the QSYS file system or IFS.

> **TYPLISTFL(*DB):** When the file system is QSYS, the output file will create a physical file (PF-DTA) with a record length of 132. For the file format in CRTLIST, you can use any of the following formats: library/file, library/file(member), file, or file(member). When a member is not entered, the member name will be the same as the file name. You should use the utility that your organization uses to edit data files.
>
> **TYPLISTFL(*IFS):** When using the IFS, the output file will create a stream file (*STMF object type). Most organization uses EDTF. For the file format in CRTLIST, you can use any of the following formats: file, file.suffix, dir1/file, dir1/dir2/../dirn/file, /dir1., etc. When the path does not start with '/', then the path starts in your current directory (relative path).

When creating a file manually, follow the creation attributes described above.

Using List Files as Input

Both PKZIP and PKUNZIP programs can use list file parameters for both selections of files (INCLFILE('file name')) and/or the excluding of file (EXCLFILE('file name'). They can also use an inlist file for the encryption recipient ENTPREC. The file name of

parameters depends on the setting of TYPLISTFL (*DB or *IFS) and should follow the guidelines in “Creating List Files,” above.

When using PKZIP, the format of files in the list file should be in the format of the iSeries files that will be processed. See the parameters FILES and EXCLUDE for specifications.

When using PKUNZIP, the format of the files in the list file should be in the format of the archive. See the parameters FILES and EXCLUDE for specifications.

PKUNZIP also has an option to create a list file in expanded mode, which will create the date and time along with the file names. This is accomplished by having a ‘>’ character being the in the first position of the CRTLIST parameter. See the two examples below.

Create normal list file:➔ PKUNZIP ARCHIVE('atest/V900/listf') CRTLIST('atest/listfile(demo)')

```
 Edit File: ATEST/LISTFILE(DEMO)
 Record :       1   of       4 by   8            Column :    1    132 by  74
CMD ....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+
                ************Beginning of data**************
    TESTLIB/MYFILE/MYMBR
    TESTLIB/MYFILEGE.R/MYMBR
    TESTLIB/MYFILETE.XT/MYMBR
    TESTLIB/MYFILE27.3/MYMBR
                ************End of Data********************
```

Create an expanded list file:➔ PKUNZIP ARCHIVE('atest/V900/listf') CRTLIST('>atest/listfile(demo)')

```
 Edit File: ATEST/LISTFILE(DEMO)
 Record :       1   of       4 by   8            Column :    1    132 by  74
CMD ....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+
                ************Beginning of data**************
    DT(05-20-03 16:16)  TESTLIB/MYFILE/MYMBR
    DT(02-14-03 16:44)  TESTLIB/MYFILEGE.R/MYMBR
    DT(02-14-03 16:44)  TESTLIB/MYFILETE.XT/MYMBR
    DT(02-14-03 16:44)  TESTLIB/MYFILE27.3/MYMBR
                ************End of Data********************
```

D Translation Tables

Text files (such as program source code) are usually held within an archive using the ASCII character set for compatibility with other versions of ***PKZIP***. For these to be usable on OS/400, they must be converted to the IBM EBCDIC character set. ***PKZIP****i* uses one of two possible internal translation tables, which should be suitable for most customers. These translation table members are used by parameters FTRAN and TRAN in both the PKZIP and PKUNZIP programs. Included (as part of the distribution) are a series of override translation tables. Some users may wish to define their own table.

The override translation tables included are stored as source members in file PKZTABLES ***PKZIP****i* resources tables. By referencing the members in parameters TRAN and FTRAN, ***PKZIP****i* will access the selected member in the PKZTABLES file and parse them to an internal hexadecimal table for use in translation.

The following translation tables are included:

Table Name	*Translation from*	*Translation to*	*Explanation*
ASCIIISO	EBCDIC	ASCII - iso	Translate Table
LATIN1	EBCDIC	ASCII	Latin Translate Table
NOOP	NO-OP		Translation Table Straight Hex
UKASCII	EBCDIC	UK	ASCII Translate Table
UKASCIIE	EBCDIC	UK	ASCII Translate Table-Euro
USASCII	EBCDIC	USA	ASCII Translate Table (Internal Default)
USASCIIE	EBCDIC	USA	ASCII Translate Table-Euro

Standard Code Page Support with Tables

Three data translation tables are available to assist with one or more of the latest standard EBCDIC text translation to ASCII. These tables were built to relate directly to IBM code pages numbers.

Code page tables available are:

Table Name	*ASCII Code Page*	*EBCDIC Code Page*	*Explanation*
PKZ819037	819	037	ASCII-819 <-> EBCDIC-037 Translation
PKZ819273	819	273	ASCII-819 <-> EBCDIC-273 Translation German
PKZ819277	819	277	ASCII-819 <-> EBCDIC-277 Translation Den/Nor
PKZ819278	819	278	ASCII-819 <-> EBCDIC-278 Translation Fin/Swe
PKZ819280	819	280	ASCII-819 <-> EBCDIC-0280 Translation Italy
PKZ819284	819	284	ASCII-819 <-> EBCDIC-284 Translation Spanish
PKZ819297	819	297	ASCII-819 <-> EBCDIC-297 Translation French
PKZ819500	819	500	ASCII-819 <-> EBCDIC-500 Translation ISO8859-1
PKZ819871	819	871	ASCII-819 <-> EBCDIC-871 Translation Icelandic
PKZ850037	850	037	ASCII-850 <-> EBCDIC-037 Translation
PKZ850284	850	284	ASCII-850 <-> EBCDIC-284 Translation Spanish

International Code Page Support

Some data-interchange environments require specialized multi-language character translation support. ***PKZIP***^{i} provides tables for character based data translation through translation tables that are also included in the PKZTABLES.

The tables for the following international code pages are provided in the ***PKZIP***^{i} PKZTABLES as members TRTxxyy (where xx = “from” and yy = “to”).

Language	*EBCDIC*	*ASCII*	*EURO/ASCII*	*FROM*	*TO*	*EURO*
German	273	850*	858	EB	AA	AI
Spanish	284	850	858	EJ	AA	AI
Portuguese	282	850	858	EI	AA	AI
Italian	280	850	858	EG	AA	AI
Danish	277	850	858	EE	AA	AI
Norwegian	277	850	858	EE	AA	AI
Swedish	278	850	858	EF	AA	AI
Finnish	278	850	858	EF	AA	AI
French	297	850	858	EM	AA	AI

* IBM-850 = IBM-4946

These members are provided "as is." It is the responsibility of the user to ensure that data translation mapping is in accordance with their business needs.

Translation Table Layout

There are two translation tables in PKZTABLES. The first table is a translation from ASCII to EBCDIC. The second is EBCDIC to ASCII.

In each table there are 256 entries representing hex values from x’00’ thru x’FF’.

Each entry is represented as a 4-character field such as 0x00 and 0xFF.

On each line there must be 8 entries with each entry separated by a space. With 8 entries per line, there must be 32 lines of table entries for each table set, representing the 256 translation values.

The tables have embedded comments to help in their documentation.

In the table example below, to translate an ASCII character **A** (hexadecimal x’41’ or decimal value of ‘65’), go to entry 65 in the table (Line 8, entry 2) and find a hexadecimal x’C1’ which is the EBCDIC **A**.

See “Example of PKZTABLES (USASCII) Translation Table.”

Note: Do not alter any other members found in the PKZTABLES file or ***PKZIP***^{i} may not function correctly.

Creating New Translation Table Members

Take the following steps to define your own translation table:

1. Copy one of the distributed members in PKZTABLES to a member name of your choice.
2. Edit the new table using the OS/400 Source Entry Utility (SEU).
3. Change the values with respect to the layout describe above, making sure not

to alter the overall table layout. If the overall layout is altered, ***PKZIP**i* may not work correctly.

4. Save the member and test your changes.

Example of PKZTABLES (USASCII) Translation Table

```
/* PKZIP/400 Translate Table USASCII to EBCDIC */
/*00-07*/ 0x00 0x01 0x02 0x03 0x37 0x2D 0x2E 0x2F  /*00-07*/
/*08-0f*/ 0x16 0x05 0x25 0x0B 0x0C 0x0D 0x0E 0x9F  /*08-0f*/
/*10-17*/ 0x10 0x11 0x12 0x13 0xB6 0xB5 0x32 0x26  /*10-17*/
/*18-1f*/ 0x18 0x19 0x3F 0x27 0x1C 0x1D 0x1E 0x1F  /*18-1f*/
/*20-27*/ 0x40 0x5A 0x7F 0x7B 0x5B 0x6C 0x50 0x7D  /*20-27*/
/*28-2f*/ 0x4D 0x5D 0x5C 0x4E 0x6B 0x60 0x4B 0x61  /*28-2f*/
/*30-37*/ 0xF0 0xF1 0xF2 0xF3 0xF4 0xF5 0xF6 0xF7  /*30-37*/
/*38-3f*/ 0xF8 0xF9 0x7A 0x5E 0x4C 0x7E 0x6E 0x6F  /*38-3f*/
/*40-47*/ 0x7C 0xC1 0xC2 0xC3 0xC4 0xC5 0xC6 0xC7  /*40-47*/
/*48-4f*/ 0xC8 0xC9 0xD1 0xD2 0xD3 0xD4 0xD5 0xD6  /*48-4f*/
/*50-57*/ 0xD7 0xD8 0xD9 0xE2 0xE3 0xE4 0xE5 0xE6  /*50-57*/
/*58-5f*/ 0xE7 0xE8 0xE9 0xBA 0xE0 0xBB 0xB0 0x6D  /*58-5f*/
/*60-67*/ 0x79 0x81 0x82 0x83 0x84 0x85 0x86 0x87  /*60-67*/
/*68-6f*/ 0x88 0x89 0x91 0x92 0x93 0x94 0x95 0x96  /*68-6f*/
/*70-77*/ 0x97 0x98 0x99 0xA2 0xA3 0xA4 0xA5 0xA6  /*70-77*/
/*78-7f*/ 0xA7 0xA8 0xA9 0xC0 0x6A 0xD0 0xA1 0x07  /*78-7f*/
/*80-87*/ 0x68 0xDC 0x51 0x42 0x43 0x44 0x47 0x48  /*80-87*/
/*88-8f*/ 0x52 0x53 0x54 0x57 0x56 0x58 0x63 0x67  /*88-8f*/
/*90-97*/ 0x71 0x9C 0x9E 0xCB 0xCC 0xCD 0xDB 0xDD  /*90-97*/
/*98-9f*/ 0xDF 0xEC 0xFC 0x4A 0xB1 0xB2 0x3E 0xB4  /*98-9f*/
/*a0-a7*/ 0x45 0x55 0xCE 0xDE 0x49 0x69 0x9A 0x9B  /*a0-a7*/
/*a8-af*/ 0xAB 0x0F 0x5F 0xB8 0xB7 0xAA 0x8A 0x8B  /*a8-af*/
/*b0-b7*/ 0x3C 0x3D 0x62 0x4F 0x64 0x65 0x66 0x20  /*b0-b7*/
/*b8-bf*/ 0x21 0x22 0x70 0x23 0x72 0x73 0x74 0xBE  /*b8-bf*/
/*c0-c7*/ 0x76 0x77 0x78 0x80 0x24 0x15 0x8C 0x8D  /*c0-c7*/
/*c8-cf*/ 0x8E 0x41 0x06 0x17 0x28 0x29 0x9D 0x2A  /*c8-cf*/
/*d0-d7*/ 0x2B 0x2C 0x09 0x0A 0xAC 0xAD 0xAE 0xAF  /*d0-d7*/
/*d8-df*/ 0x1B 0x30 0x31 0xFA 0x1A 0x33 0x34 0x35  /*d8-df*/
/*e0-e7*/ 0x36 0x59 0x08 0x38 0xBC 0x39 0xA0 0xBF  /*e0-e7*/
/*e8-ef*/ 0xCA 0x3A 0xFE 0x3B 0x04 0xCF 0xDA 0x14  /*e8-ef*/
/*f0-f7*/ 0xE1 0x8F 0x46 0x75 0xFD 0xEB 0xEE 0xED  /*f0-f7*/
/*f8-ff*/ 0x90 0xEF 0xB3 0xFB 0xB9 0xEA 0xBD 0xFF  /*f8-ff*/

/* PKZIP/400 Translate Table EBCDIC to USASCII */
/*00-07*/ 0x00 0x01 0x02 0x03 0xEC 0x09 0xCA 0x7F  /*00-07*/
/*08-0f*/ 0xE2 0xD2 0xD3 0x0B 0x0C 0x0D 0x0E 0xA9  /*08-0f*/
/*10-17*/ 0x10 0x11 0x12 0x13 0xEF 0xC5 0x08 0xCB  /*10-17*/
/*18-1f*/ 0x18 0x19 0xDC 0xD8 0x1C 0x1D 0x1E 0x1F  /*18-1f*/
/*20-27*/ 0xB7 0xB8 0xB9 0xBB 0xC4 0x0A 0x17 0x1B  /*20-27*/
/*28-2f*/ 0xCC 0xCD 0xCF 0xD0 0xD1 0x05 0x06 0x07  /*28-2f*/
/*30-37*/ 0xD9 0xDA 0x16 0xDD 0xDE 0xDF 0xE0 0x04  /*30-37*/
/*38-3f*/ 0xE3 0xE5 0xE9 0xEB 0xB0 0xB1 0x9E 0x1A  /*38-3f*/
/*40-47*/ 0x20 0xC9 0x83 0x84 0x85 0xA0 0xF2 0x86  /*40-47*/
/*48-4f*/ 0x87 0xA4 0x9B 0x2E 0x3C 0x28 0x2B 0xB3  /*48-4f*/
/*50-57*/ 0x26 0x82 0x88 0x89 0x8A 0xA1 0x8C 0x8B  /*50-57*/
/*58-5f*/ 0x8D 0xE1 0x21 0x24 0x2A 0x29 0x3B 0xAA  /*58-5f*/
/*60-67*/ 0x2D 0x2F 0xB2 0x8E 0xB4 0xB5 0xB6 0x8F  /*60-67*/
/*68-6f*/ 0x80 0xA5 0x7C 0x2C 0x25 0x5F 0x3E 0x3F  /*68-6f*/
/*70-77*/ 0xBA 0x90 0xBC 0xBD 0xBE 0xF3 0xC0 0xC1  /*70-77*/
/*78-7f*/ 0xC2 0x60 0x3A 0x23 0x40 0x27 0x3D 0x22  /*78-7f*/
/*80-87*/ 0xC3 0x61 0x62 0x63 0x64 0x65 0x66 0x67  /*80-87*/
/*88-8f*/ 0x68 0x69 0xAE 0xAF 0xC6 0xC7 0xC8 0xF1  /*88-8f*/
/*90-97*/ 0xF8 0x6A 0x6B 0x6C 0x6D 0x6E 0x6F 0x70  /*90-97*/
/*98-9f*/ 0x71 0x72 0xA6 0xA7 0x91 0xCE 0x92 0x0F  /*98-9f*/
/*a0-a7*/ 0xE6 0x7E 0x73 0x74 0x75 0x76 0x77 0x78  /*a0-a7*/
/*a8-af*/ 0x79 0x7A 0xAD 0xA8 0xD4 0xD5 0xD6 0xD7  /*a8-af*/
/*b0-b7*/ 0x5E 0x9C 0x9D 0xFA 0x9F 0x15 0x14 0xAC  /*b0-b7*/
/*b8-bf*/ 0xAB 0xFC 0x5B 0x5D 0xE4 0xFE 0xBF 0xE7  /*b8-bf*/
/*c0-c7*/ 0x7B 0x41 0x42 0x43 0x44 0x45 0x46 0x47  /*c0-c7*/
```

```
/*c8-cf*/ 0x48 0x49 0xE8 0x93 0x94 0x95 0xA2 0xED  /*c8-cf*/
/*d0-d7*/ 0x7D 0x4A 0x4B 0x4C 0x4D 0x4E 0x4F 0x50  /*d0-d7*/
/*d8-df*/ 0x51 0x52 0xEE 0x96 0x81 0x97 0xA3 0x98  /*d8-df*/
/*e0-e7*/ 0x5C 0xF0 0x53 0x54 0x55 0x56 0x57 0x58  /*e0-e7*/
/*e8-ef*/ 0x59 0x5A 0xFD 0xF5 0x99 0xF7 0xF6 0xF9  /*e8-ef*/
/*f0-f7*/ 0x30 0x31 0x32 0x33 0x34 0x35 0x36 0x37  /*f0-f7*/
/*f8-ff*/ 0x38 0x39 0xDB 0xFB 0x9A 0xF4 0xEA 0xFF  /*f8-ff*/
/* PKZIP/400 Translate Tables end */
```

E SPOOL Files Considerations

This appendix contains information on how ***PKZIP***^{i} handles spool files in different scenarios that might be helpful to consider in planning for compressing spool files.

Spool File Selections

Care should be taken when selecting spool files not to set all of the spool file selection parameters to *ALL, as this will select all spool files on your iSeries. This is why the default for the user ID is SFUSER(*CURRENT) to at least limit it to the current user in case a selection is not filled in correctly.

If a spool file is deleted after the selection but before the actual compression takes place, the PKZIP job will fail.

SPLF Attributes

When a spool file is selected and the parameter EXTRAFL is coded *YES (the default), then the extended attributes listed below are stored in archive and can be viewed with PKUNZIP TYPE(*VIEW) VIEWOPT(*ALL). Also when the spool files are stored in the archive, the date and time for the file is the spool files creation date and time and can be viewed with PKUNZIP.

Extended Attributes:

- Description: The spool file description is built as follows:

  "Job-Name/User-Name/#Job-Number/Spool-File-Name/Fspool-File-Number.Suffix" For Example: "MYJOB/BILLS#152681/INVOICE/F0021.SPLF"
- Spool file type: *SCS: SNA Character Stream, *IPDS: An intelligent printer data stream, *AFPDS: Advanced Function Print Data Stream, *USERASCII: An ASCII data stream user defined, *LINE: Line data that is very printer specific, and *AFPDSLINE: Mixed data (line data and AFPDS data).
- Target file created: Describes the target type file created during compression. SPLF: Spool Files, TXT: ASCII Text Conversion, and PDF: Portable Document Format.
- Number of pages contained in the spool file.

An example of the attributes view seen with –VIEWOPT(*ALL) for a spool file converted to a PDF might appear as follows:

```
Filename: CRTCM60.PDF
Detected File type:     Binary
Created by:             PKZIP for i5/OS   9.0    PKZIP 2.x compatible
Minimum to Extract:     PKZIP 2.0  Or Greater
Compression method:     Deflated   [Fast]
Date and Time           2003 Oct 17 07:22:00
Compressed size:        2316 bytes
Uncompressed size:      8146 bytes
32-bit CRC value (hex): 40950039
Extended attributes:    yes, [Length = 112]
Spool File Type:*SCS, Target File:PDF, Nbr Of pages(3).
SPLF Desc:EVWSS/EVWSS/#007892/CRTCM/F0060.PDF.
File Comment:"none"
```

The preceding view comes from the following spool file:

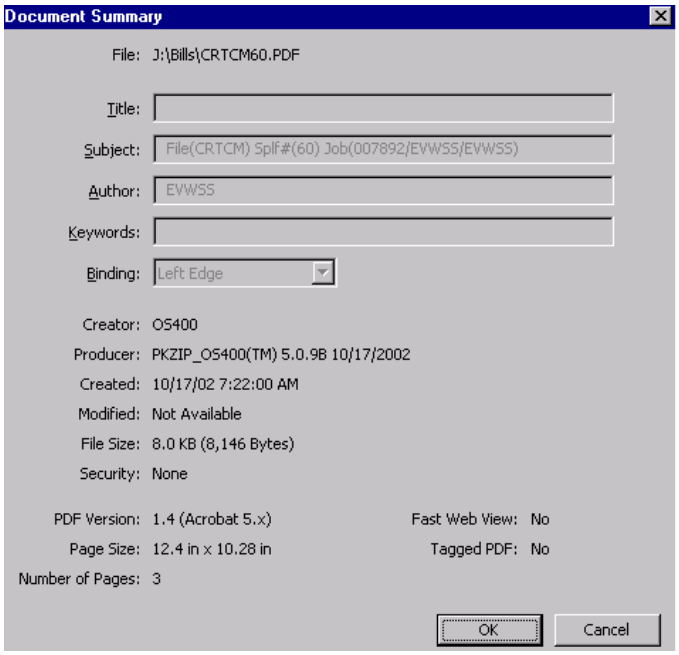
```
5722SS1  V5R1M0   Work With Output Queue    QPRINT2  in  QGPL  11/19/02 14:08:53 Page    1
File   User  User Data Status Pages  Cpy Form Tp  Pty File Number  Job    Number Date
Time
CRTCM  EVWSS           RDY      3    1   *STD     5         60    EVWSS  007892 10/17/02
07:22:00
```

PDF Creation Attributes

When creating a PDF, the attributes are also stored in the PDF document to help trace back what spool file they originated from.

- The date and time of the spool file creation will be the PDF date and time of creation.
- The author will be the user ID that created the spool file.
- The subject will be made up of the spool file name, number, and the job (number/user ID/job name).

An example of a PDF summary is:



F Contact Information

PKWARE, Inc.

Web Site: www.pkware.com

For Licensing, please contact the Sales Division at 937-847-2374 or email PKSALES@PKWARE.COM.

For Technical Support assistance, please contact the Product Services Division at 937-847-2687 or visit the Support Web site.

PROBLEM REPORTING

Providing appropriate documentation on the initial call for a problem expedites the analysis and resolution process. The following sections describe the type of information that should be supplied for each category of problem.

PROBLEM REPORTING (General)

When reporting a problem regarding ***PKZIP for i5/OS*** or ***SecureZIP for i5/OS***, please be prepared to provide the following information:

- The displayed output from ➔**CALL ziplib/WHATOSV** or the details that WHATOSV provides
- Release level of the operating system
- Release level of PKZIP for i5/OS being run
- A description of the process being run and any differentiating circumstances from job(s) that do run
- A display of the command problem with parameters
- A copy of the output and JobLog from the failing execution
- If run from a CL and practical, please include source listing of the CL
- If PKUNZIP is failing, provide the Output from the following:

  ➔ **PKUNZIP TYPE(*VIEW) VIEWOPT(*ALL)**

- If requested by Technical Support, the display with various tracing options turned on
- If practical, please include the archive/input file involved in the failing execution

PROBLEM REPORTING (Licensing)

When reporting a problem regarding licensing, please be prepared to provide the following information:

- The displayed output from **➔CALL ziplib/WHATOSV**
- A copy of the INSTPKLIC command and its parameters
- A copy of the output from the INSTPKLIC job

If the problem occurs in a ***PKZIP**^i^* job then follow the steps outlined above for ***PKZIP for i5/OS*** or ***SecureZIP for i5/OS***.

Glossary

This glossary provides definitions for items that may have been referenced in the PKZIP® documentation. It is not meant to be exhaustive. One Web site that provides excellent source documentation for computing terms is the IBM Terminology site:

http://www-306.ibm.com/ibm/terminology

Absolute Path Name

A string of characters that is used to refer to an object, starting at the highest level (or root) of the directory hierarchy. The absolute path name must begin with a slash (/), which indicates that the path begins at the root. This is in contrast to a Relative Path Name. See also Path Name.

AES

The Advanced Encryption Standard is the official US Government encryption stand for customer data.

American Standard Code for Information Interchange

The ASCII code (American Standard Code for Information Interchange) was developed by the American National Standards Institute for information exchange among data processing systems, data communications systems, and associated equipment and is the standard character set used on MS-DOS and UNIX-based operating systems. In a ZIP archive, ASCII is used as the normal character set for compressed text files. The ASCII character set consists of 7-bit control characters and symbolic characters, plus a single parity bit. Since ASCII is used by most microcomputers and printers, text-only files can be transferred easily between different kinds of computers and operating systems. While ASCII code does include characters to indicate backspace, carriage return, etc., it does not include accents and special letters that are not used in English. To accommodate those special characters, extended ASCII has additional characters (128-255). Only the first 128 characters in the ASCII character set are standard on all systems. Others may be different for a given language set. It may be necessary to create a different translation tables (see Translation Table) to create standard translation between ASCII and other character sets.

American National Standards Institute (ANSI)

An organization sponsored by the Computer and Business Equipment Manufacturers Association for establishing voluntary industry standards.

ANSI

See American National Standards Institute.

API

See Application Programming Interface, below.

Application Programming Interface

An interface between the operating system (or systems-related program) that allows an application program written in a high-level language to use specific data or services of the operating system or the program. The API also allows the user to develop an application program written in a high level language to access PKZIP data and/or functions of the PKZIP system.

Archive

The act of transferring files from the computer into a long-term storage medium. Archived files are often compressed to save space.

An individual file or group of files which must be extracted and decompressed in order to be used.

A file stored on a computer network, which can be retrieved by a file transfer program (FTP) or other means.

The PKZIP file that holds the compressed/zipped data file.

ASCII

See American Standard Code for Information Interchange.

iSeries Object

An object that exists in a library on the iSeries system and is represented by an object on the PC. For example, a user profile is an iSeries object represented on the PC by the user profile object.

Binary File

A file that contains codes that are not part of the ASCII character set. Binary files can utilize all 256 possible values for each byte in the file.

Code Page

A specification of code points for each graphic character set or for a collection of graphic character sets. Within a given code page, a code point can have only one specific meaning. A code page is also sometimes known as a code set.

Command Line

The blank line on a display console where commands, option numbers, or selections can be entered.

Control Language (CL) Program

A program that is created from source statements consisting entirely of control language commands.

CRC

See Cyclic Redundancy Check.

Cryptography

A method of protecting data. Cryptographic services include data encryption and message authentication.

In cryptographic software, the transformation of data to conceal its meaning; secret code.

The transformation of data to conceal its information content, to prevent its undetected modification, or to prevent its unauthorized use.

Current Library

The library that is specified to be the first user library searched for objects requested by a user. The name for the current library can be specified on the sign-on display or in a user profile. When you specify an object name (such as the name of a file or program) on a command, but do not specify a library name, the system searches the libraries in the system part of the library list, then searches the current library before searching the user part of the library list. The current library is also the library that the system uses when you create a new object, if you do not specify a library name.

Cyclic Redundancy Check (CRC)

A Cyclic Redundancy Check is a number derived from a block of data, and stored or transmitted with the data in order to detect any errors in transmission. This can also be used to check the contents of a ZIP archive. It's similar in nature to a checksum. A CRC may be calculated by adding words or bytes of the data. Once the data arrives at the receiving computer, a calculation and comparison is made to the value originally transmitted. If the calculated values are different, a transmission error is indicated. The CRC information is called redundant because it adds no significant information to the transmission or archive itself. It's only used to check that the contents of a ZIP archive are correct. When a file is compressed, the CRC is calculated and a value is calculated based upon the contents and using a standard algorithm. The resulting value (32 bits in length) is the CRC that is stored with that compressed file. When the file is decompressed, the CRC is recalculated (again, based upon the extracted contents), and compared to the original CRC. Error results will be generated showing any file corruption that may have occurred.

Data Compression

The reduction in size (or space taken) of data volume on the media when performing a save or store operations.

Data Integrity

The condition that exists as long as accidental or intentional destruction, alteration, or loss of data does not occur.

Within the scope of a unit of work, either all changes to the database management systems are completed or none of them are. The set of change operations are considered an integral set.

DBCS

See Double-byte Character Set.

Double-byte Character Set (DBCS)

A set of characters in which each character is represented by 2 bytes. Languages such as Japanese, Chinese, and Korean, which contain more symbols than can be represented by 256 code points, require double-byte character sets. Because each character requires 2 bytes, the typing, displaying, and printing of DBCS characters requires hardware and programs that support DBCS. Four double-byte character sets are supported by the system: Japanese, Korean, Simplified Chinese, and Traditional Chinese. See also the Single-Byte Character Set (SBCS).

EBCDIC

See the Extended Binary Coded Decimal Interchange Code, below.

Encryption

The transformation of data into an unintelligible form so that the original data either cannot be obtained or can be obtained only by decryption.

Extended Attribute

Information attached to an object that provides a detailed description about the object to an application system or user.

Extended Binary Coded Decimal Interchange Code (EBCDIC)

The Extended Binary Coded Decimal Interchange Code is an 8-bit binary code for larger IBM mainframes in which each byte represents one alphanumeric character or two decimal digits. The single-byte structure has a range of X'00' to X'FF'. Control commands are subset with a range of X'00' to X'3F' while graphic characters have a range from X'41' to X'FE'. The space character is represented by a X'40'. EBCDIC is similar in nature to ASCII code, which is used on many other computers. When ZIP programs compress a text file, they translate data from EBCDIC to ASCII characters within a ZIP archive using a translation table.

File Transfer Protocol (FTP)

In TCP/IP, an application protocol used for transferring files to and from host computers. FTP requires a user ID and possibly a passphrase to allow access

to files on a remote host system. FTP assumes that the transmission control protocol (TCP) is the underlying protocol.

FTP

See File Transfer Protocol above.

GZIP

GZIP (also known as GNU zip) is a compression utility designed to utilize a different standard for handling compressed file data in an archive. Its main advantages over other compression utilities are much better compression and freedom from patented algorithms. It has been adopted by the GNU project and is now relatively popular on the Internet. Additional information can be found at http://www.gzip.org.

Integrated File System

A function of the operating system that provides storage support similar to personal computer operating systems (such as DOS and OS/2) and UNIX systems.

Interactive Job

A job started for a person who signs on to a work station and communicates (or "converses") with another computing entity such as a mainframe or iSeries system. This is in contrast to a Batch Job.

Lempel-Ziv (LZ)

A technique for compressing data. This technique replaces some character strings, which occur repeatedly within the data, with codes. The encoded character strings are then kept in a common dictionary, which is created as the data is being sent.

Library List

A list that indicates which libraries are to be searched and the order in which they are to be searched. The system-recognized identifier is *LIBL.

Logical Partition

A subset of a single iSeries system that contains resources (such as processors, memory, and input/output devices). A logical partition operates as an independent system. If hardware requirements are sufficient, multiple logical partitions can exist within a system.

New ZIP Archive

A new ZIP archive is the archive created by a compression program when either an old ZIP archive is updated or when files are compressed and no ZIP archive currently exists. It may be thought of as the "receiving" archive. Also see Old ZIP archive.

Null Value

A parameter position within a record for which no value is specified.

n-way Processor Architecture

A processor architecture that provides expandability for future system growth by allowing for additional processors. To the user, the additional processors are transparent because they separately manage the work load by sharing the work evenly among the n-way processors.

Old ZIP Archive

An old ZIP archive is an existing archive which is opened by a compression program to be updated or for its contents to be extracted. It may be thought of as the “sending” archive. Also see New ZIP archive.

Output Queue

An AS/400 object that contains entries for spooled output files to be written to an output device.

Packed Decimal Format

A decimal value in which each byte within a field represents two numeric digits except the far right byte, which contains one digit in bits 0 through 3 and the sign in bits 4 through 7. For all other bytes, bits 0 through 3 represent one digit; bits 4 through 7 represent one digit. For example, the decimal value +123 is represented as 0001 0010 0011 1111 (or 123F in hexadecimal).

Passphrase

A sentence, phrase, or random string of characters that may include spaces and other non-alphabetical characters that serves as a password. The term *password* implies a single, recognized name or word from the dictionary. The term *passphrase* is meant as encouragement to use longer, more varied strings as passwords. Longer passwords—or passphrases—consisting of random strings that contain spaces and other such characters are much more secure than typical passwords of six to eight characters that use words from the dictionary.

Path Name

A string of characters used to refer to an object. The string can consist of one or more elements, each separated by a slash (/), and may begin with a slash. Each element is typically a directory or equivalent, except for the last element, which can be a directory or another object (such as a file).

A sequence of directory names followed by a file name, each separated by a slash.

In a hierarchical file system (HFS), the name used to refer to a file or directory. The path name must start with a slash (/) and consist of elements separated by a slash. The first element must be the name of a registered file

system. All remaining elements must be the name of a directory, except the last element, which can be the name of a directory or file. See also Absolute Path Name and Relative Path Name.

The name of an object in the integrated file system. Protected objects have one or more path names.

Physical File

Describes how data is to be presented to (or received from) a program and how data is stored in the database. A physical file contains a single record format and at least one member.

Production Library

A library which contains objects needed for normal processing. This contrasts with a Test Library.

QSYS

The library shipped with the iSeries system that contains objects, such as authorization lists and device descriptions created by a user, and the system commands and other system objects required to run the system. The system identifier is QSYS.

Qualified Name

The full name of the library that contains the object and the name of the object.

Relative Path Name

A string of characters that is used to refer to an object, starting at some point in the directory hierarchy other than the root. A relative path name does not begin with a slash (/). The starting point is frequently a user's current directory. This is in contrast to an Absolute Path Name. See also Path Name.

Return Code

A value generated by operating system software to a program to indicate the results of an operation by that program. The value may also be generated by the program and passed back to the operator.

SBCS

See Single-Byte Character Set.

Single-Byte Character Set (SBCS)

A coded character set in which each character is represented by a one-byte code point. A one-byte code point allows representation of up to 256 characters. Languages that are based on an alphabet, such as the Latin alphabet (as contrasted with languages that are based on ideographic characters) are usually represented by a single-byte coded character set. For

example, the Spanish language can be represented by a single-byte coded character set. See also the Double-Byte Character Set (DBCS).

Source File

A file of programming code that has not yet been compiled into machine language. A source file can be created by the specification of FILETYPE(*SRC) on the create command. A source file can contain source statements for such items as high-level language programs and data description specifications. Source files maintained on a PC typically use a .TXT as the extension. On a mainframe, source files are typically found in a partitioned data set or are maintained within a library management tool.

Spool File

Files that exist in an "output queue" which contain reports to printed on the AS/400 system. Theses files along with attributes can then be directed and transformed to a printer attached to your system.

Stream File

A data file that contains continuous streams of bits such as PC files, documents, and other data stored in iSeries folders. Stream files are well suited for storing strings of data such as the text of a document, images, audio, and video. The content and format of stream files are managed by the application rather than by the system.

System Library

The library shipped with the operating system that contains objects such as authorization lists and device descriptions created by a user. Also included are system commands and other system objects required to run the system. The system identifier is QSYS.

Translation Table

Translation tables are used by the PKZIP and PKUNZIP programs for translating characters in compressed text files between the ASCII character sets used within a ZIP archive and the EBCDIC character set used on IBM-based systems. These tables may be created and modified by the user as documented in the User's Guide.

Trigger

A set of predefined actions that run automatically whenever a specified action or change occurs, for example, a change to a specified table or file. Triggers are often used to automate environments, such as running a backup when a certain number of transactions are processed.

Truncate

To cut off or delete the data that will not fit within a specified line width or display. This may also be attributed to data that does not fit within the specified length of a field definition.

User Interface

The actions or items that allow a user to interact with (and/or perform operations on) a computer.

ZIP64

ZIP64 is reference to the archive format that supports more than 65,534 files per archive, uncompressed files greater than 4 Gig and archives greater than 4 Gig.

ZIP Archive

A ZIP archive is used to refer to a single file that contains a number of files compressed into a much smaller physical space by the ZIP software.

Index