CA Compress™ Data Compression

Reference Guide

Release 5.5

CA Technologies Product References

This document references the following CA Technologies product:

- CA Compress™ Data Compression (CA Compress)

Contact CA Technologies

Contact CA Support

For your convenience, CA Technologies provides one site where you can access the information that you need for your Home Office, Small Business, and Enterprise CA Technologies products. At http://ca.com/support, you can access the following resources:

- Online and telephone contact information for technical assistance and customer services
- Information about user communities and forums
- Product and documentation downloads
- CA Support policies and guidelines
- Other helpful resources appropriate for your product

Providing Feedback About Product Documentation

If you have comments or questions about CA Technologies product documentation, you can send a message to techpubs@ca.com.

To provide feedback about CA Technologies product documentation, complete our short customer survey which is available on the CA Support website at http://ca.com/docs.

Contents

Chapter 1: Introduction

CA Compress is a CA Technologies mainframe software product that transparently compresses and expands VSAM and Physical Sequential data sets, including those under CICS control. It fully supports data sets under the control of IBM's Storage Management Subsystem (SMS). No special procedures are required to compress SMS-controlled data sets.

CA Compress (CA Compress) offers several compression routines. With CA Compress, you can achieve compression of up to 80 percent. In addition, you can create custom compression routines for specific data sets to achieve even greater compression. The product includes a performance enhancement feature which optimizes VSAM data sets to improve elapsed time for sequential and direct access operations.

Data compression with CA Compress offers numerous advantages to IS organizations:

- Significantly reduced DASD requirements.
- Reduced backup and restore times.
- Improved data security.
- Easy to use interactive ISPF interface.
- Choice of maximum compression or minimum CPU overhead.
- Transparency to application programs.

Chapter 2: Control File Maintenance Utility

The Control File holds the information used by CA Compress to determine which data sets are to be compressed and how the compression is to be done. The Control File Maintenance Utility lets you perform the following functions:

- Add, modify, and delete data set names and name patterns, and their compression parameters.
- Maintain File Descriptor Tables (FDTs) in the Control File.
- Report on the VSAM data sets and patterns, physical sequential data sets and patterns, and FDTs in the Control File.

Although you may find the Interactive User Interface (IUI) more convenient most of the time, the Control File Maintenance Utility supports all maintenance functions. For instance, unlike the IUI, it can maintain the Control File even when CA Compress is not active. Also, unlike the IUI, it can add paths to the base cluster.

This section contains the following topics:

How the Utility Works

The Control File Maintenance Utility reads statements from an input data set and performs the specified actions on the Control File. The Control File Maintenance Utility can update any Control File on your system. This includes the Control File currently in use by an active CA Compress Subsystem.

The following JCL is a sample of the Control File Maintenance Utility and can be found in member CFUJCL in YOUR.CAI.CCVBJCL.

```
//*CFUMNT JOB
//*
//CFU      EXEC PGM=DEFXP001,REGION=1000K
//STEPLIB  DD  DSN=YOUR.CAI.CCVBLOAD,DISP=SHR
//CMDPRINT DD  SYSOUT=*
//MSGPRINT DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
//SUIIN    DD *
    --- ENTER MAINTENANCE COMMANDS HERE ---
/*
```

Control File Statements

Statements in the Control File are executed as they are read. By default, the production Control File is updated unless you specify otherwise in a SET statement.

Syntax Rules

The following section explains the syntax rules.

- Any line which starts with an asterisk (*) in column 1 is treated as a comment line.
- A statement consists of the statement name followed by one or more parameters. For example:

  ```
  ADD DSNAME=MY.DATA.SET,DATA=MY.DATA.SET.D,SUPEREXP
  ```

- Parameter names can be abbreviated to just enough characters to make them unambiguous, but not fewer than 3.
- Parameters can be specified in any order. The first parameter must appear on the same line as the statement name.
- A statement can span more than one line.
- A parameter cannot span more than one line unless the statement is continued through Statement Continuation Method B below.
- A blank following a parameter terminates a statement. Characters following the blank are treated as comments.
- Statement Continuation Method A: If a parameter is followed immediately by a comma and a space, then the statement is continued onto the next line. Characters following the space are considered comments. The parameters appearing on the next line can start in any column. With this continuation method, each parameter must be completed on a single line.
- Statement Continuation Method B: If the statement is coded through column 71, and an X is coded in column 72, then the statement is continued onto the next line. The first nonblank in the next line is appended to the character in column 71 in order to form the continued statement. With this continuation method, the line can be split anywhere. In the example below, the RELEASE=80 parameter is continued onto the next line:

  ```
  ----+----1----+----2----+ ..//.. +----7----+----8
      ADD DSNAME=MY.DATA.SE        DE,RELEX
           ASE=80
  ```

- The following special characters can be used in data set names:
  - An asterisk (*) means any characters in a single node.
  - A question mark (?) means any one character.
  - A slash (/) means any suffix of characters.
  - An exclamation point (!) means any characters.

The following table provides examples on how you can use special characters in data set names:

Syntax	Description
DSN=*	Selects all single-level VSAM data set names.
DSN=*.*	Selects all two-level VSAM data set names.
DSN=A.*.PROD	Selects all three-level VSAM data set names that have an A as the first node, any character or characters as the second node, and PROD as the third node.
DSN=A*.PROD	Selects all two-level VSAM data set names that have an A followed by zero to seven other characters as the first node, and PROD as the second node.
DSN=?	Selects all single-character VSAM data set names.
DSN=A.TEST??	Selects all two-level VSAM data set names that have an A as the first node, and TEST plus two other characters as the second node.
DSN=A/	Selects all VSAM data sets that begin with the character A. The data set names can have any number of nodes. The first node can be the letter A, or be a string which starts with A.
DSN=A.TEST/	Selects all VSAM data sets that begin with the string A.TEST. Examples: A.TEST A.TEST1 A.TEST1.TEST2 A.TEST.PROD
DSN=A.*.C?./	Selects all VSAM data sets which have a first node of A, any second node, a third node which is two characters in length and the first character is the letter C, and any node or nodes which follow.
DSN=!TEST	Selects all VSAM data sets that end with the string TEST.

Syntax	Description
DSN=!TEST!	Selects all VSAM data sets that have the string TEST somewhere in it. The string TEST can be at the beginning or the end of the data set name.
DSN= !TEST!VSAM	Selects all VSAM data sets, which have the string TEST somewhere in the name and VSAM at the end. Examples: A.TEST.VSAM A.TESTVSAM A.TEST1.VSAM A.TEST1.KSDSVSAM

ADD Statement (VSAM)

The ADD statement for VSAM defines a cluster or pattern to CA Compress, indicating that the discrete data set(s) matching the pattern should be compressed using the attributes specified.

```
ADD     {DSNAME=datasetname,DATA=datacomponentname}|
        {PATTERN=patternname}
        [,PATHS=(pathname1[,pathname2...
        {,SUPEREXP|
        FDT=fdtname[,STANDARD]|DICTIONARY=dictionary}
        [,SCHEDULED={SCHED|OPEN|LOAD}]
        [,EXCLUDE]
        [,RELEASE=percent]
        [,NON-COMP=#]
        [,SCOPE=CICS|ALL]
        [,IAM=YES|NO]
        [,UEXIT=YES|NO]
```

DSNAME=datasetname

Specifies a VSAM cluster which is to be compressed. The DATA parameter is required when the DSNAME parameter is specified.

DATA=datacomponentname

Specifies the data component of the VSAM cluster specified by DSNAME. This parameter is required when the DSNAME parameter is specified.

PATTERN=patternname

Specifies a data set name pattern which selects data sets for compression. The pattern can contain any character that is valid in a data set name along with the special characters ?, *, ! and /. These special characters are defined in the section Syntax Rules in this chapter.

PATHS=(pathname1 [,pathname2 ...

Specifies all path names associated with the DSNAME. This parameter is required if there are any paths defined for this cluster. A maximum of 20 path names can be specified.

SUPEREXP|FDT=fdtname[,STANDARD]|DICTIONARY=dictionary

Selects the type of compression desired for the data set or pattern being defined.

The optional parameter STANDARD can be specified only in conjunction with the FDT=fdtname parameter. It informs the CA Compress Subsystem that the FDT specified should be used to provide the compression or expansion tables only, and that the RDL specified in the FDT should be ignored. The STANDARD parameter allows one FDT to be used for a number of data sets that share character distribution characteristics but not key structure. STANDARD is implied if the FDT parameter specifies a name of the form STDTBLxx (where xx is a numeric value from 01 to 06).

The DICTIONARY parameter can specify a CA-supplied IBM Hardware Compression dictionary of the form HC#STDxx (where xx is numeric 01 to 05).

SUPEREXP specifies the low-overhead compression algorithm called Super Express.

SCHEDULED={SCHED|OPEN|LOAD}

The SCHEDULED parameter provides the capability to compress records in a data set as they are written or updated. OPEN specifies that the data set be compressed the next time it is opened for OUTPUT. LOAD specifies that compression should begin the next time the data set is completely reloaded; the entire data set will be compressed during the load process. SCHED specifies that the data set is currently compressed and was compressed as SCHEDULED. Because SCHEDULED data sets have certain requirements, CA Compress must be made aware that these data sets were originally SCHEDULED. The compression method for a data set defined as SCHEDULED can be changed using the ALTER statement any time before compression begins.

EXCLUDE

Specifies that the data set is excluded from compression.

RELEASE=percent

Specifies that excess space should be released from the VSAM cluster. It is also used to provide the percentage of the space that is to be RETAINED. The percentage is applied to the file size (high-used RBA) and the resulting value is added to the file size; the resulting value is the amount retained. Space can only be released from the last extent of the data set.

Note: A RELEASE value of 0 causes the default value (10%) to be used.

NON-COMP=#

Specifies a data set's original noncompressible field length for other than custom Huffman or Tailored RDL methods instead of allowing CA Compress to calculate it from the data set's attributes. This parameter is especially helpful for exempting nonkey bytes from compression or for setting aside noncompressible bytes at the beginning of ESDS records. Do not code a value less than the end of the last key.

SCOPE=CICS|ALL

Specifies whether data set should be considered defined as compressed only under CICS (SCOPE=CICS) or always (SCOPE=ALL). This parameter is primarily intended as a conversion aid. The default is SCOPE=ALL.

IAM=YES|NO

Specifies that this Control File entry applies to IAM (Innovation Access Method) data sets as well as VSAM data sets (YES), or that this entry does not apply to IAM data sets (NO). The default is YES.

UEXIT=YES|NO

Specifies whether the Transparency User Exit is to be invoked during compression and expansion of each record. The default is NO.

ALTER Statement (VSAM)

The ALTER statement changes the compression attributes of a data set or pattern that is currently defined to CA Compress. There are no defaults.

```
ALTER{DSNAME=datasetname| PATTERN=patternname}
    [,ADDPATHS=(pathname1[,...,pathname5)]
     [,DELPATHS=(pathname1[,...,pathname5)]
     [,SUPEREXP|
    FDT=fdtname[,STANDARD]DICTIONARY=dictionary]
     [,SCHEDULED={SCHED|OPEN|LOAD}]
     [,EXCLUDE={YES|NO}]
     [,RELEASE={percent|NO}]
     [,NEWNAME=newdatasetname]
     [,NON-COMP=#]
     [,SCOPE=CICS|ALL]
     [,IAM=YES|NO]
     [,UEXIT=YES|NO],
     [,FORCE]
```

DSNAME=datasetname

Specifies the name of the Control File entry that is being ALTERed. For a simple rename function (NEWNAME is the only other parameter specified), this value can be a cluster, data component, or path name. For all other functions, only the cluster name is valid.

PATTERN=patternname

Specifies the pattern name that is to be ALTERed. The pattern can contain any character that is valid in a data set name along with the special characters ?, *, ! and /. These special characters are defined in the section Syntax Rules in this chapter.

ADDPATHS=(pathname1,...pathname5)

Specifies the path names to be added to the existing Control File entry.

DELPATHS=(pathname1,...pathname5)

Specifies the path names to be deleted from the existing Control File entry.

SUPEREXP|FDT=fdtname[,STANDARD]|DICTIONARY=dictionary

Specifies the type of compression desired for the data set or pattern being ALTERed.

The optional parameter STANDARD can be specified only in conjunction with the FDT=fdtname parameter. It informs the CA Compress Subsystem that the FDT specified should be used to provide the Huffman tables only and that the RDL specified in the FDT should be ignored. The STANDARD parameter allows one FDT to be used for a number of data sets that can share character distribution characteristics but not key structure. The FDT is assumed to be STANDARD if the FDT parameter specifies a name of the form STDTBLxx (where xx is a numeric value from 01 to 06). The DICTIONARY parameter can specify a CA-supplied IBM Hardware Compression dictionary of the form HC#STDxx (where xx is numeric 01 to 05).

The SUPEREXP parameter requests the low-overhead compression algorithm called Super Express Compression.

SCHEDULED={SCHED|OPEN|LOAD}

The SCHEDULED parameter provides the capability to compress each record in a data set as it is accessed. OPEN specifies that the data set should start being compressed the next time it is opened for OUTPUT. LOAD specifies that compression should begin the next time the data set is completely reloaded; the entire data set is compressed during the load process. SCHED specifies that the data set is currently compressed and was compressed as SCHEDULED. Because SCHEDULED data sets have certain requirements, CA Compress must be made aware that these data sets were originally SCHEDULED. The compression type of a SCHEDULED data set can be changed, by using the ALTER statement, any time before compression begins.

EXCLUDE={YES|NO}

Specifies that the data set is excluded from compression (YES), or no longer excluded from compression (NO).

RELEASE={percent | NO}

Specifies that any excess space should be released from the space that the VSAM cluster occupies. It also provides the percentage of the space that is to be RETAINED. The percentage is applied to the file size (high-used RBA) and the resulting value is added to the file size. The resulting value is the amount retained. If the value NO is coded, the release option is turned off for this data set.

Note: A value of 0 for the RELEASE parameter causes the default value of 10% to be used.

NEWNAME=newdata setname

Specifies the new data set name or data set pattern for a previously-defined entry. This parameter can be specified in conjunction with any other parameters when renaming a cluster or pattern entry.

Note: For data component and path names, this is the only valid parameter.

NON-COMP=#

Specifies a file's original noncompressible field length for other than custom Huffman methods instead of allowing CA Compress to calculate it from the data set's attributes. This parameter is especially helpful for exempting nonkey bytes from compression or for setting aside noncompressible bytes at the beginning of ESDS records. Do not code a value less than the end of the last key.

SCOPE=CICS|ALL

Specifies whether data set should be considered defined as compressed only under CICS (SCOPE=CICS) or always (SCOPE=ALL). This parameter is primarily intended as a conversion aid. The default is the current value in the record.

IAM=YES|NO

Changes this Control File entry to apply to IAM (Innovation Access Method) data sets as well as VSAM data sets (YES), or to not include IAM data sets (NO).

UEXIT=YES|NO

Specifies whether the Transparency User Exit is to be invoked during compression and expansion of each record.

FORCE

Allows ALTER to be done even if the data set is already compressed. Valid uses include changing values such as the noncompressible area on a new entry created from an existing entry by the COPY statement, which can appear compressed even though it is a new entry, or to correct an error in a new VSAM entry added with SCHED=SCHED. Misusing this parameter can cause compressed data not to expand, or can cause new records to be compressed differently from those already compressed in the data set.

ADD Statement (Physical Sequential)

The ADD statement for Physical Sequential defines a physical sequential data set or pattern to CA Compress, indicating that the individual data set or data sets matching the pattern should be compressed using the attributes specified.

```
ADD{PSDSN=datasetname}|{PSPATTERN=patternname}
    {,SUPEREXP|FDT=fdtname,[STANDARD]DICTIONARY=dictionary}
    {,DCBMODEL=datasetname}|{,RECFM=recfm,LRECL=lrecl,BLKSIZE=blksize}
    {,RECFM=recfm,LRECL=lrecl,BLKSIZE=blksize}
    {,EFFDATE=yyddd}|{ANYDATE}
     [,GDG=YES|NO|ONLY]
     [,SDB=YES|NO]
     [,ERASEUNCAT=YES|NO]
     [,EXCLUDE]
     [,NON-COMP=#]
     [,SCOPE=CICS|ALL]
     [,DEVTYPE=TAPE|DA|ALL]
     [,UEXIT=YES|NO]
```

PSDSN=data-setname

Specifies a physical sequential data set which is to be compressed.

PSPATTERN=patternname

Specifies a data set name pattern which selects data sets for compression. The pattern can contain any character that is valid in a data set name along with the special characters ?, !, and /. These special characters are defined in the section Syntax Rules in this chapter.

SUPEREXP|FDT=fdtname[,STANDARD]|DICTIONARY=dictionary

Selects the type of compression desired for the data set or pattern being defined.

The optional parameter STANDARD can be specified only in conjunction with the FDT=fdtname parameter. It informs CA Compress that the FDT specified should provide the compression or expansion tables only, and that the RDL specified in the FDT should be ignored. The STANDARD parameter allows one FDT to be used for a number of data sets that can share character distribution characteristics but not different noncompressible area. STANDARD is implied if the FDT parameter specifies a name of the form STDTBLxx where xx is a numeric value from 01 thru 06. The DICTIONARY parameter can specify a CA-supplied IBM Hardware Compression dictionary of the form HC#STDxx (where xx is numeric 01 to 05).

SUPEREXP specifies the low-overhead compression algorithm called Super Express.

DCBMODEL=data-setname/RECFM=recfm/LRECL=lrecl/BLKSIZE=blksize

These parameters supply the DCB attributes of the uncompressed data. These values must be kept in the Control File because they are unavailable elsewhere.

DCBMODEL specifies a cataloged data set from which to extract RECFM, LRECL, and BLKSIZE attributes. It can be the data set being compressed, or any other PS or PO disk data set, unless it is compressed but not implemented.

RECFM, LRECL, and BLKSIZE supply each individual attribute. They can be combined with DCBMODEL, in which case any individual attribute coded overrides the value implied by DCBMODEL.

For PS Patterns, these parameters are optional. DCB attributes in PS Pattern records serve only to supply defaults to matching PS Dsname records as they are created.

For PS Dsname records, these parameters are individually optional, but some combination must provide a valid RECFM, LRECL, and BLKSIZE for the data set. A consistency check is performed. If any are missing, or if in combination they are invalid, or if DCBMODEL specifies a data set which is compressed but not implemented in the Control File, the statement is rejected. For these values to be correct is vital.

EFFDATE=yyddd/ANYDATE

These parameters are optional and mutually exclusive.

EFFDATE specifies the Julian date that compression is to be implemented on the data set. When the date becomes current or past, the data set is compressed the next time it is created or replaced.

ANYDATE specifies that the data set should be considered compressed already. If it is opened for output, compressed data is written, and if it is read, each record is expanded. ANYDATE should be specified when you are implementing a data set already compressed in an earlier release of CA Compress using the SUBSYS JCL parameter; otherwise CA Compress assumes that the data is still uncompressed until the data set is recreated, and applications receive compressed data.

GDG=YES|NO|ONLY

Specifies whether pattern matches should be recognized for all data sets including GDGs (GDG=YES), excluding GDGs (GDG=NO), or only for GDGs (GDG=ONLY). The default is GDG=YES.

SDB=YES|NO

Specifies whether compressed data set BLKSIZE should be calculated by the IBM System Determined Blocksize (SDB) facility or set to the uncompressed data set BLKSIZE. SDB=YES gives better compression and I/O performance, but may cause I/O errors if a smaller BLKSIZE is coded on JCL which reads the compressed data set without CA Compress. The input BLKSIZE should never be coded except when using SUBSYS to invoke CA Compress, but until such JCL is corrected, SDB=NO will make it work correctly. The default is YES unless NOSDB is specified in the CA Compress started task JCL.

ERASEUNCAT=YES|NO

Specifies whether the entry should be purged from the Control File when the data set is uncataloged. This is an optional parameter and interacts with the GDG parameter. The default is YES for GDG=ONLY and NO for GDG=YES and GDG=NO.

EXCLUDE

Specifies that the data set is excluded from compression.

NON-COMP=#

Specifies a noncompressible area at the beginning of each record for other than custom Huffman or Tailored RDL methods. This parameter is especially helpful for setting aside noncompressible bytes to enable record selection or other processing on the data set without having to expand and compress records.

SCOPE=CICS|ALL

Specifies whether data set should be considered defined as compressed only under CICS (SCOPE=CICS) or always (SCOPE=ALL). This parameter is primarily intended as a conversion aid. The default is SCOPE=ALL.

DEVTYPE=TAPE|DA|ALL

Specifies whether data set should be considered defined as compressed only if on tape (DEVTYPE=TAPE), on direct access (DEVTYPE=DA), or anywhere. The default is DEVTYPE=ALL.

UEXIT=YES|NO

Specifies whether the Transparency User Exit is to be invoked during compression and expansion of each record. The default is NO.

ALTER Statement (Physical Sequential)

The ALTER statement for Physical Sequential changes the compression attributes of a data set or pattern that is currently defined to CA Compress.

```
ALTER {PSDSN=datasetname}|{PSPATTERN=patternname}
    {,SUPEREXP|FDT=fdtname,[STANDARD]|DICTIONARY=dictionary}
    {,DCBMODEL=datasetname}|{,RECFM=recfm,LRECL=lrecl,BLKSIZE=blksize}
    {,RECFM=recfm,LRECL=lrecl,BLKSIZE=blksize}
    {,EFFDATE=yyddd}|{ANYDATE}
     [,GDG=YES|NO|ONLY]
     [,SDB=YES|NO]
     [,ERASEUNCAT=YES|NO]
     [,EXCLUDE=YES|NO]
     [,NEWNAME=newdatasetname]
     [,NON-COMP=#]
     [,UEXIT=YES|NO]
     [,FORCE]
```

PSDSN=data-setname

Specifies the physical sequential data set entry being ALTERed.

PSPATTERN=patternname

Specifies the physical sequential data set name pattern being ALTERed. The pattern can contain any character that is valid in a data set name along with the special characters ?, !, and /. These special characters are defined in the section Syntax Rules in this chapter.

SUPEREXP|FDT=fdtname[,STANDARD]|DICTIONARY=dictionary

Selects the type of compression desired for the data set or pattern being ALTERed.

The optional parameter STANDARD can be specified only in conjunction with the FDT=fdtname parameter. It informs CA Compress that the FDT specified should provide the compression or expansion tables only, and that the RDL specified in the FDT should be ignored. The STANDARD parameter allows one FDT to be used for a number of data sets that can share character distribution characteristics but not different noncompressible area. STANDARD is implied if the FDT parameter specifies a name of the form STDTBLxx where xx is a numeric value from 01 thru 06. The DICTIONARY parameter can specify a CA-supplied IBM Hardware Compression dictionary of the form HC#STDxx (where xx is numeric 01 to 05).

SUPEREXP specifies the low-overhead compression algorithm called Super Express.

DCBMODEL=data-setname

RECFM=recfm; LRECL=lrecl; BLKSIZE=blksize

These parameters change the DCB attributes of the uncompressed data. These attributes must be kept in the Control File because they are unavailable elsewhere.

RECFM, LRECL, and BLKSIZE change each individual attribute.

You need not change them all, but the result after your specified changes must be a valid combination of RECFM, LRECL, and BLKSIZE for the data set. A consistency check is performed. If in combination they are invalid, the statement is rejected. For these values to be correct is vital.

DCBMODEL is ignored when an entry already has DCB attributes defined. For this reason, it has no effect on the ALTER statement except on PS Patterns for which no DCB attributes were defined when the Pattern was added.

EFFDATE=yyddd/ANYDATE

These parameters are optional and mutually exclusive.

EFFDATE specifies the Julian date that compression is to be implemented on the data set. When the date becomes current or past, the data set is compressed the next time it is created or replaced.

ANYDATE specifies that the data set should be considered compressed already. If it is opened for output, compressed data is written, and if it is read, each record will be expanded. ANYDATE should be specified when you are implementing a data set already compressed in an earlier release of CA Compress using the SUBSYS JCL parameter; otherwise CA Compress expects to read uncompressed records until the data set is recreated, and applications receive compressed data.

GDG=YES|NO|ONLY

Specifies whether pattern matches should be recognized for all data sets including GDGs (GDG=YES), excluding GDGs (GDG=NO), or only for GDGs (GDG=ONLY).

SDB=YES|NO

Specifies whether compressed data set BLKSIZE should be calculated by the IBM System Determined Blocksize (SDB) facility or set to the uncompressed data set BLKSIZE. SDB=YES gives better compression and I/O performance, but may cause I/O errors if a smaller BLKSIZE is coded on JCL which reads the compressed data set without CA Compress. The input BLKSIZE should never be coded except when using SUBSYS to invoke CA Compress, but until such JCL is corrected, SDB=NO will make it work correctly. The default is YES unless NOSDB is specified in the CA Compress started task JCL.

ERASEUNCAT=YES|NO

Specifies whether the entry should be purged from the Control File when the data set is uncataloged. This is an optional parameter, but it interacts with the GDG parameter. For instance, if ERASEUNCAT=NO and you specify GDG=ONLY, ERASEUNCAT is changed to YES unless you explicitly say ERASEUNCAT=NO.

EXCLUDE=YES|NO

Specifies that the data set is excluded from compression (YES), or no longer excluded from compression (NO).

NEWNAME=newdatasetname

Specifies the new data set name or data set pattern for a previously defined entry. This parameter can be specified in conjunction with any other parameters when renaming a PS data set or pattern entry.

NON-COMP=#

Specifies a noncompressible area at the beginning of each record for other than custom Huffman or Tailored RDL methods. This parameter is especially helpful for setting aside noncompressible bytes to enable record selection or other processing on the data set without having to expand and compress records. The Utility rejects this parameter if the data set is already compressed.

SCOPE=CICS|ALL

Specifies whether data set should be considered defined as compressed only under CICS (SCOPE=CICS) or always (SCOPE=ALL). This parameter is primarily intended as a conversion aid.

DEVTYPE=TAPE|DA|ALL

Specifies whether data set should be considered defined as compressed only if on tape (DEVTYPE=TAPE), only if on direct access (DEVTYPE=DA), or anywhere.

UNINHIBIT

Reactivates compression for this data set if compression has been inhibited due to an earlier error, usually a DCB parameter mismatch. The most common reason for CA Compress to inhibit an entry is that a compressed data set was expected, but the data set was actually uncompressed, perhaps because an uncompressed data set was renamed to this compressed data set name. This condition is normally corrected when the data set is recreated.

If the data set is correctly compressed, but the entry was inhibited due to a JCL error, an earlier CA Compress logic error, or some other condition, UNINHIBIT restores compression and expansion to the data set without having to recreate it. If the error is due to having used the wrong DCBMODEL data set or coding one of the other DCB parameters incorrectly in the Control File entry, you should correct these at the same time.

UEXIT=YES|NO

Specifies whether the Transparency User Exit is to be invoked during compression and expansion of each record.

FORCE

Allows ALTER to be done even if the data set is already compressed. Valid uses include changing values such as the noncompressible area on a new entry created from an existing entry by the COPY statement, which can appear compressed even though it is a new entry. Misusing this parameter can cause compressed data not to expand, or can cause new records to be compressed differently from those already compressed in the data set.

COPY Statement

The COPY statement copies an entire VSAM or Physical Sequential entry.

```
COPY    {DSNAME=datasetname,DATA=newdatacomponentname|PSDSN=datasetname}
,NEWNAME=newdatasetname
```

DSNAME=datasetname

Specifies the name of a VSAM entry to COPY to the new entry. The value should be a valid cluster name, not a data component name or physical sequential data set name. DSNAME= must be accompanied by DATA=, which specifies the DATA component name of the new entry.

DATA=newdatacomponentname

Specifies the data component of the new VSAM cluster entry specified by NEWNAME. This parameter is required when the DSNAME parameter is specified.

PSDSNAME=datasetname

Specifies the name of a Physical Sequential data set entry to COPY. DATA= should not be coded with PSDSNAME=. If DATA= is present, the CFMU ignores it and issues a warning message.

NEWNAME=newdatasetname

Specifies the new data set name entry to be created from the DSNAME or PDSNAME entry specified. For DSNAME, DATA specifies the new data component name.

DELETE Statement

The DELETE statement is used to remove the following from the Control File:

- A VSAM data set or pattern definition
- A physical sequential data set or pattern definition
- A system entry

```
DELETE      {DSNAME=data setname|PATTERN=patternname|PSDSNAME=data setname|
     PSPATTERN=patternname|SYSTEM=systemname}
    [,ONLY]
```

DSNAME=data-setname

Specifies the name of a VSAM entry to DELETE from the Control File. The value should be a valid cluster or path name, not a data component name or physical sequential data set name. If it is a cluster name and ONLY is not specified, the cluster entry and all its associations are deleted from the Control File. If it is a path entry, only the path entry is deleted.

PATTERN=patternname

Specifies the name of a VSAM pattern entry to DELETE. The entry can contain any character valid in a data set name, together with special characters ?, *, ! and/or /. These special characters are defined in the section Syntax Rules in this chapter.

PSDSNAME=data-setname

Specifies the name of a physical sequential data set to DELETE from the Control File.

PSPATTERN=patternname

Specifies the name of a physical sequential pattern to DELETE. The entry can contain any character valid in a data set name, together with special characters ?, *, ! and /. These special characters are defined in the section Syntax Rules in this chapter.

SYSTEM=systemname

Specifies the system name of a currently inactive CA Compress system that is to be deleted from the Control File. The SMF system id is used to maintain the in-storage list of patterns whenever they change. You do not need to DELETE the system name regularly because it is automatically removed when the CA Compress subsystem is brought down normally.

If a system name on the Control File is identical to the one currently starting and it has not been used for more than 2 minutes, it is reused automatically. You are queried at startup if the system entry has been used in the last 2 minutes to determine if it should be reused. You are informed that it is not possible to start CA Compress if 8 system names are in the Control File and you must add one. Then you must delete (using the DELETE statement) one of the inactive entries.

ONLY

Specifies that only the record specified in the DSNAME parameter is to be removed from the Control File, even when DSNAME is a cluster. ONLY is intended to remove orphan records in the Control File resulting from hardware or system failures.

FDT Statement

The FDT statement adds or replaces a File Descriptor Table (FDT) entry in the Control File.

```
FDT     FDTNAME=fdtname
[,REPLACE]
```

FDTNAME=fdtname

Specifies the name of the FDT that is to be added or replaced in the Control File.

REPLACE

Specifies that the FDT should be replaced if it currently exists on the Control File. The FDT is added if it does not currently exist.

REPORT Statement

The REPORT statement generates reports on one or more definitions of data sets, patterns, File Descriptor Table (FDT) entries, and Systems in the Control File.

```
REPORT  [DSNAMES={ALL|(data-set-name-1 [, . data-set-name-10])]
[,PATTERNS={ALL|(pattern-name-1 [, . pattern-name-10])]
 [PSDSNAMES={ALL|(data-set-name-1 [, . data-set-name-10])]
 [PSPATTERNS={ALL|(pattern-name-1 [, . pattern-name-10])]
 [,SYSTEMS]
 [,FDTS={ALL|(fdt-name-1 [, ... fdt-name-10])]
 [,FORMAT={SHORT|LONG|DUMP}]
```

DSNAME={ALL|data-setname(s)}

Specifies the name(s) of the VSAM cluster(s) to be reported on. The value should be a valid cluster name, path name or a data component name. The data set name can be a pattern, in which case all VSAM data set definitions matching the pattern are reported on. The value ALL specifies that all VSAM data set definitions in the Control File are to be reported.

PATTERN={ALL|patternname(s)}

Specifies the VSAM pattern name to be the subject of the report. The pattern can contain any character that is valid in a data set name, along with the special characters ?, *, ! and /. The value ALL specifies that all VSAM pattern definitions in the Control File are to be reported.

PSDSNAME={ALL|data-setname(s)}

Specifies the name(s) of the physical sequential data set(s) to be reported on. The value should be a valid discrete physical data set name. The data set name can be a pattern, in which case all PS data set definitions matching the pattern are reported on. The value ALL specifies that all PS data set definitions in the Control File are to be reported.

PSPATTERN={ALL|patternname(s)}

Specifies the physical sequential pattern name to be reported on. The pattern can contain any character valid in a data set name, along with the special characters ?, *, ! and /. The value ALL specifies that all PS pattern definitions in the Control File are reported.

SYSTEMS

Specifies that all system name records are the subject of the report.

FORMAT={SHORT|LONG|DUMP}

Specifies the format of the report. A SHORT report contains the basic information about a File Descriptor Table (FDT), data set, pattern, or system record. A LONG report for a data set or pattern record contains the size of the noncompressible area for a Scheduled, Super Express, or Standard data set, and the pattern that matched it for a pattern match data set. The DUMP format contains a dump of each record that meets the selection criteria. The default is FORMAT=SHORT.

SET Statement

The SET statement will select a data set or subsystem Control File entry upon which the subsequent statements are to act. It can also set the abend flag to abend the job if the step completes with a nonzero return code.

```
SET      [DDNAME=ddname|SUBSYS=subsystem|PRODUCTION]
[,ABEND={NO|YES}]
```

DDNAME=ddname

Specifies the DDNAME of a VSAM file that is to be used as a Control File.

SUBSYS=subsystem

Specifies the name of the CA Compress subsystem that manages the desired Control File. The subsystem specified must be active.

PRODUCTION

Specifies that a search is to be made for an active CA Compress subsystem. The located subsystem is used to access the Control File.

ABEND={NO|YES}

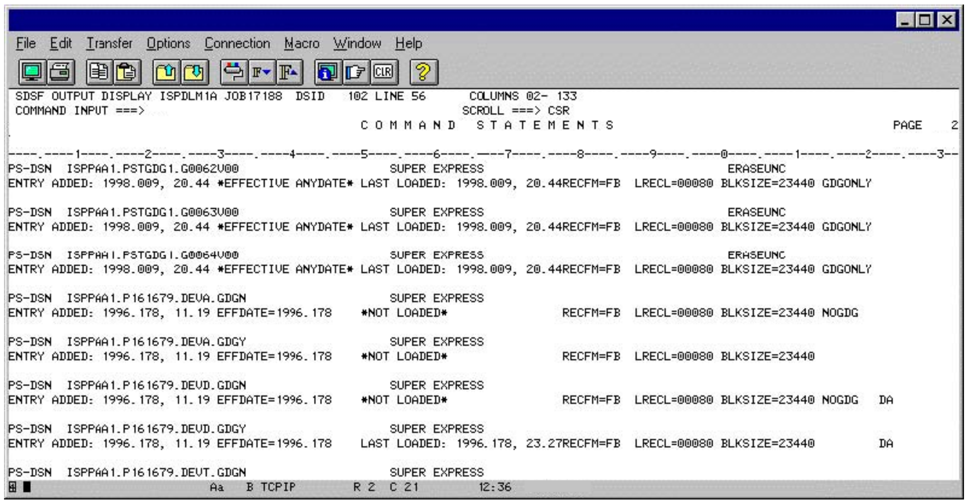
Specifies if an abend is to be generated when a nonzero return code occurs. The default is ABEND=NO.

Control File Maintenance Utility Reports

The Control File Maintenance Utility (CFMU) Report provides a list of processed control statements, information about the effects of control statement processing, and output from the REPORT control statement. This statement can be shown in short, long, or dump formats.

CFMU Short Format

The following image shows the Control File Maintenance Utility Report in short format:



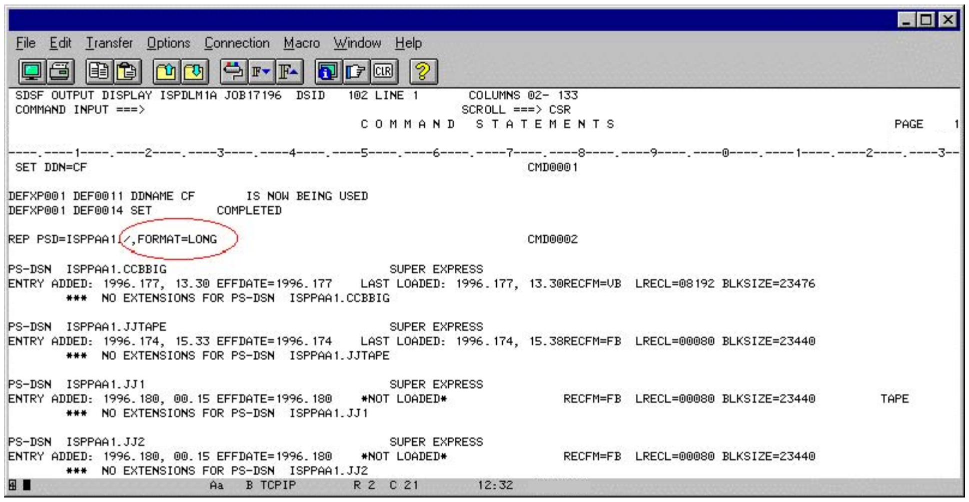
```
 SDSF OUTPUT DISPLAY ISPDLM1A JOB17188  DSID   102 LINE 56      COLUMNS 02- 133
 COMMAND INPUT ===>                                            SCROLL ===> CSR
                                                 C O M M A N D   S T A T E M E N T S                                       PAGE    2

----.----1----.----2----.----3----.----4----.----5----.----6----.----7----.----8----.----9----.----0----.----1----.----2----.----3--
PS-DSN  ISPPAA1.PSTGDG1.G0062V00                     SUPER EXPRESS                                          ERASEUNC
ENTRY ADDED: 1998.009, 20.44 *EFFECTIVE ANYDATE* LAST LOADED: 1998.009, 20.44RECFM=FB  LRECL=00080 BLKSIZE=23440 GDGONLY

PS-DSN  ISPPAA1.PSTGDG1.G0063V00                     SUPER EXPRESS                                          ERASEUNC
ENTRY ADDED: 1998.009, 20.44 *EFFECTIVE ANYDATE* LAST LOADED: 1998.009, 20.44RECFM=FB  LRECL=00080 BLKSIZE=23440 GDGONLY

PS-DSN  ISPPAA1.PSTGDG1.G0064V00                     SUPER EXPRESS                                          ERASEUNC
ENTRY ADDED: 1998.009, 20.44 *EFFECTIVE ANYDATE* LAST LOADED: 1998.009, 20.44RECFM=FB  LRECL=00080 BLKSIZE=23440 GDGONLY

PS-DSN  ISPPAA1.P161679.DEVA.GDGN                    SUPER EXPRESS
ENTRY ADDED: 1996.178, 11.19 EFFDATE=1996.178    *NOT LOADED*                 RECFM=FB  LRECL=00080 BLKSIZE=23440 NOGDG

PS-DSN  ISPPAA1.P161679.DEVA.GDGY                    SUPER EXPRESS
ENTRY ADDED: 1996.178, 11.19 EFFDATE=1996.178    *NOT LOADED*                 RECFM=FB  LRECL=00080 BLKSIZE=23440

PS-DSN  ISPPAA1.P161679.DEVD.GDGN                    SUPER EXPRESS
ENTRY ADDED: 1996.178, 11.19 EFFDATE=1996.178    *NOT LOADED*                 RECFM=FB  LRECL=00080 BLKSIZE=23440 NOGDG   DA

PS-DSN  ISPPAA1.P161679.DEVD.GDGY                    SUPER EXPRESS
ENTRY ADDED: 1996.178, 11.19 EFFDATE=1996.178    LAST LOADED: 1996.178, 23.27RECFM=FB  LRECL=00080 BLKSIZE=23440         DA

PS-DSN  ISPPAA1.P161679.DEVT.GDGN                    SUPER EXPRESS
                            Aa    B TCPIP         R 2  C 21         12:36
```

Fields and Contents—The CFMU short report contains the following information:

- Command Statement report heading, which includes: the date, day, and time that the report was generated; and the page number and CA Compress release used to generate the report.
- REPORT command followed by a statement specifying the subject of the report: data set name, pattern name, and system.
- Module name, message number, and message text.
- The detail of the item being reported: data set name, pattern name, the compression routine used or FDT name, implementation type, and exclusion indicator.

CFMU Long Format

The following image shows the Control File Maintenance Utility Report in long format:



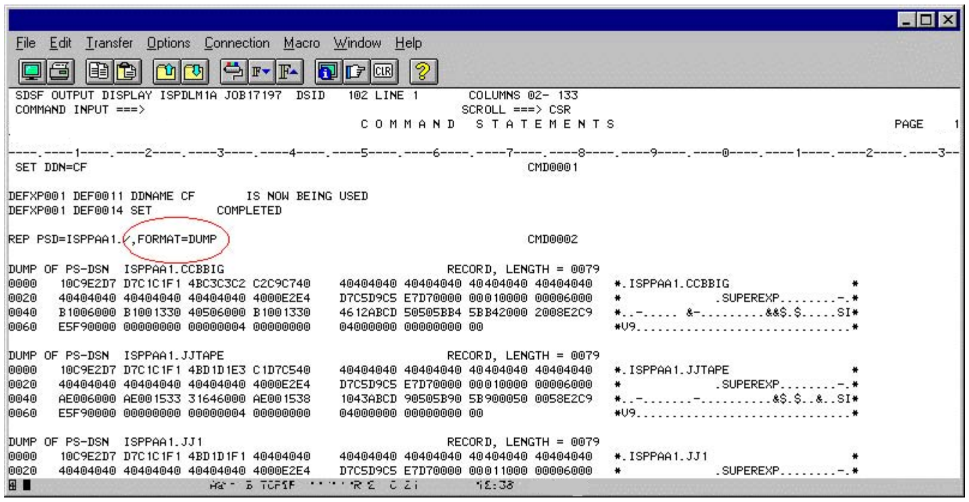
```
 SDSF OUTPUT DISPLAY ISPDLM1A JOB17196  DSID   102 LINE 1       COLUMNS 02- 133
 COMMAND INPUT ===>                                            SCROLL ===> CSR
                                       C O M M A N D   S T A T E M E N T S                                     PAGE    1

----.----1----.----2----.----3----.----4----.----5----.----6----.----7----.----8----.----9----.----0----.----1----.----2----.----3--
 SET DDN=CF                                                           CMD0001

DEFXP001 DEF0011 DDNAME CF       IS NOW BEING USED
DEFXP001 DEF0014 SET        COMPLETED

REP PSD=ISPPAA1/,FORMAT=LONG                                          CMD0002

PS-DSN  ISPPAA1.CCBBIG                             SUPER EXPRESS
ENTRY ADDED: 1996.177, 13.30 EFFDATE=1996.177    LAST LOADED: 1996.177, 13.30RECFM=VB  LRECL=08192 BLKSIZE=23476
        ***  NO EXTENSIONS FOR PS-DSN  ISPPAA1.CCBBIG

PS-DSN  ISPPAA1.JJTAPE                             SUPER EXPRESS
ENTRY ADDED: 1996.174, 15.33 EFFDATE=1996.174    LAST LOADED: 1996.174, 15.38RECFM=FB  LRECL=00080 BLKSIZE=23440
        ***  NO EXTENSIONS FOR PS-DSN  ISPPAA1.JJTAPE

PS-DSN  ISPPAA1.JJ1                                SUPER EXPRESS
ENTRY ADDED: 1996.180, 00.15 EFFDATE=1996.180    *NOT LOADED*             RECFM=FB  LRECL=00080 BLKSIZE=23440        TAPE
        ***  NO EXTENSIONS FOR PS-DSN  ISPPAA1.JJ1

PS-DSN  ISPPAA1.JJ2                                SUPER EXPRESS
ENTRY ADDED: 1996.180, 00.15 EFFDATE=1996.180    *NOT LOADED*             RECFM=FB  LRECL=00080 BLKSIZE=23440
        ***  NO EXTENSIONS FOR PS-DSN  ISPPAA1.JJ2
                            Aa    B TCPIP        R 2  C 21        12:32
```

Fields and Contents—The CFMU long report contains the following information:

- Command Statement report heading, which includes: the date, day, and time that the report was generated; and the page number and CA Compress release used to generate the report.
- REPORT command followed by a statement specifying the subject of the report: data set name, pattern name, and system.
- Module name, message number, and message text.
- The detail of the item being reported: data set name, pattern name, the compression routine used or FDT name, implementation type, and exclusion indicator. In addition, it shows the original noncompressible area of the record for a Scheduled, Super Express, or Standard data set, the pattern match time and name, and the Release percent.

CFMU Dump Format

The following image shows the Control File Maintenance Utility Report in dump format:



```
 SDSF OUTPUT DISPLAY ISPDLM1A JOB17197  DSID   102 LINE 1       COLUMNS 02- 133
 COMMAND INPUT ===>                                            SCROLL ===> CSR
                                                      C O M M A N D   S T A T E M E N T S                              PAGE    1

----.----1----.----2----.----3----.----4----.----5----.----6----.----7----.----8----.----9----.----0----.----1----.----2----.----3--
 SET DDN=CF                                                                CMD0001

DEFXP001 DEF0011 DDNAME CF       IS NOW BEING USED
DEFXP001 DEF0014 SET          COMPLETED

REP PSD=ISPPAA1./,FORMAT=DUMP                                              CMD0002

DUMP OF PS-DSN  ISPPAA1.CCBBIG                              RECORD, LENGTH = 0079
0000    10C9E2D7 D7C1C1F1 4BC3C3C2 C2C9C740    40404040 40404040 40404040 40404040   *.ISPPAA1.CCBBIG                *
0020    40404040 40404040 40404040 4000E2E4    D7C5D9C5 E7D70000 00010000 00006000   *               .SUPEREXP........-.*
0040    B1006000 B1001330 40506000 B1001330    4612ABCD 50505BB4 5BB42000 2008E2C9   *..-..... &-.........&&$.$.....SI*
0060    E5F90000 00000000 00000004 00000000    04000000 00000000 00                  *V9..............................*

DUMP OF PS-DSN  ISPPAA1.JJTAPE                              RECORD, LENGTH = 0079
0000    10C9E2D7 D7C1C1F1 4BD1D1E3 C1D7C540    40404040 40404040 40404040 40404040   *.ISPPAA1.JJTAPE                *
0020    40404040 40404040 40404040 4000E2E4    D7C5D9C5 E7D70000 00010000 00006000   *               .SUPEREXP........-.*
0040    AE006000 AE001533 31646000 AE001538    1043ABCD 90505B90 5B900050 0058E2C9   *..-.......-..........&$.$..&..SI*
0060    E5F90000 00000000 00000004 00000000    04000000 00000000 00                  *V9..............................*

DUMP OF PS-DSN  ISPPAA1.JJ1                                 RECORD, LENGTH = 0079
0000    10C9E2D7 D7C1C1F1 4BD1D1F1 40404040    40404040 40404040 40404040 40404040   *.ISPPAA1.JJ1                   *
0020    40404040 40404040 40404040 4000E2E4    D7C5D9C5 E7D70000 00011000 00006000   *               .SUPEREXP........-.*
```

Fields and Contents—The CFMU dump report contains the following information:

- Command Statement report heading, which includes: the date, day, and time that the report was generated; and the page number and CA Compress release used to generate the report.
- REPORT command followed by a statement specifying the item to be reported: data set name, pattern name, system, and FDT.
- Module name, message number, and message text.
- The dump of the FDT in the command statement.

FDT Compare Utility DEFXP050

The FDT Compare Utility compares all the FDTs in a specified load library to the CA Compress Control File being used by the active CA Compress started task. If CA Compress is not active, an error message is issued and the system abends.

The utility lists those FDTs in the Control File which are unequal to their equivalents in the load library, or which are damaged and cannot be fetched from the Control File. For each FDT not found in the Control file, the utility produces a Control File Maintenance Utility (CFMU) FDT control statement to add the FDT to the Control File. You can supply this data set to a subsequent CFMU step to add the missing FDTs.

Executing the FDT Compare Utility

To execute the FDT Compare Utility, you can customize the JCL in member FDTCOMPR in YOUR.CAI.CCVBJCL.

Additional notes for Sample JCL for the FDT Compare Utility:

- BADNAMES specifies the list of damaged or unequal FDTs. This is ordinarily a SYSOUT data set, but it can be a sequential data set which your program can use to process the problem FDTs. You can specify any BLKSIZE compatible with the utility's specification of RECFM=FB and LRECL=80.
- ADDFDTS is the sequential data set containing the CFMU FDT control statements. You can specify any BLKSIZE compatible with the utility's specification of RECFM=FB and LRECL=80.

Chapter 3: Record Definition Language

Record Definition Language (RDL) provides you with a formal means of describing characteristics of the data comprising a file that is compressed (and expanded) by the CA Compress system, using a user-generated FDT. The more you know about the data, the more detailed and precise the RDL specifications can be.

RDL is used only for user-defined FDTs. Because the execution of any RDL specification must be serialized, user-defined FDTs impose a significant I/O penalty where multiple I/Os are issued concurrently, as in busy CICS systems. Super Express, Standard Tables, and Hardware compression do not require this serialization, so we strongly recommend them where they give acceptable compression.

Even for user-defined FDTs, CA Compress assumes default record definitions (described at the end of this chapter) if you choose not to code RDL specifications. CA Compress effectiveness in achieving impressive compression ratios using default record definitions (and corresponding low processing overhead) is considerable. In the absence of other considerations (for example, the need to exempt specific nonkey fields from compression, for which user-coded RDL specifications are necessary) we recommend that the default definitions assumed by the CA Compress system be used in the first attempts to compress any file. Then, if performance is satisfactory, no more user coding is necessary.

This chapter describes how to create your own RDL, if you need to do it.

This section contains the following topics:

Performance Considerations When Using RDL

Consider three factors when you evaluate performance:

- I/O—Including the need to serialize concurrent I/Os when using any user generated FDTs.
- Compression Ratio—The amount of storage space saved by compressing a data set.
- Processing Overhead—The amount of additional CPU cycles required to transform record images from the compressed state to the uncompressed state for processing by an application program, and to retransform records to the compressed state for storage in the data set.

Whenever RDL is used, non-reentrant code is compiled to execute the compression and expansion specified by the RDL. When I/Os are performed sequentially, this consideration is negligible. However, for the CICS or other systems performing concurrent reads and writes to the same ACB, this penalty becomes severe. For any data set used under CICS or similar systems, we strongly recommend Super Express, Standard Tables, or Hardware Compression, if at all possible.

Maximizing the compression ratio represents a cost saving for users, because more data can be stored per unit of storage available. The accompanying increase in processing overhead may or may not represent an increased cost to the user, depending upon the circumstances unique to each user.

Many factors influence whether the increase in processing overhead results in a cost or a saving to the user. Although it seems that increased processing overhead to compress and expand records always costs the user more, more records can be stored per block when compressed, reducing the number of times that data blocks must be transferred between the application and the storage device. This reduction represents a decrease in processing overhead and may result in a net saving. The availability of CPU cycles during a typical job mix is also important. The total number of CPU cycles available per unit of time is a fixed cost to the user, directly related to the computing power of the user's installed hardware. If CPU cycles are available during a typical job mix, increased processing overhead may result in no increased cost.

In any event, there is a trade-off between the compression ratio and processing overhead. As the compression ratio approaches the theoretical maximum (that is, storing the greatest amount of data in the fewest number of bits), processing overhead tends to increase. By benchmark testing, you can determine the optimum trade-off for yourself, based upon your requirements.

How the RDL Operates

The theoretical maximum compression ratio is different for each data set, because it depends upon the actual data contained in the data set. The CA Compress system provides multiple algorithms for compressing data, which you can selectively apply to individual fields within data records to maximize their compression.

Using RDL specifications, which you code and supply as input to build the File Descriptor Table (FDT), accomplishes this compression. The more completely and accurately the RDL describes the records, the closer the compression ratio approaches the theoretical maximum. However, each RDL specification coded has an associated cost in processing overhead.

For example, you know that a certain field contains textual data, such as a customer name and that the customer name field never contains numeric characters. This characteristic can be used to differentiate this field from a customer address field, which, while containing textual information, does contain numeric characters. By using 2 different RDL specifications to define these fields, you may achieve a higher compression ratio for both fields than if both fields are defined by the same RDL specification.

While it is often sufficient to know what kind of data is in a field, it is also helpful to know the distribution of values contained in the field across the file. For example, one of the most efficient ways to define a field to CA Compress is as a small set of fixed expected values. A file may contain a warehouse name field, where there is only a small number of warehouses represented on the file. An RDL specification can be coded which provides these names as a set of expected values.

RDL Terminology

To understand how to use CA Compress RDL, understand certain technical terms used to describe the RDL in the following discussion. You code RDL specifications to describe the records that comprise the file that is compressed and expanded. Each RDL specification defines one data field.

Note: Unless otherwise noted, references to the term *field* in this section means a field as defined to CA Compress, not a record data element.

A field, to CA Compress, is a series of consecutive byte locations, the contents of which have similar compression/expansion characteristics, as determined by the user. The boundaries of fields defined to CA Compress need not correspond with actual field boundaries of data elements. For example, the entire record can be defined to CA Compress as a single CA Compress field.

Fields are differentiated by the type of data they contain (for example, character data, packed decimal data, and so on). Thus, each RDL specification partly consists of a field type code. From the field type code you specify, the CA Compress system selects a compression/expansion algorithm appropriate for the field's content.

In addition to the type of data contained in a field, CA Compress must know the extent of the field; that is, where it begins and ends—the boundaries of the field. Thus, each RDL specification coded contains an indication of the length of the field, in bytes.

The location in the record where the field begins is implied by the sum of the lengths of fields previously defined. CA Compress evaluates user-coded RDL specifications left-to-right and maintains an internal field pointer (IFP). The value in the IFP is initially zero, corresponding to the first position of the record. CA Compress automatically adjusts the IFP value for each RDL specification, increasing it by the length of the previous field definition.

In a few special cases, you may need to set the value of the IFP explicitly by coding a special RDL specification, the Position Function. Use of the Position Function is described later in this chapter.

Field lengths are not always fixed. The field length of a variable-length field must be determinable from information contained in the record. A separate field normally contains the length of a variable-length field, or contains a value indicating the number of times that a variably occurring fixed-length segment appears. CA Compress allows you to perform arithmetic within a field definition to calculate a variable field length, using a special symbol, VS, to represent the calculated value. The VS (Variable Symbol) can be coded in certain field definitions in lieu of an integer length. Detailed description of calculating a value for the VS is presented later in this section.

You may need to repeat an RDL specification twice or more in succession. For example, the record may contain 20 successive packed decimal numbers of identical length. To reduce coding in such cases, the RDL provides for specification of a repetition factor for a single RDL specification or a group of RDL specifications coded consecutively. This RDL specification structure has the same effect as coding the RDL specification or a group of RDL specifications as often, in sequence, as indicated by the repetition factor. This RDL specification structure is referred to as a *repetition group*. The VS can be coded in cases where the repetition factor is variable and can be calculated from information within the record. This process is described later in this section.

Files can contain multiple record formats, where the format of a particular record can be determined from the contents of 1 or more individual fields. CA Compress allows alternative record definitions, which are effective for particular records based upon the contents of a field. Such an RDL coding structure is referred to as a condition group and is described later in this section.

RDL Syntax Rules

Syntax rules for the Record Definition Language are as follows:

- Definitions appear within Columns 1—72 of each card.
- Definitions can be continued onto any number of cards.
- Each field definition consists of a 1- or 2-character type code followed by a field length descriptor.
- Definitions are separated by commas and/or blanks.
- Groups of definitions are enclosed within quotes and preceded by a repetition factor.
- Condition groups (definitions whose pertinence is dependent on record content) are enclosed in parentheses.
- Numbers appearing in the language, either as field lengths or arithmetic constants, must be between 1 and 32767 (inclusive), unless otherwise specified.
- Definitions are terminated with a period or by end-of-file on the RECDEF data set. Information appearing after the period is treated as a comment.

As the record definitions are processed by the Prepass Utility or the Interactive User Interface (IUI), they are checked for syntactical validity (but not applicability to the data) and printed. If syntax errors are encountered, each error is underscored by an alphanumeric character identifier, which corresponds to the initial character of an explanatory error message printed directly below the RDL statement in error.

Each RDL specification consists of a field type code and a field length descriptor. Valid field type codes and valid forms for coding field length descriptors are shown in the following tables.

The following example of the use of field definitions with length specifications has the following characteristics:

- The first 8-byte field is a right-justified, zoned decimal number.
- The next 100-byte field is character data.
- The next 2 fields are 4-byte packed decimal numbers.
- The remainder of the record is treated as character data.

```
ZRF8, C1F100, PDF4, PDF4, C1VER.
```

The following table describes valid field code types:

Field Type	Type of Data Defined
C1, C2, C3	Character data using internal frequency table 1, 2, or 3, as specified

Field Type	Type of Data Defined
CS	Character data using SHRVL algorithm
GA	Garbage, filler, padding, alignment bytes, and so on.
L	Insert binary length indication (for COBOL users)
MA, MB	Pattern matching
N	Exempt from compression (keys)
PD	Packed decimal data
S, X	Set of expected values
UN	Undefined field
V, VP, VZ	Variable definition
ZL, ZR	Zoned decimal, left- or right-justified, as specified

The following table describes valid field lengths descriptor codes:

Code	Description
Fn	Fixed-length field of length n, where 1<n<16384; n may contain leading zeros but may not exceed eight decimal digits.
FVS	Length determined by the previous type-V field. This descriptor is valid only for types C1, C2, C3, UN, and GA.
VER	Variable-length field extending to the end of the record. This descriptor is valid only for types C1, C2, C3, UN, and GA. This specification gives the RDL some independence from record length. In particular, the record length can be increased indefinitely without having to recreate the FDT and reimplement the data set. We strongly encourage its use.
Dc	Variable-length field delimited by a given EBCDIC character, *c*, or end of input record, whichever comes first. The length must be less than 128 bytes. This descriptor is valid only for types C1, C2, C3, UN, and GA. The following field definition, if any, begins beyond the delimiter.

RDL Field Type Descriptions

The following subsections describe individual RDL field type codes, with suggestions and restrictions concerning their use.

Field Types C1, C2, and C3—Character Data

These field types are compressed using the Huffman algorithm, coupled with elimination of successive repetitions of the same byte value. The value in each byte is assigned a variable-length bit code, with the most-frequently occurring value assigned the shortest bit code and the least-frequently occurring value assigned the longest bit code. The frequency of occurrence of each value is determined during the Prepass and is stored in 1 of 3 character frequency tables. A separate character frequency table is associated with each of the character-type RDL field specifications C1, C2 and C3. When coding RDL specifications for type C fields, you should attempt to group together in the same frequency table those fields whose byte values are likely to have a similar distribution.

For example, you can define predominantly alphabetic fields as type C1, predominantly numeric fields as C2, and fields with another kind of distribution as C3. The compression ratio thus obtained is better, at no increase in processing overhead, than if all fields are defined as the same type. With the exception that types C1, C2, and C3 have their own individual frequency table, they are treated identically by CA Compress.

For example, on a name and address file, suppose the name appears in the first 40 positions, the street address in the next 39, the city and state in the next 28, and the ZIP code in the final 5 positions. The code could be:

```
C1F112.
```

but

```
C1F40, C2F39, C3F28, C2F5.
```

gives better results. After the Prepass, the C1 table is heavily skewed toward alphabetics, with the letters M, R, S, blank and the vowels used most frequently. The more frequently the character is used, the shorter is its bit code representation. The second field is preponderantly alphabetics, numerics, and blank, so that its compression may be improved using a separate frequency table, C2. Because the last field is numeric, it can also be grouped in C2, although it probably is better for both fields to code it ZRF5. The city and state field may have the same approximate distribution as the first C1 field, but because there is one more table to spare, (C3), it should be used.

Note: In the improbable event that all 256-byte values are equally represented in the file, each character translates into an 8-bit code. But even in this case, some compression may be obtained through the type C automatic elimination of successive duplicate byte values.

Field Type CS—Character Data (SHRVL Compression)

This field type is compressed using the SHRVL (pronounced *shrivel*) algorithm. SHRVL provides better compression than Huffman, especially in circumstances where the data characteristics vary considerably from record to record.

Field Type GA—Garbage Data (Permanently Unused Fields)

Field type GA is specified when the content of a field is no longer of value in the file. This type can be specified for permanently unused fields, fillers, alignment bytes, and so on. They are deleted in the compressed record, but appear as binary zeros upon re-expansion. *If these fields were not originally zeros, the expanded record is not an exact replica of the original.* Checking used by CA Compress does not consider type GA fields.

Consider variable-length input records, where the 2 low-order bytes of the IBM standard Record Descriptor Word (RDW) are always binary zeros. Those 2 bytes can conveniently be specified as GAF2.

All 4 bytes of the RDW are sometimes superfluous in defining variable-length records, because the length may be implicit in the record content. The RDW may then be defined as GAF4. The Expansion Utility fills in the length automatically. The EXPAND subroutine supplies the length if the output record address (first parameter) and the record length address (third parameter) are the same.

Field Type L—Insert Tally of Actual Length

This type is used to insert a binary length indicator at the front of each compressed record. When issued, it must be the first specification of the record definition statements. In the compressed record it appears as a 2-byte binary number at the start of the record after the RDW, if any, and before the type N fields. The type L field contains the number of bytes in the record that follows this field. It is coded in the record definitions simply as the letter L, with no length descriptor.

Type L is useful in reading and writing compressed records in COBOL. For example, the COBOL record definition (or redefinition) for records of maximum length 1000 can be specified as:

```
01 CMP-RCD SYNC.
   02 LEN PIC 9(4) COMP.
   02 CHARS PIC X OCCURS 1000 TIMES DEPENDING ON LEN.
```

Field Types MA and MB—Pattern Matching

When data field content is similar from record to record, a pattern matching specification may produce more efficient compression than other field type specifications.

Fields defined by this field type are compared with a pattern, and matching characters, as well as character repetitions, are compressed. Type MA fields use the data in the first record as a pattern. Type MB fields use the data in the previous record as a pattern. When SHRINK or EXPAND is called from the user's program, these patterns are set during the initial CALL. Type MB patterns are reset during each subsequent CALL.

Note: Using MA and MB pattern matching causes an 0C4 abend in both CA Compress IMS and MVS Editions. For this reason, pattern matching can only be used with CA Compress/2 and its use is strongly discouraged.

Certain unavoidable restrictions apply to the use of these fields:

- They must be fixed-length.
- They cannot be specified within condition groups.
- When they appear within a repetition group or a nest of repetition groups, none of the enclosing repetition factors can be the variable symbol, VS.
- The pattern used to expand the field must be the same as that used to compress the field.

The use of type MB requires clarification. In general, the compression obtained with type MB is better than with type MA, but its usage is more restrictive. MB can only be conveniently used for sequential processing, for example, old master updated by transaction file yields new master. The following rules must be followed for this type of sequential update:

Each old master record must be passed to the EXPAND subroutine in sequence. This requirement precludes the use of MA or MB field definitions with ISAM files retrieved randomly.

Each new master record must be passed to the SHRINK subroutine in sequence.

If FDTs are in sequential data set format, the old master file and the new master file must have separate TABLxx DD statements. If the FDTs are in load module format, each master file must have its own SCB. However, these can, and usually do, refer to the same File Descriptor Table. Pattern matching works best when field values are partially or wholly repeated from record to record, as in a file containing multiple consecutive records with the same name and address. It is useful for print files, files that are transmitted, and seldom updated read-only files where the speed of compression—and especially expansion—is more important than compression ratios.

This type involves substantial setup timing overhead per field and should, therefore, not be used for fields shorter than 10 bytes, unless a considerable compression payoff is expected. The longer the field, the less processing overhead for setup is required per byte.

Field Type N—Fields Exempted From Compression

Type N fields are not compressed. They are placed at the front of each record (after the RDW, if any, and after the type L field, if any) in the same order as they are defined. Type N fields can be used as control fields for sorting, retrieving or updating the compressed record, and are required for key fields.

Type N fields are exempted from the check byte calculation. Thus, they can be modified within the compressed record without a *Check Byte Mismatch* condition occurring upon re-expansion.

Type N fields cannot appear within conditional or repetition groups. Type N fields must appear at fixed offsets from the start of the record, and their definition must apply to all records in the file. If such a field happens to occur in the record after variable-length fields, variable repetition groups, and/or condition groups, the type N field must be defined before these other fields through the use of a Position Function, which is described later in this section. The total length of all type N fields must not exceed 4095 bytes.

The decision as to whether or not to exempt a field from compression depends upon several factors. Key fields used to retrieve records from the file must be exempted from compression to enable record retrieval. Sort key fields, upon which the file is regularly sorted, and match key fields, for record matching applications, should also be exempted from compression.

When sort key fields are exempted from compression, users can invoke a sort utility program to sort the file in its compressed state, avoiding all expansion overhead. When match key fields are exempted from compression, you can avoid expansion overhead in application programs until it is determined that compressed fields from the record require processing.

Finally, any field at a fixed offset from the record origin, which appears in all records of the file, can be considered for exemption from compression. Consideration must be given to the trade-off between compression ratio and processing overhead. If the field is always or frequently operated upon in application programs, you can exempt the field from compression. Where compression ratio is critical, the field should be compressed. Where minimum processing overhead is critical, the field should probably be exempted from compression. Benchmark testing both ways enables you to determine the optimal trade-off.

Field Type PD—Packed Decimal Data

Type PD fields contain packed decimal data (USAGE COMPUTATIONAL–3 for COBOL users). The field length must be less than nine bytes. Valid fields must meet the following conditions:

- The number is between –2147483647 and 2147483647.
- The sign is a hexadecimal C, D, E, or F.
- Only decimal digits (0–9) occur. (For efficiency, type PD should not be specified for fields of lengths 1 or 2. Types C1, C2, C3, UN, X, or S can be used in these cases.)

Invalid fields are automatically treated as type UN by CA Compress. Instead of being compressed, a field defined as type PD, which contains invalid data, is enlarged by one bit. No data is lost, and upon expansion, field content is the same as it was before compression. If a field often contains invalid data, type C yields greater compression and lower processing overhead than PD. A message is printed with the statistics produced by the Compression Utility indicating the number of times invalid data was encountered in a PD field during compression.

Packed decimal numbers are converted by CA Compress to binary, bit aligned, variable-length *floating point*. If the packed decimal numbers are large in magnitude and fill their fields with significant digits (for example, the packed date 76 36 5C in a 3-byte field), then defining the field as type PD yields poor performance in both average time per byte and compression. It is better to specify a type C dedicated to these packed decimal fields. However, if the numbers rarely come close to filling the field with significant digits, type PD yields better performance than other specifications.

Specifically, fields with a value of 0 compress to 6 bits, while fields with a value between -16n-1 and 16n-1 compress to 6+4n bits, regardless of sign or field width.

Consider a file where each record consists of 20, 4-byte packed fields, either written as PDF4, PDF4, ..., 20 times, or abbreviated as a repetition group

```
20'PDF4'
```

If the average number of significant digits is 5 or greater, then it is better to define the record as

```
C1F80
```

If compression is more important than speed, then specifying

```
20'C1F3,C2F1'
```

separately defining the low-order sign bytes gives better results for the same record.

Field Types S and X—Set of Expected Values

A table reference compression technique can be used where the set of expected values contained in a field across the file is small, and these values are known. It is not necessary to include all values that occur in the field in the table of expected values. If data is encountered in the field for which no matching table entry is specified, the data is not compressed; instead, it grows by one bit. No data is lost in this case, and the expanded field is identical to the field before compression. A message is printed with the statistics produced by the Compression Utility, indicating the number of times a value was encountered in a type S or X field that was not specified in the table of expected values. To achieve efficient compression, it is important that most values occurring in a field defined as type S or X are specified in the table of expected values.

Field types S and X are functionally equivalent. The only difference is how to specify the table of expected values. The table for type S is coded in EBCDIC characters. Any of the 256 possible byte values can be coded, but nongraphic data must be multipunched. The table for type X is coded in hexadecimal format. Each byte value is coded as 2 hexadecimal digits.

RDL specifications for field types S and X are coded in a special format:

```
tmnv
```

Parameter	Description
t	The field type specification, either S or X.
m	A 2-digit number (01<m<99—code the leading zero for values between 01 and 09), indicating the field length.
n	A 2-digit number (01<n<16—code the leading zero for values between 01 and 09), indicating the number of entries in the table of expected values.
v	The table of expected values. In this table, entries are coded consecutively until n values are specified. For type S, no space can be left between consecutive entries. The table must occupy exactly m*n positions in the field definition. If the table specification continues past column 72 of the current RDL statement, it starts again in column 1 of the next RDL statement. For type X, spaces can appear between pairs of hexadecimal digits for readability, but the table must contain exactly 2*m*n hexadecimal digits.

CA Compress uses a sequential search algorithm to determine if a field value in the record appears in the table of expected values. For maximum efficiency in processing overhead, the entries of the table should be coded in decreasing order of probability of occurrence. Code first the expected value most likely to occur first; code last the expected value least likely to occur.

The only limit on the size of an expected value table, other than 99 entries maximum, is the total space available in the FDT. Where applicable, this is the most efficient method, both in terms of compression ratio and processing overhead.

To show correctly coded type S and X field definitions, and to show the difference in the way expected values are coded between types S and X, consider the following definitions, which are equivalent:

```
S0103AB1
X0103C1C2F1
```

Suppose a file has a 4-byte field containing DOGb/, CATb/, FISH, BIRD, FROG, or *other*, where *other* occurs infrequently. If this field is specified as:

```
S0405DOGbCATbFISHBIRDFROG
```

the 32-bit (that is, 4-byte) field compresses to 4 bits, one bit as an error flag and 3 bits to represent which of the 5 values occur. An *other* field cannot be compressed, but grows by one bit to 33 bits.

Field Type UN — Undefined Fields

This type is used for fields which fall into none of the other categories. Its chief use is to define fields which cannot readily be compressed (floating point, binary, bit switches, and so on), particularly when types C1, C2 and C3 are already in use.

For example, consider defining a floating point field, and types C1, C2 and C3 are already in use. Defining the floating point field as one of the C types alters the distribution of values in the corresponding character frequency table. Defining a floating point field as a type C has a detrimental effect upon the compression ratio for all other fields defined by type C. To avoid this effect, define the floating point field as type UN. Type UN fields are not compressed but do not grow in length. Processing overhead for type UN fields is minimal.

Field Types V, VP, and VZ — Calculate Variable Symbol Value

Using type V to access the RDW may obtain slightly better compression, because the RDW is excluded from analysis, but the FDT cannot be used for VSAM or other than RECFM=V(B) PS data sets, so considerable inconvenience is likely. Moreover, SUBSYS and the transparency support only V2-4 to access the RDW.

CA Compress RDL provides the capability for defining variable-length fields, and fields which occur a variable number of times, if the length of the variable-length field, or the number of times a variably occurring field is actually present, it is stored within the record or can be calculated from information stored within the record. This capability is implemented using the Variable Symbol, which is coded in RDL specifications as VS.

Field types V, VP and VZ provide the means to calculate and store a value in the VS. VS is then coded in subsequent RDL specifications as a field length, a position reference or a repetition factor. The value stored in the VS during file processing is substituted in the RDL specification in which it appears for each record for which the specification applies. The VS can be referenced multiple times within the definition of the record. The value stored in the VS is changed every time a type V, VP or VZ field is processed.

Field types V, VP and VZ are functionally equivalent. The only difference is the format of the data in the type V, VP or VZ field. Use type V to define fields containing binary integer data, type VP for fields containing packed decimal data, and type VZ for fields containing right-justified zoned decimal data.

RDL specifications for field types V, VP and VZ are coded in one of 3 possible special formats at the user's discretion: *tn; tno1i1; tno1i1o2i2* .

Parameter	Description
t	is the field type specification, either V, VP, or VZ.
n	is the length of the V, VP, or VZ field, in bytes.
	For type V: 1<n<4. For types VP and VZ: 1<n<8.
o1 and o2	+ = addition - = subtraction * = multiplication / = division
i1 and i2	are integers between 0 and 32767.

Any remainder resulting from a division operation is dropped.

Arithmetic operations are evaluated left to right. The following are examples of the special formats:

Format	Description
V4	The 4-byte field currently defined contains a binary integer whose value is to be stored in the VS. Using the VS in a subsequent RDL specification refers to the value of the binary integer.
V4+500	500 is added to the binary integer, as described in the previous example, and the result is stored in the VS.
VP3-3/20	The 3-byte field currently defined contains a packed decimal number. Three is subtracted from the number, the result is divided by 20, and the end result is stored in the VS.

The type V specification is often used in CA Compress/2 calls to process variable-length records using the RDW. For example, for a variable-length record that is treated as a single type C1 field, the record definition is written as

```
V2-4,GAF2,C1FVS.
```

The record length is picked up from the first 2 bytes of the RDW, from which the RDW length, 4, is subtracted. The next 2 unused bytes of the RDW are treated as a garbage field, while the remainder of the record is character data.

Consider a variable-length record made up of the 4-byte RDW, followed by a fixed 80-byte field that is followed by a variable number of 40-byte appendages. The fixed portion is treated as type C1, while each appendage contains 10, 4-byte packed decimal numbers. The record is defined as

```
V2-84/40,GAF2,C1F80,VS'10'PDF4''.
```

One restriction applies to type VP and VZ fields. If invalid packed decimal data is encountered in a field defined using VP, or invalid right-justified zoned decimal data is encountered in a field defined using VZ, CA Compress abends with a user code of 20, and the following message is written to the system output writer:

```
VP
INVALID TYPE VZ FIELD
```

Coding the RC parameter when calling the SHRINK or EXPAND subroutine suppresses the abend, as shown with the messages:

```
REC DEFS IMPLY WRONG LENGTH
```

and

```
CHECK BYTE MISMATCH
```

See the chapter *CA Compress/2* and the *CA Compress Data Compression Messages Guide* for more information.

Field Types ZL and ZR — Zoned Decimal Data

Field types ZL and ZR can be used to define fields which contain zoned decimal data. Type ZL defines a field containing left-justified zoned numeric data, possibly followed by 1 or more filler characters. Type ZR defines a field containing right-justified zoned numeric data, possibly preceded by 1 or more filler characters.

Valid fields must meet these conditions:

- The numeric portion must contain only digits (0–9). Commas and decimal points are not permitted.
- The value in the numeric portion must be less than 2147483648. Negative numbers are not permitted.
- The length of the field must not exceed 128 bytes.
- For ZL fields, the filler character, if any, must be blank. All-blank fields are valid, as is a field containing a single, left-justified zero, followed by all blanks.
- For ZR fields, the filler character can be either blank or zero, but not both. All-blank and all-zero fields are valid. A field containing a single, right-justified zero preceded by all blanks is valid.

Data contained in fields defined as ZL or ZR is converted to binary, bit-aligned, variable-length *floating point*. Blank or zero fields compress to 5 bits, while fields with a value between -16n-1 and 16n-1 compress to 5+4n bits, regardless of field length.

If invalid data is encountered in a field defined as ZL or ZR, the field is not compressed. Instead, it grows by one bit in the compressed record. No data is lost in this event; the expanded field contains exactly the same data as it did before compression. A message is printed with the statistics produced by the Compression Utility indicating the number of times invalid data was encountered in a ZL or ZR field during compression.

ZL and ZR specifications are most useful in cases where a separate type C specification cannot be dedicated to zoned numeric fields and/or multiple zoned numeric fields are not contiguous in the record. Consider an 80-byte record containing all numeric digits. The most efficient specification is C1F80. However, if 80 contiguous bytes containing zoned numerics occurred within a record for which all type C definitions were already in use, the best definition for the numeric data is 8'ZRF9',ZRF8. For zoned decimal fields of 9 bytes or less, a definition using type ZR or ZL is better than using a type C specification.

RDL Repetition Groups

If a sequence of 1 or more field definitions is repeated n times, it can be coded once with a repetition factor by enclosing the sequence in single quotes and preceding it with a 2-digit number, n, where 02<n<99. For example:

```
02'ZRF2,03'PDF4,C1F25''.
```

is an abbreviated form of

```
ZRF2,PDF4,C1F25,PDF4,C1F25,PDF4,C1F25,
ZRF2,PDF4,C1F25,PDF4,C1F25,PDF4,C1F25.
```

Repetition groups can be completely, but not partially, contained in conditional groups and conversely. Thus:

```
...(...03'...)...'  and   03'...(...'...)
```

are syntactical errors.

The symbol VS can be used instead of the 2-digit repetition factor to indicate that the value used is specified in a previous type V definition. VS is useful when the actual number of times that a field or a series of fields occurs within the record is variable and is contained in a separate field. For example, an invoice file consisting of variable-length records has a variably occurring series of 3 fields:

- Line item description, 20 bytes of character data
- Line item quantity, 4 bytes packed decimal data
- Line item amount, 7 bytes packed decimal data

A separate field in the record contains a value indicating the number of line items represented in the record. This field is 2 bytes in length and is zoned decimal format. The following is coded to define these fields:

```
...,VZ2,...,VS'C1F20,PDF4,PDF7' etc.
```

Field	Description
VZ2	The field containing the actual number of occurrences
VS	The Variable Symbol reference used to refer to the value (contained in the previously defined V-type field) which contains the actual number of occurrences

The characters enclosed in single quotes represent the line item fields.

RDL Condition Groups

CA Compress RDL provides the capability for defining multiple record formats for a file, where the format of an individual record can be determined from the contents of a field within the record by coding RDL specifications in a special structure, the condition group.

The general form of a condition group follows:

```
(nv,f,...,f)
```

Parameter	Description
n	A 2-digit number, indicating the length of the field that is tested.
v	A value for comparison, n bytes in length.
f	Any RDL specification or repetition group.

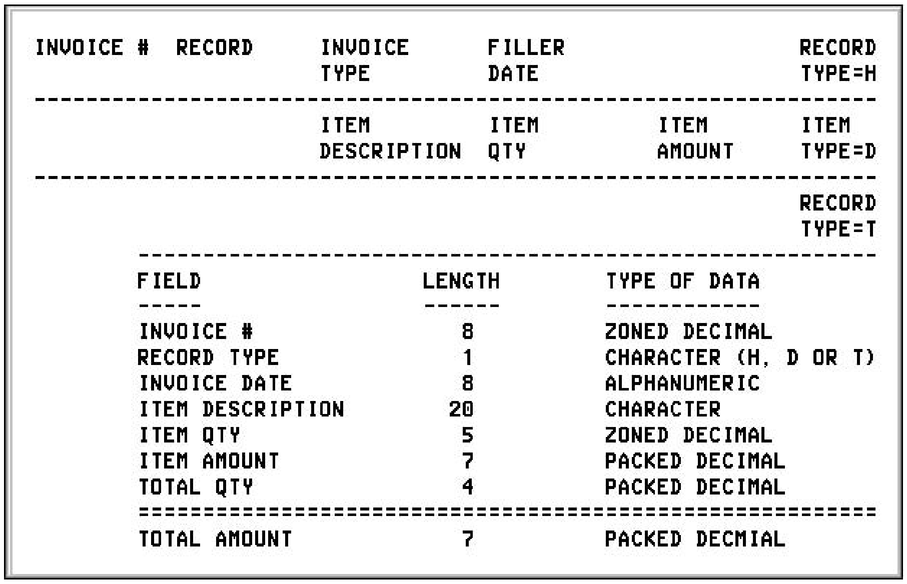
If the current n bytes in the input record being processed are equal to the value v, then the remaining definitions within the parentheses apply to the record; otherwise, they are skipped.

The value coded for v can consist of any of the 256 possible byte values, but you have to set the edit screen to hexadecimal to enter nongraphic values.

For readability, you can code the value v in hexadecimal format. To do so, code an X before the length specification n, and code the value as pairs of hexadecimal digits. For readability, leave spaces between pairs of hexadecimal digits. For example, the following 2 condition group specifications are equivalent:

```
(03XYZ,C1F80)
 (X03E7E8E9,C1F80)
```

A series of consecutively coded condition groups which are not separated by a comma (that is, separated by 1 or more blanks) indicates that the first condition group in the series whose condition is met applies to the record being processed, and the remaining condition groups in the series are skipped. For example, consider a hypothetical elementary invoicing file. Assume this file consists of sets of records, where each set of records represents 1 invoice. Each set of records consists of a header record, 1 or more detail records and a trailer record. The following image illustrates these records and the fields contained in them.



```
INVOICE #  RECORD     INVOICE      FILLER                   RECORD
                      TYPE         DATE                     TYPE=H
------------------------------------------------------------------
                      ITEM         ITEM         ITEM        ITEM
                      DESCRIPTION  QTY          AMOUNT      TYPE=D
------------------------------------------------------------------
                                                            RECORD
                                                            TYPE=T
        ----------------------------------------------------------
```

FIELD	LENGTH	TYPE OF DATA
INVOICE #	8	ZONED DECIMAL
RECORD TYPE	1	CHARACTER (H, D OR T)
INVOICE DATE	8	ALPHANUMERIC
ITEM DESCRIPTION	20	CHARACTER
ITEM QTY	5	ZONED DECIMAL
ITEM AMOUNT	7	PACKED DECIMAL
TOTAL QTY	4	PACKED DECIMAL
TOTAL AMOUNT	7	PACKED DECMIAL

The following RDL specifications define these records and show the use of condition groups:

```
ZRF8,(01H,C1F8,GAF23)b(01D,C2F20,PDF4,PDF7)b(01T,GAF20,PDF4,PDF7).
```

It is important that alternative condition groups in a series be separated from one another by 1 or more blanks—not a comma. The comma is used to separate the last condition group in a series from any RDL specifications (which could be another condition group) that follow.

To show this point, consider the sample RDL specifications above, describing the records in Invoicing File Record Set. Suppose a comma separated the first condition group from the second condition group. Then if a record whose RECORD TYPE field contained an H was encountered, the first condition group applies. But after processing the C1F8,GAF23 RDL specifications, the following condition groups are not skipped. CA Compress expects more data beyond the *filler* at the end of the record, looking to compare for a *D,* according to the next condition group. Because the record definitions do not accurately define the record that is processed, unpredictable results may occur.

If the file contains a record whose record type is not H, D or T, CA Compress will abend with a user code of 15 and the message *REC DEFS IMPLY WRONG LENGTH.* This occurs because no record definitions are specified past the invoice number for any record whose record type is neither H, D nor T. You can avoid this situation by coding a default condition group at the end of the condition group series.

It is permissible (and frequently necessary) to code a default condition group at the end of a series of condition groups. Such coding supplies a set of record definitions if none of the preceding condition group tests are met and the actual content of the byte(s) in the record being tested is not known. The general form of a default condition group is the following: (00,f,...,f). In this expression:

Parameter	**Description**
00	is coded exactly as shown.
f	is any RDL specification or repetition group.

A condition group coded in this form means that the RDL specifications in the condition group apply, no matter the contents of the current byte in the record being processed. Any default condition group coded in a series of condition groups must be coded as the last in the series. To show this procedure, the following definition can be coded to describe the records in Figure 2-1. Invoicing File Record Set.

```
ZRF8,(01H,C1F8,GAF23)b(01D,C2F20,PDF4,PDF7)b(01T,GAF20,PDF4,PDF7)b(00,C3F32).
```

If you omit coding a default condition group as the last condition group in a series, CA Compress automatically supplies the default condition group *(00)*. This procedure indicates that if none of the preceding condition groups apply for a particular record, any RDL specifications following the condition group series apply beginning at the current byte location in the record. This is the same byte location within the record which the preceding series of condition groups tested. To show this point, note that the following 2 sets of RDL specifications are equivalent:

```
ZRF8,(01H,C1F8,GAF23)b(01D,C2F20,PDF4,PDF7)b(01T,GAF20,PDF4,PDF7)b(00,C3F32).ZRF8
,(01H,C1F8,GAF23)b(01D,C2F20,PDF4,PDF7,)b(01T,GAF20,PDF4,PDF7),C3F32.
```

The maximum number of condition groups that can be coded in a series is 16, including the final default condition group, whether user-specified or automatically provided by CA Compress. The maximum number of condition group series that can be coded is limited only by the amount of space available in the FDT.

If the condition specified in a condition group is met, the value specified in the condition group and found in the record is compressed to 4 bits, regardless of the length of the value. No separate RDL specification is used to compress the value. The lengths of the values can differ within a condition group series.

RDL Position Function

As fields are processed, CA Compress automatically adjusts an internal field pointer (IFP) to the current displacement within the record. In a few special cases, you can alter this IFP with the Position Function.

For example, fields exempted from compression (that is, fields defined with the field type-N RDL specification) must be defined before any variable or condition group RDL specifications. If a field exempted from compression is located at a higher displacement from the record origin than variable-length fields or conditionally present fields, the Position Function must be used to set the IFP at the field exempted from compression, so it can be defined first. Then the Position Function must be used again to reset the IFP to the lower displacement so the variable-length and/or conditionally present fields can be defined.

The Position Function has 4 possible forms, chosen at the user's discretion:

Form	Description
Pn	Set IFP to n.
P+n	Add n to the IFP.
P-n	Subtract n from the IFP.
P	Reset IFP to prior value.

P is coded as shown; n is either a 1- to 5-digit integer or the Variable Symbol, VS.

Displacements are computed relative to 0, which indicates the start (origin) of the record. The 4 Position Function forms perform the following functions:

Form	Description
Pn	repositions to the n+1th byte in the record.
P+n	repositions forward n bytes.
P-n	repositions backwards n bytes.

P resets the IFP to its value immediately preceding the last Pn, P+n or P-n. If there were no previous Position Functions executed, the P is ignored. P cannot be specified as the initial Position Function. You must adhere to the following rules:

- The IFP must remain within the bounds of the record. This error is usually caught in the Prepass or compression phase, but not always, because complete checking involves substantial time overhead during compression.
- The IFP must be at the byte following the last byte of the record after all fields are processed. Otherwise, a wrong length record abend 15 occurs.
- If a field is bypassed, a check byte mismatch abend 10 may occur on re-expansion.
- Redefining through repositioning degrades performance and should be specified only when absolutely necessary.

For example, a hypothetical name and address file contains 3 types of 80-byte records as shown below.

The following RDL specifications define this file:

```
P79,(01A,P,C1F79)b(01B,P,C2F79)b(01C,P,C3F74,ZRF5),P+1
```

RDL Specification	Description
P79,	sets the IFP at the RECORD TYPE field. (Note that the IFP is relative to zero, thus IFP of zero is the first record position, and IFP of 79 is the 80th byte of the record.)
(01A,P,C1F79)b	If the RECORD TYPE field contains A, resets the IFP to the beginning of the record, and defines the NAME field.
(01B,P,C2F79)b	If the RECORD TYPE field contains B, resets the IFP to the beginning and defines the STREET ADDRESS field.
(01C,P,C3F74,ZRF5),	If the RECORD TYPE field contains C, resets the IFP to the beginning and defines the CITY, STATE and ZIP CODE fields.

RDL Specification	Description
P79,	sets the IFP at the RECORD TYPE field. (Note that the IFP is relative to zero, thus IFP of zero is the first record position, and IFP of 79 is the 80th byte of the record.)
P+1	At this point, one of the condition groups has been applied to the record (provided that all records in the file contain either A, B or C in the RECORD TYPE field), and thus the IFP is again pointing at the RECORD TYPE field. It is necessary to code P+1 to position the IFP at the byte location following the last byte of the record. Failure to do so results in an abend with a user code of 4 and the message, *REC DEFS IMPLY WRONG LENGTH*.

General Restrictions on RDL Use

The RDL is employed to construct the File Descriptor Table (FDT). The FDT has a maximum size, and there is a corresponding upper limit on the number of RDL specifications allowed for any one file. In the unlikely event that this maximum is reached, a user abend 4 occurs and the RDL specifications must be reduced by combining adjacent field definitions to form group fields. The space in the FDT required to contain the RDL specifications can be calculated using the following table.

RDL Specification	Space Required (In Bytes)
GA, PD, ZL, ZR	6
C1, C2, C3, CS	18
MA, MB	92 for first occurrence. 54 for each subsequent occurrence
UN	14
N	20
v	48
VP, VZ	40
S, X	26
First *(* in a condition group series	72

RDL Specification	**Space Required (In Bytes)**
Each remaining *(* in a condition group series	54
Fn, where n < 128	4
Fn, where n > 128	38
FVS	46
VER, Dc	88
P	12
P + n	18
P + VS	8
Fixed repetition group	16
Variable repetition group	30

The space required must be summed according to RDL specifications as coded, and the total cannot exceed 2800. Consider the following example:

```
V2-4,GAF2,C1FVS.
```

Using the following table, the space required for these RDL specifications is as follows and well within the 2800 limit:

V2-4		GA		F2		C1		FVS		total
48	+	6	+	4	+	18	+	46	=	122

Guide to Correct RDL Specifications

The following table describes the correct RDL specifications:

Field Type	Description
C1, C2, C3	Type C is used to define groups of fields whose byte values have similar frequency distributions. Up to 3 different frequency distributions can be accommodated, one each by type C1, C2 and C3. If no other type code is clearly preferable, choose a type C. Compression is variable, depending on the skewedness of distribution. The more greatly skewed the distribution, the greater the compression ratio. Processing overhead is minimal (3).a
CS	Type CS is used to define groups of fields where the data varies considerably. Compression is variable, according to the data characteristics. Processing overhead is minimal (3).
GA	Type GA is used to eliminate unneeded fields from the compression record. Compression is 100 percent. Processing overhead is negligible (1).a
L	Type L is used to insert a binary tally of the compressed actual number of bytes comprising the compressed record as the first 2 bytes of the record following the RDW. Particularly useful for COBOL users. This field type actually increases the compressed record length by 2 bytes. Processing overhead is negligible (1).a
MA, MB	Type M is used when data in a field repeats from record to record. Several restrictions govern use of this field type. For fields smaller than 10 bytes in length, type C is preferable. Compression is variable, depending upon the degree of data repetition. Processing overhead is variable, decreasing as field length increases (3–5).a
N	Type N is used to exempt a field from compression. Use for retrieval, match and sort keys, and any field which you want to access without expanding the data. There are several restrictions governing use of this field type specification. This field type yields no compression. Processing overhead is negligible (1).a

Field Type	Description
PD	Use type PD for fields containing packed decimal data, preferably with many high-order zero digits. If significant digits frequently fill the field or if invalid data is frequently present in the field, choosing a type C specification is preferable. Compression is excellent when the value is zero, and variable, increasing as the proportion of significant digits to total digits decreases. Processing overhead is moderate (5–6).a
S, X	Use type s or x when the number of values occurring in a field is small (16 or fewer) and these values are known in advance. Compression is excellent, and the processing overhead is minimal (3).a
UN	Use Type UN for fields which cannot readily be compressed (for example, bit switches, floating point numbers), particularly when all 3 type C specifications are already in use. This field type yields no compression. Processing overhead is minimal (2).a
V, VP, VZ	Use one of these field type specifications to define a field whose content is used to calculate the actual length of a variable-length portion of the record, or a field which contains the actual length of a variable-length portion of the record. Type V is not compressed. Type VP is compressed as PD. Type VZ compressed as ZR. Processing overhead is minimal to moderate, depending on field type: V=2 VZ=4 VP=6a
ZL, ZR	Type Z is used for fields containing zoned decimal data, particularly when a type C specification cannot be dedicated to zoned decimal data. Several restrictions govern the use of this field type specification. Compression is excellent when value is zero, variable, increasing as the magnitude of the value increases. Processing overhead is moderate (3–4).a

a. Numbers shown are a relative measure of processing efficiency. 1=most efficient, 6=least efficient.

RDL Defaults

If the RECDEF DD statement is not present in the execution JCL for the File Prepass Utility or by the IUI, or if it specifies a null data set (that is, one with no records), default RDL specifications are generated based upon characteristics of the data set defined by the INFILE DD statement. The defaults are also generated if the only RDL specification supplied by the user is a single type L field.

Note: If *L.* is specified, all dfaults begin with *L,...*.

In the default RDL specification formulas shown below, the following variables are substituted with appropriate values, obtained from the data set label or JCL specifications:

Variable	Description
x	The number of bytes before the key, excluding the RDW (if present)
x'	x-1
y	The number of bytes following the key; if there is no key, then y is the record length (LRECL)
k	The number of bytes in the key (KEYLEN)
k'	k+1
j	k + the relative key position (RKP)

The generated defaults are printed on the PRINT data set by the File Prepass Utility or the IUI.

	Files	Formulas for Default RDL Specifications
Sequential	Fixed-length	C1Fy
	Variable-length	V2-4,GAF2,C1FVS
	Undefined	C1VER
ISAM	Fixed-length, key at beginning of record	Nk,,C1Fy
	Fixed-length, RKP=1a	Nk',,C2Fy
	Fixed-length, RKP>1a	N1,,C1Fx',,Nk,,C2Fy
	Fixed-length, key at end of record	N1,,C1Fx',,Nk

	Files	Formulas for Default RDL Specifications
	Variable-length, relative key position = 4	V2-j,,GAF2,,Nk,,C2FVS
	Variable-length, RKP=5a	V2-j,,GAF2,,Nk',,C2FVS
	Variable-length, RKP>5a	V2-j,,GAF2,,N1,,C1Fx', Nk,,C2FVS
VSAM	Fixed-length, key at beginning of record	Nk,,C1Fy
	Fixed-length, key somewhere within the record	N1,,C1Fx',,Nk,,C2Fy
	Fixed-length, key at end of record	N1,,C1Fx',,Nk
	Variable-length, key at beginning of record	Nk,,C2VER
	Variable-length, RKP=1a	Nk',,C2VER
	Variable-length, RKP>1a	N1,,C1Fx',,Nk,,C2VER

a. Default definition permits record deletion when the DCB parameter OPTCD=L is specified.

Determining the Best Compression

To determine the best possible compression to implement for a file, you can use the facilities of the IUI. Follow the steps below to determine the best compression obtainable for the file. These steps comprise a summary procedure.

1. Analyze the characteristics of the file.
2. Use the *b* subcommand on the data set in the worklist to browse the statistics.

You now have a basis on which you can judge the effectiveness of the RDL for each record. Fine-tuning the RDL for a record may increase compression but may increase the CPU overhead.

Note: While RDL is the acronym for Record Definition Language, the term *the RDL* is commonly used to mean the set of RDL statements which represent the record definition (for compression purposes) for a given data set or pattern. The RDL is located in the Analysis File as an entity within the record for a discrete file or pattern entry and is in the external character format. The RDL is also a subsection of an FDT. The RDL portion of an FDT is in the internal format and describes the record format for the file to be processed under control of this FDT.

You can respecify the RDL for the record and perform the testing procedure again. If you change the RDL for a record, the Byte Distribution Analysis (BDA) for the record is performed again.

As you fine-tune the RDL for the records in the file, follow the steps below, which can be repeated as often as needed until you have obtained the compression that you want to implement for the file.

1. Update the RDL you have devised for the record.
2. Redo the data set analysis.
3. Evaluate the results of the test compression.

When the testing and evaluating have produced the optimum compression controls, follow the step-by-step procedure outlined earlier in this section to implement compression for the file.

How to Enter or Change the RDL Using the IUI

Before actually implementing a data set, you can devise different RDL specifications in order to achieve better compression. To devise these specifications, you must update the RDL for a data set and then run a trial compression job. Detailed instructions on how to update the RDL follow.

1. Select Maintenance from the Task menu.
2. Select Analysis File to display the Analysis File Maintenance menu.
3. Select All Records or Limit Search.
   - All Records displays a list of all the data set names that are in the Analysis File.
   - Limit Search displays the Data set Name selection window. Type the data set name for the search, and press [Enter] to see a list of the data set names you selected.
4. Move the cursor to the Action field next to the data set you want.
5. Type R, and press [Enter] to display the RDL User Parameter Maintenance screen.

   Initially, each noncompressible field (type *N*) represents 1 or more keys. The definition for the *N* fields cannot be shortened or eliminated but can be increased in length to cover more data. Consider doing this if, for example, there is one byte between 2 type *N* fields. The other fields can be altered as needed. Any altered definition must not alter the total length of the definition as this can change the offset of the type *N* fields and thus the key positions, causing the Dialog to reject the RDL. You can now enter the RDL for the compressible area.
6. Enter up to 12 lines of 78 characters each for the data set's RDL. When all changes to the RDL are made, press [Enter] to store the new RDL. If you need to cancel the changes you made in the RDL, press the [End] PF key, which returns you to the Analysis File Maintenance screen.

   If you have made any errors in the RDL, an error message is displayed in the upper right-hand corner of the screen and the cursor appears at the location of the error. To obtain more information about the error, press the [Help] PF key. Correct the error in the RDL and press [Enter].

   When the RDL is correct, it is recorded and you are returned to the Analysis File Maintenance screen.

Chapter 4: CA Compress/2

The CA Compress/2 Data Compression System enables your applications to invoke subroutines to compress and expand data, reducing the cost of storing data on disk or tape and supporting compression exits to a wide variety of products. In addition, utilities can be used to compress or expand the following:

- Entire key-sequenced or entry-sequenced VSAM files.
- Entire sequential files.
- Entire direct access files and members of partitioned data sets when being processed sequentially.
- CA Compress/2 is provided to all CA Compress users.

This section contains the following topics:

Features

The CA Compress/2 system provides a unique combination of capabilities:

High Degree of Data Compression

Data sets compressed by CA Compress/2 require as little as 10 percent, and commonly 35 percent, of their original space allocation.

Efficient Implementation

Application programs using the CA Compress/2 system to process compressed data sets may in some cases complete in less time than was once required to process uncompressed data sets.

Demonstrated Reliability

CA Compress/2 uses a check byte technique to ensure that no data is altered during compression and expansion.

Ease of Use

CA Compress/2 provides excellent compression without requiring you to provide information about the structure and content of records to be compressed. If you want to achieve compression ratios approaching the theoretical maximum, CA Compress/2's Record Definition Language (RDL) provides the opportunity to describe the record structure and content of the data set to be compressed. In the absence of user-coded RDL specifications, CA Compress/2 assumes standard defaults based upon attributes of the input data set to achieve excellent compression.

Flexibility

The RDL provides a powerful tool for constructing a compressed data set tailored to your specific requirements. User-specified fields can be exempted from compression, allowing sorting of compressed data sets without requiring prior re-expansion. Application programs can expand records selectively by examining fields exempted from compression, avoiding needless expansion overhead.

If the fields being sorted on are exempt from compression, sort utility programs can sort the records without expansion and recompression.

Similarly, if match key fields are exempted from compression, application programs containing logic to match records based on match key fields can perform the matching logic on the compressed records, limiting record expansion to only those cases which require further processing.

The RDL enables you to exempt fields from compression as well as to optimize compression. Fields in the record are defined by field type code specifications, which indicate the method of compression for the defined field. The available type codes let you define virtually any combination of field formats. Using the RDL is optional. If you omit RDL specifications, CA Compress/2 assumes default RDL specifications based upon data set attributes provided by JCL or by the data set label.

Using Subroutines

CA Compress/2 provides the following subroutines which can be invoked from your application programs for processing of compressed data:

- Custom Compression
- Standard Tables Compression
- IBM Hardware Compression
- Super Express Compression

Custom Compression

The following are the Custom Compression subroutines:

SHRINK

Converts an uncompressed record image to compressed form.

EXPAND

Converts a compressed record image to its original uncompressed form.

CLOSE

Frees storage dynamically acquired by calls to SHRINK or EXPAND when it is no longer required.

Standard Tables Compression

The following are the Standard Tables Compression subroutines:

Note: Standard Tables STDTBL0x must be available through STEPLIB, JOBLIB or the linklist, or you must link edit any routine you require into your program.

SHRINKS

Converts an uncompressed record image to compressed form using a Standard Table.

EXPANDS

Converts a record image compressed with a Standard Table to its original uncompressed form.

CLOSES

Frees the dynamic area acquired by SHRINKS or EXPANDS.

IBM Hardware Compression

The following lists IBM Hardware Compression subroutines:

Note: The needed IBM Hardware Compression dictionary must be available through STEPLIB, JOBLIB or the linklist. CA strongly recommends that your dictionaries be kept in the linklist. The CA supplied dictionaries are named HC#STDnn where nn is numeric 01–99.

SHRKHCS

Converts an uncompressed record image to compressed form using an IBM Hardware Compression dictionary.

SHRKHCX

Converts a record image compressed with an IBM Hardware Compression dictionary to its original uncompressed form.

SHRKHCC

Frees the dynamic area acquired by SHRKHCS or SHRKHCX. Because another application in the address space may be using the dictionary, SHRKHCC does not issue DELETE to remove it from storage. You can do so yourself when you no longer need it.

If you supply your own dictionary, it must be named HC#USRnn where nn is numeric 00—27. The compression dictionary must be immediately followed by the expansion dictionary in one load module. Both the compression and expansion routines expect the whole module and fail if both dictionaries are not provided in this order.

Super Express Compression

The following are the Super Express Compression subroutines:

Note: SHRINKZ and EXPANDZ do not acquire any dynamic storage, so there is none to free.

SHRINKZ

Converts an uncompressed record image to compressed form using the Super Express string compression algorithm.

EXPANDZ

Converts a record image compressed using Super Express to its original uncompressed form. EXPANDZ also expands records compressed with the old EXPRESS algorithm.

These subroutines are fully reentrant, and can reside in the Link Pack Area, LINKLIB, or a load module PDS defined through STEPLIB or JOBLIB in the JCL.

JCL Implications for Existing Application Programs

When CA Compress/2 subroutines are called by an application program to process compressed data sets, corresponding changes must be made to the JCL which defines the compressed data sets. The definition of the compressed version of the data set must be substituted for the definition of each formerly uncompressed data set.

Information needed to code the JCL for the compressed data set is available from the data set label or, if the Compression Utility was used, from the Compression Utility job which compressed the data set. In general, the RECFM is variable and the LRECL must be increased by 8 bytes (12 if originally fixed length). See the section *JCL Defaults* in this chapter for detailed information.

When	How Much
Always	+2
Type L defined	+2
VER length descriptor defined	+2
RDW present (not RECFM=U, F, or VSAM)	+4
Each type GA definition	-field length
Each non-GA field redefined using the Position function	+field length

Accessing the FDT

The File Descriptor Table (FDT), created by Prepass or the Interactive User Interface, contains all the information (code tables, edited record definitions, file characteristics, and so on) needed to compress and expand the data it was built from. Each file processed by CA Compress/2 must have an associated FDT unless Super Express is being used. The FDT associated with a particular file must be present (through JCL specification) in every job step in which records from that file are processed by CA Compress/2.

Note: We strongly encourage you to use FDTs created by the IUI rather than Prepass, especially for users of Physical Sequential Transparency. The IUI and Prepass with SS05300 create FDTs with an Integrity Check Block (ICB), by which the expansion routines recognize uncompressed records and avoid a number of problems, including accidental double expansion.

The Super Express, Standard Tables, and Hardware Compression algorithms do not refer to user-defined FDTs, but the SHRINK, EXPAND, and CLOSE routines do refer to the user-defined FDT associated with each compressed data set. The FDT for each compressed data set that is processed must be defined or made accessible via the JCL used to invoke the application program which uses CA Compress/2 subroutines.

FDTs maintained in sequential data set format require separate definitions on a TABLxx DD statement, where xx, coded as a two-digit number ranging from 00 through 31, corresponds to a file number supplied by the user as a parameter in the CALL to SHRINK, EXPAND, or CLOSE. CA discourages use of this format.

FDTs should preferably be stored as members of a PDS in load module format. Coding the STEPLIB DD statement to define the PDS containing FDTs in load module format makes every FDT contained in the PDS accessible to the application program. In this manner, all FDTs used in an application program can be accessed through a single DD statement.

Within the application program, each FDT in load module format must have an associated 48-byte, fullword-aligned Shrink Control Block (SCB). Subroutine calls pass the SCB as a parameter to identify the FDT, instead of the file number as with FDTs in sequential data set format. Before the first subroutine CALL using a particular FDT, its associated SCB must be initialized as follows:

- The first 8 bytes must contain the member name of the FDT load module, left-justified with trailing blanks as necessary to fill the full 8 bytes.
- The remaining 40 bytes must contain binary zeros.
- The following model definitions demonstrate properly initialized SCBs:

Assembler Language:

```
SCBNAME DS 0F
DC CL8'fdt-member-name'
DC 10F'0'
```

COBOL—must be in WORKING—STORAGE SECTION:

```
01 SCBNAME.
 03 FILLER PICTURE X(8) VALUE 'fdt-member-name'.
 03 FILLER PICTURE X(40) VALUE LOW-VALUES.
```

PL/I:

```
DCL 01 SCBNAME,
 03 name CHAR(8) INIT ('fdt-member-name'),
 03 N(10) FIXED BIN(31) INIT ((10)0);
```

After initialization, the SCB must not be modified in any way. Reentrant programs must place the SCB in dynamically allocated main storage.

JCL Implications for Existing Application Programs

When CA Compress/2 subroutines are called by an application program to process compressed data sets, corresponding changes must be made to the JCL which defines the compressed data sets. The definition of the compressed version of the data set must be substituted for the definition of each formerly uncompressed data set.

Information needed to code the JCL for the compressed data set is available from the data set label or, if the Compression Utility was used, from the Compression Utility job which compressed the data set. In general, the RECFM is variable and the LRECL must be increased by 8 bytes (12 if originally fixed length). See the section JCL Defaults in this chapter for detailed information.

When	How Much
Always	+2
Type L defined	+2
VER length descriptor defined	+2
RDW present (not RECFM=U, F, or VSAM)	+4
Each type GA definition	-field length
Each non-GA field redefined using the Position function	+field length

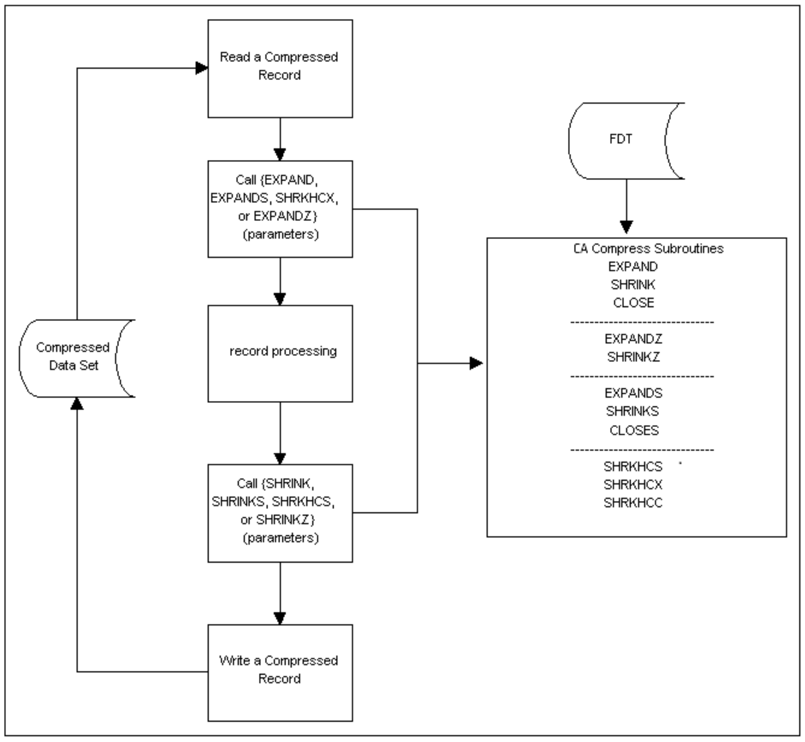
Calling the Subroutines

The application program performs all I/O. The first call to SHRINK or EXPAND which refers to a particular file number or SCB causes 1K to 8K bytes of dynamic storage to be acquired, the FDT to be loaded, and code to perform the function to be custom compiled. Subsequent calls execute only the custom-compiled code. This provides the fastest possible execution time.

SHRINKZ and EXPANDZ do not require this initial processing.

SHRINKS and EXPANDS build a small dynamic area, and depending on how they are called, they may need to load the appropriate Standard Table.

SHRKHCS and SHRKHCX also build a small dynamic area, and they need to load the appropriate hardware compression dictionary load module. The following illustration shows the general flow of subroutines.



The language dependent conventions for parameter passing and subroutine linkage are described in the following sections.

Assembler Language

The subroutines should be called using the CALL macro with the VL option or an equivalent method. Register 13 must point to a valid 18-word save area and register 1 must point to the parameter list.

COBOL

Normal COBOL CALL statements are used. Parameters are data element names of the areas being passed to the appropriate subroutine. The following considerations must be carefully observed:

SHRINK and EXPAND subroutines

COBOL programs calling the CA Compress subroutines cannot be compiled with the DYNAM option. DYNAM causes unpredictable results for the SHRINK and EXPAND subroutines. Calls to SHRINK or EXPAND pass control to the SHRINK or EXPAND utility, respectively, instead of the SHRINK or EXPAND entry point in the subroutines, resulting in message SHR005I for SHRINK or SHR050I for EXPAND.

This restriction does not apply to the Super Express subroutines, SHRINKZ and EXPANDZ; to the Standard Tables subroutines, SHRINKS and EXPANDS; or to the Hardware Compression subroutines, SHRKHCS and SHRKHCX.

OCCURS DEPENDING ON clause

Because compressed data sets are variable length, to write compressed records of the proper length requires use of the OCCURS DEPENDING ON clause. To avoid data loss and other problems, carefully adhere to the COBOL rules.

Any data beyond the length defined by the OCCURS DEPENDING variable is undefined, so any data placed there by a READ or a call to a CA Compress/2 subroutine may be destroyed. Accordingly, before reading, compressing, or otherwise moving data of an unknown length, always set the variable to its maximum. Afterwards use the data to set it to the correct length before writing.

The variable must be set explicitly by a COBOL statement that names the variable in order for the new value to be recognized by COBOL. In particular, reading a record into an area that contains the variable or changing its value by passing it to a called routine does not cause COBOL to recognize the new value.

Failing to observe these precautions can lead to data loss. For instance, you may successfully compress a record containing more characters than the OCCURS DEPENDING ON variable provides for. COBOL processing then may destroy the extra characters and the damaged record is successfully written. Months later, the record cannot be expanded, and the data is lost.

PL/I Optimizing Compiler

The compiler must be informed that the CA Compress/2 subroutines are Assembler subroutines by declaring all entry points, as follows:

```
DCL (SHRINK,EXPAND,SHRINKZ,EXPANDZ,SHRINKS,EXPANDS,CLOSE)
 OPTIONS (ASM INTER);
```

Any unused subroutines can be omitted. The CA Compress/2 subroutines can then be called using the standard PL/I CALL statement.

As in the case of COBOL, dynamic calls to SHRINK and EXPAND invoke the utilities, causing unpredictable results.

CALL to Subroutine SHRINK

Each CALL to SHRINK compresses one record image in main storage. There are five parameters (four required and one optional) which the user codes in a SHRINK CALL. The parameters are positional and must be coded in the same sequence as shown in below. The following are model statements for calling SHRINK:

Assembler Language

```
CALL SHRINK,(URA,CRA,URL,FDT[,RC]),VL
```

COBOL

```
CALL 'SHRINK' USING URA CRA URL FDT [RC].
```

PL/I Optimizer

```
CALL SHRINK(URA CRA URL FDT [RC]);
```

The following table describes the parameters available to the SHRINK Subroutine:

Parm	Meaning
URA	Uncompressed Record Address (Required) The address of the in-core record to be compressed. The SHRINK subroutine does not alter this area. If the record is variable length, URA is the address of the RDW.
CRA	Compressed Record Address (Required) The address of a user-supplied main storage area at least 290 bytes larger than the uncompressed record. The SHRINK subroutine places the compressed record in this area in variable length format, including the standard 4-byte RDW. If the length exceeds 32K, the RDW is x'80' followed by a 3 byte length.

Parm	Meaning
URL	Uncompressed Record Length (Required) The address of a halfword or fullword in which the user supplies the length of the uncompressed record in binary form. The fullword format is specified by setting the first byte to x'80' and supports data up to 24 megabytes in length. If variable-length records are being compressed, URL should be the same as the URA parameter, the address of the RDW.
FDT	File Descriptor Table Identifier (Required) If the FDT is in sequential data set format, FDT is the address of a binary fullword containing the file number (a value between 0 and 31). This value must correspond to the value coded for the xx in the TABLxx DD statement, which defines the FDT. If the FDT is in load module format, the File Descriptor Table Identifier is the address of the SCB associated with this file.
RC	Return Code (Optional) The address of a binary fullword in which the SHRINK subroutine can store a value to indicate whether an error occurred in compressing the record. If this optional parameter is coded, certain ABEND conditions are suppressed, and a return code value is stored in the binary fullword addressed by RC and in register 15. If no error occurs during processing by the SHRINK subroutine, register 15 is zero upon return to the user's program, and RC is set to zero.

CALL to Subroutine EXPAND

Each call to EXPAND returns one compressed record image in main storage to its original uncompressed form. There are five parameters (four required and one optional) which you code in an EXPAND CALL. The parameters are positional and must be coded in the same sequence as they appear above. The following are model statements for calling EXPAND:

Assembler Language

```
CALL EXPAND,(URA,CRA,URL,FDT,[,RC]),VL
```

COBOL

```
CALL 'EXPAND' USING URA CRA URL FDT [RC].
```

PL/I Optimizer

```
CALL EXPAND(URA CRA URL FDT [RC]);
```

The following table describes the parameters available to the EXPAND Subroutine:

Parm	Meaning
URA	Uncompressed Record Address (Required) The address of the main storage area where the EXPAND subroutine places the uncompressed record image which it constructs. If the record was a variable-length record before compression, URA should be the address of the RDW of the expanded record image. The storage area provided by the user must be large enough to contain the entire expanded record.
CRA	Compressed Record Address (Required) The address of the compressed record image to be expanded by the subroutine. CRA must be the address of the data portion of the record, not the RDW. This storage area, that is, the compressed record, is not modified by the EXPAND subroutine.
URL	Uncompressed Record Length (Required) The address of a halfword or fullword from which EXPAND receives the length of the compressed record in binary form and into which the subroutine returns the length of the expanded record in binary form. The subroutine does not always require the compressed length, but you should always provide it. Setting the first byte to x '80' specifies the fullword format. If the record was a variable-length non-VSAM record before compression, URL may be identical to the URA parameter.

Parm	Meaning
FDT	File Descriptor Table Identifier (Required) The address of a binary fullword which contains the file number, a value between zero and 31, if the FDT is in sequential data set format. If the FDT is in load module format, the File Descriptor Table Identifier is the address of the SCB associated with this file.
RC	Return Code (Optional) The address of a binary fullword in which the EXPAND subroutine can store a value to indicate whether or not an error occurred in expanding the record. If this optional parameter is coded, ABEND 10 (check byte mismatch) and certain other user abends are suppressed. Instead, the corresponding completion code is stored in the binary fullword addressed by the RC parameter and in register 15. If no error occurs during EXPAND processing, register 15 and RC are set to zero.

CALL to Subroutine CLOSE

Each call to the CLOSE subroutine frees all main storage obtained to support one FDT-compressed data set. Application programs normally call CLOSE only if all processing is complete for the compressed file in question, or if the storage needs to be released for other processing.

CA Compress/2 obtains main storage for each data set compressed using an FDT for the FDT, the compression and expansion tables, and the custom-compiled code. The amount of storage obtained for a particular file varies, depending upon the extent and complexity of record definitions. It can vary from about 2K to as much as 24K per file, though approximately 6K is usual.

Once CLOSE is called for a file, the FDT, compression and expansion tables and custom-compiled code to perform subroutine functions for the file are no longer available. Any subsequent call referencing that data set causes the FDT and compression and expansion tables to be brought into storage again, and custom code to be recompiled. CLOSE requires one parameter—the File Descriptor Table Identifier. If the FDT is in sequential data set format, this is the address of a binary fullword containing the file number. If the FDT is in load module format, the File Descriptor Table Identifier is the address of the SCB associated with this data set. See *Accessing the FDT* in this chapter for more information.

The following are model statements for calling CLOSE:

Assembler Language

```
CALL CLOSE,(FDT),VL
```

COBOL

```
CALL 'CLOSE' USING FDT.
```

PL/I Optimizer

```
CALL CLOSE(FDT);
```

CALL to Subroutine SHRINKS

Each CALL to SHRINKS compresses one record image in main storage. There are five parameters (four required and one optional), which you code in a SHRINKS CALL. The parameters are positional and must be coded in the same sequence as shown in Table 5-4. The following are model statements for calling SHRINKS:

Assembler Language

```
CALL SHRINKS,(URA,CRA,URL,WRK[,RC]),VL
```

COBOL

```
CALL 'SHRINKS' USING URA CRA URL WRK [RC].
```

PL/I Optimizer

```
CALL SHRINKS(URA CRA URL WRK [RC]);
```

The following table describes the parameters available to the SHRINKS Subroutine:

Parm	Meaning	
URA	Uncompressed Record Address (Required) The address of the in-core record to be compressed. The SHRINKS subroutine does not alter this area. If the record is variable length, URA is the address of the actual data, not the RDW.	
CRA	Compressed Record Address (Required) The address of a user-supplied main storage area at least 290 bytes larger than the uncompressed record. The SHRINKS subroutine places the compressed data in this area.	
URL	Uncompressed Record Length (Required) The address of a halfword or fullword in which the user supplies the length of the uncompressed record in binary form. The fullword format is specified by setting the first byte to x'80' and supports data up to 24 megabytes in length.	
WRK	Standard Tables Work Area (Required) The address of a 64-byte work area.	
	Bytes	**Meaning**
	0–7	*STDTBL01* through *STDTBL06* as appropriate.

Parm	Meaning	
	8–11	Address of the selected Standard Table. If the desired Standard Table is already link edited or LOADed, insert its address here before the first call. If the address is zero, routine will load it and fill in its address for you. An invalid nonzero value will cause an abend or other unpredictable results, so you must set it either to zero or to the valid table address.
	12–13	Noncompressible area in binary format. If there is no noncompressible area, this value must be zero.
	14–15	Compressed data length (excluding RDW) if the length is <32K, or -1 if the length is >32K. The length is also returned in register 0 in all cases. Add 4 to this length to build an RDW if required.
RC	Return Code (Optional) The address of a binary fullword in which the SHRINKS subroutine can store a value to indicate whether an error occurred in compressing the record. If this optional parameter is coded, certain ABEND conditions are suppressed, and a return code value is stored in the binary fullword addressed by RC and in register 15. If no error occurs during processing by the SHRINKS subroutine, register 15 is zero upon return to the user's program, and RC is set to zero. If the data does not compress, the return code is -1, and the uncompressed data is copied to the compressed data location.	

CALL to Subroutine EXPANDS

Each call to EXPANDS returns one compressed record image in main storage to its original uncompressed form. There are five parameters (four required and one optional) which you code in an EXPANDS CALL. The parameters are positional and must be coded in the same sequence as they appear in below. The following are model statements for calling EXPANDS:

Assembler Language

CALL EXPANDS,(URA,CRA,URL,FDT,[,RC]),VL

COBOL

CALL 'EXPANDS' USING URA CRA URL FDT [RC].

PL/I Optimizer

CALL EXPANDS(URA CRA URL FDT [RC]);

The following table describes the parameters available to the EXPANDS Subroutine:

Parm	Meaning	
URA	Uncompressed Record Address (Required) The address of the main storage area where the EXPANDS subroutine places the uncompressed data which it constructs, excluding any RDW. The storage area provided by the user must be large enough to contain the entire expanded record.	
CRA	Compressed Record Address (Required) The address of the compressed data to be expanded by the subroutine. CRA must be the address of the data portion of the record, not the RDW. This storage area, that is, the compressed record, is not modified by the EXPANDS subroutine.	
URL	Uncompressed Record Length (Required) The address of a halfword or fullword from which EXPANDS receives the length of the compressed record in binary form and into which it returns the length of the expanded record in binary form. Setting the first byte to x '80' specifies the fullword format.	
WRK	Standard Tables Work Area (Required) The address of a 64-byte work area.	
	Bytes	**Meaning**
	0–7	*STDTBL01* through *STDTBL06* as appropriate.

Parm	Meaning	
	8–11	Address of the selected Standard Table. If the desired Standard Table is already link edited or LOADed, insert its address here before the first call. If the address is zero, routine will load it and fill in its address for you. An invalid nonzero value will cause an abend or other unpredictable results, so you must set it either to zero or to the valid table address.
	12–13	Noncompressible area in binary format. If there is no noncompressible area, this value must be zero.
RC	Return Code (Optional) The address of a binary fullword in which the EXPANDS subroutine can store a value to indicate whether or not an error occurred in expanding the record. If this optional parameter is coded, ABEND 10 (check byte mismatch) and certain other user abends are suppressed. Instead, the corresponding completion code is stored in the binary fullword addressed by the RC parameter and in register 15. If no error occurs during EXPANDS processing, register 15 and RC are set to zero. If the *compressed* data is not compressed, RC and register 15 are set to -1. You can consider the *compressed* data to be the uncompressed data.	

CALL to Subroutine CLOSES

CLOSES frees the small dynamic area acquired by the SHRINKS or EXPANDS subroutines. This call should be used by online applications when they finish processing. CLOSES accepts one parameter—the work area (WRK) used by SHRINKS or EXPANDS.

The following are model statements for calling CLOSES:

Assembler Language

```
CALL CLOSES,(WRK),VL
```

COBOL

```
CALL 'CLOSES' USING WRK.
```

PL/I Optimizer

```
CALL CLOSES(WRK);
```

CALL to Subroutine SHRKHCS

Each CALL to SHRKHCS compresses one record image in main storage. There are five parameters (four required and one optional) which the user codes in a SHRKHCS CALL. The parameters are positional and must be coded in the same sequence as shown in below. The following are model statements for calling SHRKHCS:

Assembler Language

```
CALL SHRKHCS,(URA,CRA,URL,WRK[,RC]),VL
```

COBOL

```
CALL 'SHRKHCS' USING URA CRA URL WRK [RC].
```

PL/I Optimizer

```
CALL SHRKHCS(URA CRA URL WRK [RC]);
```

The following table describes the parameters available to the SHRKHCS Subroutine:

Parm	Meaning
URA	Uncompressed Record Address (Required) The address of the uncompressed data to be compressed by the subroutine, excluding any RDW. This storage area (that is, the uncompressed record) is not modified by the SHRKHCS subroutine.

Parm	Meaning	
CRA	Compressed Record Address (Required) The address of a user-supplied main storage area where the SHRKHCS subroutine places the compressed data. The storage area must be large enough to contain the entire compressed record and does not include the RDW.	
URL	Uncompressed Record Length (Required) The address of a halfword or fullword from which SHRKHCS receives the length of the uncompressed record in binary form. Setting the first byte to x '80' specifies the fullword format.	
WRK	Hardware Compression Work Area (Required) The address of a 60-byte work area.	
	Bytes	**Meaning**
	0–7	*HC#STDnn* where nn is 01 thru 99 for a CA supplied dictionary, or *HC#USRnn* where nn is 00 thru 27 for a user defined dictionary.
	8–11	Address of the selected dictionary load module. Although only the expansion dictionary is used, the compression dictionary must be the first half of the load module. If the desired dictionary is already LOADed, insert its address here before the first call. If the address is zero, the subroutine will load it and fill in its address for you. An invalid nonzero value will cause an abend or other unpredictable results, so you must set it either to zero or to the valid dictionary address.
	12–14	ICB. Set this to zero for the first call to permit the subroutine to calculate it for you.
	15	Dictionary size. Set this to zero for the first call to permit the subroutine to calculate it for you.
	16–17	Noncompressible length if any, or zero.

Parm	Meaning	
	18–20	Return code from CSRCMPSC macro.
	24–27	Dynamic area address. Set to zero for first call.
RC	Return Code (Optional) The address of a binary fullword in which the SHRKHCS subroutine can store a value to indicate whether or not an error occurred in expanding the record. If this optional parameter is coded, certain user abends are suppressed. Instead, the corresponding completion code is stored in the binary fullword addressed by the RC parameter and in register 15. If no error occurs during SHRKHCS processing, register 15 and RC are set to zero. If the data does not compress, the return code is -1, and the uncompressed data is copied to the compressed data location.	

CALL to Subroutine SHRKHCX

Each CALL to SHRKHCX returns one compressed record image in main storage to its original uncompressed form. There are five parameters (four required and one optional) which you code in a SHRKHCX CALL. The parameters are positional and must be coded in the same sequence as shown in the following table. The following are model statements for calling SHRKHCX:

Assembler Language

```
CALL SHRKHCX,(URA,CRA,URL,WRK[,RC]),VL
```

COBOL

```
CALL 'SHRKHCX' USING URA CRA URL WRK [RC].
```

PL/I Optimizer

```
CALL SHRKHCX(URA CRA URL WRK [RC]);
```

The following table describes the parameters available to the SHRKHCX Subroutine:

Parm	Meaning
URA	Uncompressed Record Address (Required) The address of the main storage area where the SHRKHCX subroutine places the uncompressed data which it constructs, excluding any RDW. The storage area provided by the user must be large enough to contain the entire expanded record.

Parm	Meaning	
CRA	Compressed Record Address (Required) The address of the compressed data to be expanded by the subroutine. CRA must be the address of the data portion of the record, not the RDW. This storage area (that is, the compressed record) is not modified by the SHRKHCX subroutine.	
URL	Uncompressed Record Length (Required) The address of a halfword or fullword from which SHRKHCX receives the length of the compressed record in binary form and into which it returns the length of the expanded record in binary form. Setting the first byte to x '80' specifies the fullword format.	
WRK	Hardware Compression Work Area (Required) The address of a 60-byte work area.	
	Bytes	**Meaning**
	0–7	*HC#STDnn* where nn is 01 thru 99 for a CA supplied dictionary, or *HC#USRnn* where nn is 00 thru 27 for a user defined dictionary.
	8–11	Address of the selected dictionary load module. Although only the expansion dictionary is used, the compression dictionary must be the first half of the load module. If the desired dictionary is already LOADed, insert its address here before the first call. If the address is zero, the subroutine will load it and fill in its address for you. An invalid nonzero value will cause an abend or other unpredictable results, so you must set it either to zero or to the valid dictionary address.
	12–14	ICB. Set this to zero for the first call to permit the subroutine to calculate it for you.
	15	Dictionary size. Set this to zero for the first call to permit the subroutine to calculate it for you.

Parm	Meaning	
	16–17	Noncompressible length if any, or zero.
	18–20	Return code from CSRCMPSC macro.
	24–27	Dynamic area address. Set to zero for first call.
RC	Return Code (Optional) The address of a binary fullword in which the SHRKHCX subroutine can store a value to indicate whether or not an error occurred in expanding the record. If this optional parameter is coded, ABEND 10 (check byte mismatch) and certain other user abends are suppressed. Instead, the corresponding completion code is stored in the binary fullword addressed by the RC parameter and in register 15. If no error occurs during SHRKHCX processing, register 15 and RC are set to zero. If the *compressed* data is not compressed, RC and register 15 are set to -1. You can consider the *compressed* data to be the uncompressed data.	

CALL to Subroutine SHRKHCC

SHRKHCC frees the small dynamic area acquired by SHRKHCS or SHRKHCS subroutines. This call should be used by online applications when they finish processing. SHRKHCC accepts one parameter—the work area (WRK) used by SHRKHCS or SHRKHCS. The following are model statements for calling SHRKHCC:

Assembler Language

```
CALL SHRKHCC,(WRK),VL
```

COBOL

```
CALL 'SHRKHCC' USING WRK.
```

PL/I Optimizer

```
CALL SHRKHCC(WRK);
```

CALL to Subroutine SHRINKZ

Each CALL to SHRINKZ compresses one record image in main storage. There are five parameters (four required and one optional) that the user codes in a SHRINKZ CALL. The parameters are positional and must be coded in the same sequence as shown in the following table. The following are model statements for calling SHRINKZ:

Assembler Language

```
CALL SHRINKZ,(URA,CRA,URL,WRK[,RC]),VL
```

COBOL

```
CALL 'SHRINKZ' USING URA CRA URL WRK [RC].
```

PL/I Optimizer

```
CALL SHRINKZ(URA CRA URL WRK [RC]);
```

The following table describes the parameters available to the SHRINKZ Subroutine:

Parm	Meaning
URA	Uncompressed Record Address (Required) The address of the in-core record to be compressed. The SHRINKZ subroutine does not alter this area. If the record is variable length, URA is the address of the actual data, not the RDW.
CRA	Compressed Record Address (Required) The address of a user-supplied main storage area at least 8 bytes larger than the uncompressed record. The SHRINKZ subroutine places the compressed data in this area.
URL	Uncompressed Record Length (Required) The address of a halfword or fullword in which the user supplies the length of the uncompressed record in binary form. The fullword format is specified by setting the first byte to x'80' and supports data up to 24 megabytes in length.
WRK	Super Express Work Area (Required) The address of a 40-byte work area. The first 2 bytes must be the length of the noncompressible area in binary format. If there is no noncompressible area, this value must be zero. The next 2 bytes are set to the compressed data length (excluding RDW) if the length is <32K, or -1 if the length is >32K. The length is also returned in register 0 in all cases. Add 4 to this length to build an RDW if required.

Parm	Meaning
RC	Return Code (Optional) The address of a binary fullword in which the SHRINKZ subroutine can store a value to indicate whether an error occurred in compressing the record. If this optional parameter is coded, certain ABEND conditions are suppressed, and a return code value is stored in the binary fullword addressed by RC and in register 15. If no error occurs during processing by the SHRINKZ subroutine, register 15 is zero upon return to the user's program, and RC is set to zero. If the data does not compress, the return code is -1, and the uncompressed data is copied to the compressed data location.

CALL to Subroutine EXPANDZ

Each call to EXPANDZ returns one compressed record image in main storage to its original uncompressed form. There are five parameters (four required and one optional) that the user codes in an EXPANDZ CALL. The parameters are positional; they must be coded in the same sequence as they appear in the following table. The following are model statements for calling EXPANDZ:

Assembler Language

```
CALL EXPANDZ,(URA,CRA,URL,WRK,[,RC]),VL
```

COBOL

```
CALL 'EXPANDZ' USING URA CRA URL WRK [RC].
```

PL/I Optimizer

```
CALL EXPANDZ(URA CRA URL WRK [RC]);
```

The following table describes the parameters available to the EXPANDZ Subroutine:

Parm	Meaning
URA	Uncompressed Record Address (Required) The address of the main storage area where the EXPANDZ subroutine places the uncompressed data which it constructs. The storage area provided by the user must be large enough to contain the entire expanded record.
CRA	Compressed Record Address (Required) The address of the compressed data to be expanded by the subroutine. CRA must be the address of the data portion of the record, not the RDW. This storage area, that is, the compressed record, is not modified by the EXPANDZ subroutine.

Parm	Meaning
URL	Uncompressed Record Length (Required) The address of a halfword or fullword from which EXPANDZ receives the length of the compressed record in binary form and into which it returns the length of the expanded record in binary form. Setting the first byte to x '80' specifies the fullword format. The subroutine uses the compressed record length to make sure the compressed record ends where expected. If it does not, a check byte mismatch condition is forced.
WRK	Super Express Work Area (Required) The address of a 40-byte work area. The first 2 bytes must be the noncompressible area in binary format. If there is no noncompressible area, this value must be zero.
RC	Return Code (Optional) The address of a binary fullword in which the EXPANDZ subroutine can store a value to indicate whether or not an error occurred in expanding the record. If this optional parameter is coded, ABEND 10 (check byte mismatch) and certain other user abends are suppressed. Instead, the corresponding completion code is stored in the binary fullword addressed by the RC parameter and in register 15. If no error occurs during EXPANDZ processing, register 15 and RC are set to zero. If the *compressed* data is not compressed, RC and register 15 are set to -1. You can consider the *compressed* data to be the uncompressed data.

Incorporating Subroutine Calls in Existing Application Programs

In most cases, incorporating CA Compress/2 subroutine calls in an application program requires only minor changes. DD statements which omit the DCB subparameters need not be changed, because the data set label supplies the correct values. Attributes explicitly specified on a DD statement or in a DCB macro, COBOL FD or PL/I file definition must be changed to conform to the compressed data set's attributes.

During compression, except by Super Express, the compressed record image may temporarily grow up to 290 bytes longer than the uncompressed record. Consequently, the area supplied for the compressed record must be 290 bytes longer than the uncompressed record. After compression is complete, the compressed record is almost never greater than eight bytes longer than the uncompressed record, even in worst-case situations.

If input records are processed in locate mode, further adjustments are necessary. In Assembler Language, the general register used with the DSECT must be loaded with the address of the uncompressed record image after calling EXPAND, EXPANDS, SHRKHCX, or EXPANDZ in order to make the program consider the uncompressed record to be the record just read. In COBOL, the entire uncompressed record definition should be moved from the FILE SECTION to the WORKING-STORAGE SECTION. It is then a simple matter to insert the proper call to EXPAND or EXPANDZ immediately following the READ (or GET) for the compressed file. Few if any logic changes should be necessary following the subroutine CALL. Calls to SHRINK, SHRINKS, SHRKHCS, and SHRINKZ can be handled in a similar way.

Defining Compressed Records in COBOL Application Programs

Compressed records are variable-length records. This fact requires special attention in COBOL application programs, because COBOL does not support variable length data conveniently. For this reason, it is especially advisable with COBOL to use the CA Compress Transparency or SUBSYS and not CA Compress/2.

Variable-length records are defined by the OCCURS DEPENDING ON data-name clause, where the element defined by *data-name* contains a value which indicates the actual number of characters in the variable- length record. The SHRINK subroutine (but not SHRINKZ or SHRINKS, which do not use RDL) generates this value and places it in the compressed record in response to a field type L record definition, but COBOL does not recognize this value until it is explicitly placed in data-name by a COBOL statement. Field type L does not support records larger than 32K.

The following procedure is recommended for altering a COBOL application program to process a compressed file.

1. FOR SHRINK, use the field type L RDL specification as the first, or only, RDL specification when the file is compressed. This specification causes CA Compress/2 to store the actual data length of each variable-length compressed record as a two-byte binary field at the beginning of each compressed record. For SHRINKZ, define a field in front of the compressed data and set it to the length of the compressed data that follows. In both cases, remember to move it to the OCCURS DEPENDING ON variable—to itself if necessary—to let COBOL know that the value has changed.
2. Move the definition(s) of the uncompressed data records from the FILE SECTION to the WORKING-STORAGE SECTION.
3. Change RECORDING MODE from F to V or S, as necessary.
4. Supply the following as the data record in the FILE SECTION for the compressed file (update the FD to refer to this record definition as DATA RECORD):
   ```
   01 SHRUNK-RECORD.
   03 LENGTH PICTURE 9(4) USAGE COMPUTATIONAL.
   03 SHRUNK-DATA PICTURE X OCCURS n TIMES DEPENDING ON LENGTH.
   ```
   Substitute the maximum record length for the value n.
5. Move the maximum value to LENGTH before the READ to ensure that SHRUNK-RECORD is long enough to hold all the bytes that are read.
6. Code a call to EXPAND, EXPANDS, SHRKHCX, or EXPANDZ, as appropriate, and insert it immediately after the READ for the file in question. Code SHRUNK-RECORD as the CRA parameter in the call to the expansion routine. Code the data record name of the record definition moved to the WORKING-STORAGE SECTION as the URA parameter in the call to the expansion routine.
7. If issuing calls to the SHRINK subroutine, provide the following compressed record area in the WORKING-STORAGE SECTION (not the FILE SECTION):

```
01 COMPRESS-AREA.
03 RDW PICTURE 9(5) USAGE COMPUTATIONAL.
03 SHRUNK-RECORD-OUT.
05 LENGTH-OUT PICTURE 9(4) USAGE COMPUTATIONAL.
05 SHRUNK-DATA-OUT PICTURE X OCCURS n TIMES
   DEPENDING ON LENGTH-OUT.
```

The user must substitute the value of the OUTFILE LRECL increased by 290 for the value n, and must explicitly move this value to LENGTH-OUT before the SHRINK call to ensure that COMPRESS-AREA is long enough. Code COMPRESS-AREA as the second (CRA) parameter of the SHRINK subroutine call. If the output record definition in the FILE SECTION is coded:

```
01 UPDATED-SHRUNK-RECORD.
03 UPDATED-LENGTH PICTURE 9(4) USAGE COMPUTATIONAL.
03 UPDATED-SHRUNK-DATA PICTURE X OCCURS n TIMES
       DEPENDI      NG ON UPDATED-LENGTH.
```

The updated compressed record is written as follows:

MOVE LENGTH-OUT TO UPDATED-LENGTH.
WRITE UPDATED-SHRUNK-RECORD FROM SHRUNK-RECORD-OUT.

8. Compile using the NOTRUNC option.

Linking Subroutines With Applications

The Super Express subroutines, SHRINKZ and EXPANDZ; the Standard Tables subroutines, SHRINKS and EXPANDS; and the Hardware Compression subroutines, SHRKHCS and SHRKHCX, can be included explicitly via the linkage editor INCLUDE statement or resolved implicitly from SYSLIB.

CA Compress/2 provides two methods to link edit the SHRINK, EXPAND and CLOSE subroutines with application programs. In the first method, SHRINK, EXPAND, and CLOSE are link edited with the user's application program by including the module SHRKEXPD using an explicit INCLUDE linkage editor control statement. This method results in making a copy of the subroutines a part of the application program load module.

The second method provides an improved capability for link editing the SHRINK, EXPAND and CLOSE subroutines with the user's application program. The subroutines are link edited by including the module SHRKSTUB via an explicit INCLUDE linkage editor control statement. This module contains entry points for SHRINK, EXPAND and CLOSE and functions as a loader. Because SHRKSTUB does not support the Super Express entry points, SHRINKZ and EXPANDZ; the Standard Tables entry points, SHRINKS and EXPANDS; or the Hardware Compression entry points, SHRKHCS and SHRKHCX, this method is not available for Super Express, Standard Tables, or Hardware Compression.

The first CALL to a particular subroutine causes SHRKSTUB to load the correct subroutine from the CA Compress/2 library. Subsequent calls to that subroutine result in a direct branch to the previously loaded subroutine with several important advantages:

- Because copies of CA Compress/2 modules are not link edited into multiple application programs, library space is conserved.
- Application programs do not have to be relink edited when a new release of CA Compress/2 is installed. The subroutine support can reside on a single shared library.
- Application programs link edited with the SHRKEXPD module must be relinked each time a CA Compress subroutine is changed. Programs link edited with the SHRKSTUB module require no modification when a subroutine is changed because the fixed module is loaded from the CA Compress/2 library at execution.
- The subroutines can be shared between tasks. Attached modules are link edited only with the SHRKSTUB module (approximately 200 bytes).

To use this method, the CA Compress/2 library must either be in the linklist or supplied through STEPLIB or JOBLIB.

Using CA Compress/2 Under CICS

Because the Super Express, Standard Tables, and Hardware Compression subroutines do not load FDTs or acquire storage, CICS applications can use them freely, in just the same way that batch programs do. The only additional consideration is that you MUST pass the return code parameter in order to prevent the subroutines from issuing the ABEND macro when compression or expansion is unsuccessful.

Because the use of FDTs involves program and storage management, CA Compress/2 provides special facilities to enable CICS to support the SHRINK and EXPAND subroutines. The CA Compress/2 for CICS facility supports any mainframe IBM operating system that supports the CICS Command Level interface.

Install the Callable SHRINK Subroutines for CICS

CA Compress, because it fully supports data sets within a CICS region, replaced these SHRINK subroutines. In the past, CA Compress provided compression functions in CICS environments by calling these subroutines. Now, only the rarest circumstances justify their use. If you must use them, complete the steps in this section to call the SHRINK subroutines directly from programs running under CICS. Except as otherwise noted, all members referenced are in the *YOUR.CAI.CCVBSAMP* library.

- Set the CWASHRK value in member SHRKWORD to point to a reserved fullword in the CWA. Change the member FDTNAMES to list the FDTs to be used.
- Assemble and link edit CA Compress/2 source modules. Modify and use the JCL MEMBER INSTALL in the CICS CA Compress/2 source library.
- Modify the CICS tables appropriately. Do both items below:
    - Define the SHRNKMOD and SHRKSCBS modules and all FDT modules to be used in the CICS Processing Program Table (PPT).
    - Define SHRNKMOD in the CICS Program List Table (PLT).
- Modify the application programs calling SHRINK/2 CICS subroutines and link edit them with the SHRKCICS module.

These steps are described in the following section.

Step 1. Specify the FDT Names and a Fullword in the CWA

In the CICS CA Compress/2 source library:

- Edit member SHRKWORD to specify the offset from the start of the CWA of an aligned fullword in the CWA to be used exclusively by CA Compress/2 for CICS.
- Edit member FDTNAMES to specify the names of the FDTs for the data sets used with CA Compress.

  If you need to change the offset of the reserved fullword, you must repeat this and reassemble using INSTALL. If converting from the macro level interface, you must change SHRKWORD to the offset from the CWA, not the CSA.

  For seldom used files, you can omit the record in FDTNAMES specifying the corresponding FDT, because the FDT is loaded dynamically as needed, along with about 3K of storage for each transaction. This storage, and the dynamically loaded FDT, is freed when it is no longer needed. Because each FDT may use between 4 and 8K of storage, save storage in this way or, whenever appropriate, use a single FDT for several files.

  CICS CA Compress/2 for CICS can issue the following CICS transaction abends:

  ```
  SHR1 - SHRKWORD NOT INITIALIZED OR DdOES NOT POINT TO SHRKSCBS
  SHR2 - NO RETURN CODE PARAMETER PASSED TO SHRKSTUB
  ```

  The CICS interface requires a return code argument which is optional in batch. This prevents CICS from ABENDing on CA Compress/2 errors. Correct the application program so that it passes a return code parameter.

  The names of FDTs prefixed by SCB in the FDTNAMES member are the FDTs that are loaded into storage at CICS startup. Edit this member as needed.

Step 2. Assemble and Link the Program Modules

Next, modify and run YOUR.CAI.CCVBJCL(INSTALL) to assemble and link the program modules. INSTALL contains two in-stream procedures.

- INIT deletes and reallocates the SHRINK CICS load library.
- CMDASM assembles and links the program modules.

Make the changes shown below to assemble and link the SHRNKMOD, SHRKCICS, and SHRKSCBS modules. If you need to change FDTNAMES, the member that holds the list of FDT names, you must assemble and link the SHRKSCBS module and update the PPT to reflect the current list of FDT names.

- The DSN of the Assembler H library.
- The DSN of the standard system macro library.
- The DSN assigned to YOUR.CAI.CCVBSAMP.
- The DSN of the CICS macro library.
- The DSN of the CICS source library.
- Link edit parameters must default or be specified for *AMODE=24* and *RMODE=24*.
- The DSN assigned to YOUR.CAI.CCVBLOAD.

Step 3. Modify the CICS Tables

After you run INSTALL to assemble and link edit the modules, define SHRNKMOD, SHRKSCBS, and all FDT modules in the PPT and link the application programs as follows:

- In the Processing Program Table (PPT), define SHRNKMOD, SHRKSCBS, and all FDT modules to be used with CA Compress/2 for CICS. Specify all PPT entries as PGMLANG=ASSEMBLER, PGMSTAT=ENABLED, and RELOAD=NO.
- In the Program List Table (PLT) that specifies programs to be executed during CICS initialization, define SHRNKMOD, or you can run it as a startup transaction. Make sure that this PLT is identified by the PLTPI operand of the DFHSIT macro.

Step 4. Modify and Link edit the Application Programs

You must convert your application programs to call the SHRINK and EXPAND subroutines correctly and link edit them with the SHRKCICS module. If you are converting from the Macro Level facility, which is not supported under CICS Version 3, you must not mix this Command Level facility with any modules, including SHRKCICS, from the Macro Level facility. Convert the application modules as follows:

- If you are converting existing application programs from the macro-level product that was supplied with releases of CA Compress before version 4.6.3, you must change the programs as described in the comments in member INSTALL. In particular, the call to SHRKCICS must conform to command level specifications:
  - Register 13 must point to a valid save area.
  - The 5 arguments required by CA Compress must be preceded by the EIB and a dummy DFHCOMMAREA.
- Link edit the application programs with the SHRKCICS module for CICS. Do not use SHRKSTUB, SHRKEXPD or the Macro Level version of SHRKCICS.
- Define the application programs in the PPT as usual.

CA Compress/2 Subroutines Under CICS

CA Compress/2 for CICS provides two subroutines for processing compressed data in CICS application programs. Each application program must be link edited with the module SHRKCICS (rather than with SHRKSTUB or SHRKEXPD, as specified earlier in this chapter for CA Compress/2). The subroutines are as follows:

SHRINK

Converts an uncompressed record image to compressed form.

EXPAND

Converts a record image to its original uncompressed form.

Note: The called subroutines from the application modify storage obtained by the startup module, SHRNKMOD. Unless disabled, Intertest will issue a breakpoint on these instructions. Proceed and disregard the breakpoint. One such instruction that will cause a breakpoint is located at label SE in the CSECT SHRKCICS.

These differ slightly from the subroutines with these names in CA Compress/2:

- The 5 arguments passed to the SHRINK or EXPAND subroutines must be preceded by the EIB and a dummy DFHCOMMAREA, as required by command level CICS.
- The fourth parameter must be an SCB.

Because the CA Compress/2 subroutines modify the SCB, it must be in transaction-related dynamic storage. The SCB should be specified as documented in the section *Accessing the FDT* in this chapter.

The CA Compress/2 Utilities

Unlike the subroutines, the utilities offer no capabilities beyond those supplied through the CA Compress Transparency or SUBSYS implementations. All utility functions are supported more effectively by the subsystem and the Interactive User Interface, and so the utilities are seldom called for. The utilities supply the following functions:

- Prepass performs test compression, like the Interactive User Interface, and creates a sequential FDT.
- The FDTLOADR utility converts a sequential FDT to load module format. The Interactive User Interface executes this utility automatically because the Transparency and SUBSYS implementations do not support sequential FDTs.
- Two compression utilities are supplied to support Hardware Compression and custom compression using FDTs generated either by the Prepass function or by the Interactive User Interface. No utility support is provided for compression using Standard Tables and Super Express.
- Three expansion utilities are provided to support both custom and noncustom methods, including Hardware Compression—EXPAND and EXPANDX, and SHRHCXPD. These enable disaster recovery in the event that no subsystem is available. Minor differences between the utilities and the subsystem require care when expanding data compressed by the subsystem.

The CA Compress/2 utilities are not reentrant and cannot reside in the Link Pack Area. JCL to execute these utilities is provided in the distribution JCL, file number 1 on the release tape.

Prepass

Prepass test compresses a data set and creates a File Descriptor Table (FDT) for CA Compress/2 to use to compress and expand the data. The Interactive User Interface also performs this function. Both methods create FDTs with an Integrity Check Block (ICB) for additional safety.

Note: FDTs created with older releases of the Prepass utility may not have the ICB.

All or a portion of the file can be Prepassed. Prepass collects statistical information relating to compression of the Prepassed file and builds a sequential FDT to contain this information. The FDT must be present (through JCL specification) in every job step that expands or compresses the data set.

Prepass is invoked by executing the program named SHRINK and supplying a PARM value in the form *P=xxx*, where xxx is either the word ALL or a three-digit number. If ALL is coded, the entire data set defined by the INFILE DD statement is Prepassed (read). Coding a three-digit number indicates, in thousands, the number of logical records to Prepass. The first xxx thousand records in the data set are read, the FDT is constructed and written, and the utility goes immediately to end-of-job without reading the remaining records in the data set.

Member SHR2PASS in YOUR.CAI.CCVBJCL executes the Prepass function. The PARM specification *P=015* indicates that the first 15,000 records of the INFILE data set are Prepassed (or the entire data set if it contains fewer than 15,000 records). In most cases, it is unnecessary to Prepass more than five to ten percent of a data set in order to collect accurate compression statistics for the FDT.

The optional RECDEF DD statement defines the data set containing user-specified RDL for the data set. As shown, user-specified record definitions are optional. If omitted, Prepass assumes defaults based on the data set's attributes. The defaults assumed by Prepass Utility produce good compression and often execute faster than more precise user specifications. Seethe chapter *Record Definition* for more information.

COBOL users should specify record definitions in all cases. To direct Prepass to supply default definitions for COBOL users, use the following minimum record definition:

```
//RECDEF DD *
L.
 /*
```

Compressed records are variable-length, and the L. specification prefixes the compressed data with a binary halfword containing its length, including the length field. This gives the COBOL program access to the length of the compressed data.

Prepass Statistics

The printed output from the File Prepass Utility comprises the following:

- The input file's DCB information (RECFM, LRECL, BLKSIZE, and if applicable, RKP and KEYLEN).
- Record definitions (RDL statements) either specified by the user or assumed by default.
- Error messages about incorrect usage of the RDL. Each message is prefixed by an alphanumeric character, which also underscores the error in the RDL statement to which the message applies.
- The number of records Prepassed.
- The shortest, longest, and average record lengths on the Prepassed file. For fixed-length records, all three numbers are identical.

If field types C1, C2 and/or C3 are defined in the RDL for the file Prepassed (either by user specification or by default), a table is printed for each such field type defined showing the compression bit code for each of the 256 possible values containable in one byte, plus the bit code used to represent the repetition indicator. The bit codes are assigned based on the relative frequency of each byte; most frequently encountered values are assigned the shortest bit codes, and least frequently encountered values are assigned the longest bit codes.

FDTLOADR Utility

The FDT can be converted from a sequential data set to a load module by using the FDTLOADR Utility. Because sequential FDTs are not supported by CA Compress Transparency or SUBSYS, the Interactive User Interface performs this function automatically.

Note: FDT member names must be unique within a PDS library containing FDTs and unique across the installation. It is extremely dangerous to have more than one FDT with the same name.

Because FDTs in load module format must be accessed in a slightly different way from sequential FDTs, converting existing FDTs to load module format requires minor modifications to the application programs. These differences are discussed in *Accessing the FDT* in this chapter, and in the descriptions of the FDT parameter in the detailed discussions of the SHRINK and EXPAND subroutine calls.

Conversion to load module format offers the following benefits:

- Compatibility with CA Compress, which does not support sequential FDTs.
- Access to all FDTs through STEPLIB instead of having to code a separate TABLxx DD statement for each FDT.

Member SHR2FLDR in the distribution JCL executes the FDTLOADR utility. The TABL00 DD statement defines the FDT in sequential data set format. The SYSLMOD DD statement defines the PDS library and the member name in which the FDT is to be stored in load module format.

Compression Utilities

After a file is Prepassed, all or part of the file can be compressed by the SHRINK Compression Utility, which also prints compression statistics. The utility supports only custom compression using an FDT, not Standard Tables, Hardware Compression, or Super Express.

The SHRINK Compression Utility is invoked by executing the program named SHRINK and supplying a PARM value in the form C=xxx (xxx is either the word ALL or a three-digit number). ALL causes the entire data set defined by the INFILE DD statement to be compressed. A three-digit number indicates, in thousands, the number of logical records to compress from the data set defined by the INFILE DD statement.

Member SHR2CMP2 uses a load module FDT, which is defined by allocating its load module library in STEPLIB and specifying its name using the FN= keyword in the PARM. In both cases, INFILE is the data set to be compressed and OUTFILE is the compressed data set.

The SHRHCCMP Compression Utility is invoked by executing the program named SHRHCCMP and supplying a PARM value in the form:

```
DICT=dictname,C=xxx,N=nnn
```

where xxx is a three-digit count in thousands of records to compress or ALL and nnn is a three-digit noncompressible length. The defaults are C=ALL and N=000. You can omit either one or both of these defaults. DICT= is always required.

For both Compression Utilities, the OUTFILE LRECL should be at least eight bytes greater than the INFILE LRECL, 12 bytes greater if the uncompressed file is non-VSAM fixed length in order to provide for the RDW in the compressed records. In most cases, DCB parameters in the output data set can be defaulted.

Compression Statistics

The Compression Utilities direct a printed report to the data set defined by the PRINT DD statement. This report contains the following information:

- The DCB information (RECFM, LRECL, and BLKSIZE) of a sequential data set, or the VSAM information (LRECL, cluster type, and, if applicable, the RKP and KEYLEN) for both the INFILE and OUTFILE data sets.
- The data set names of both the INFILE and OUTFILE data sets. These are not printed for dummy data sets.
- The number of records compressed and the number of bytes compressed and produced.
- The shortest, longest and average record lengths, both before and after compression, including the four-byte RDW, if present.
- The average compression percentage (0 = no compression and 100 = total compression), defined as:

  100*X–Y/X

  X = average record length before compression.
  Y = average record length after compression.
- For the SHRINK Utility, the number of type PD, ZL, ZR, S, and X fields which contained information incompatible with the field definition. See the chapter *Record Definition* for more information on these field types.

Expansion Utilities

A compressed file can be returned to its original uncompressed form by one of the Expansion Utilities. The entire data set is expanded. The expanded data set is identical to the one that existed before compression. Data integrity is ensured by comparing the check byte appended to each compressed record to a check byte calculated as each record is expanded.

The EXPAND Utility can only expand data sets compressed using a custom FDT. The EXPANDX Utility can only expand data sets compressed using Standard Tables or the string compression methods, Super Express, or Express. The SHRHCXPD Utility can only expand data sets compressed using a Hardware Compression dictionary.

The OUTFILE DD statement defines the uncompressed data set written by the Expansion Utilities. RECFM and LRECL must be the same as in the original uncompressed data set, but BLKSIZE may differ.

The Expansion Utilities assume default values for certain DCB parameters if they are not specified by the user. RECFM, LRECL, and BLKSIZE may be omitted in most cases. The Expansion Utilities report the same INFILE and OUTFILE information, as does the Compression Utility.

Note on Disaster Recovery: If the CA Compress Subsystem is unavailable the Expansion Utilities can be used to expand compressed data sets, thus making them accessible. Minor differences between subsystem and utility logic require caution. In particular, to expand physical sequential data sets compressed by the subsystem whose uncompressed format is variable-length, you must include an INSUBSYS DD DUMMY statement in the utility expansion step to process them correctly.

JCL Defaults

Defaults are calculated to make sense, based on the attributes of the original uncompressed data set. The FDT contains the original attributes, so they are available to the SHRINK and EXPAND Utilities, but not to the EXPANDX or Hardware Compression Utilities, which do not use custom FDTs.

Extensive error checking is performed to ensure that the combination of user specifications and defaults assumed by the utilities (for both JCL and RDL) are compatible. When a JCL/RDL conflict is not serious, it is permitted, often with a warning message. Only if it makes no sense at all does the utility end with an error message. It is prudent to perform a trial compression and re-expansion of a few records to do the following:

- Check the defaults (JCL and RDL).
- Analyze possible warning messages.
- Check the RDL for *Record Definitions Imply Wrong Length*.

DSORG

DSORG is not required for the OUTFILE data set. If the data set has partitioned organization, a member name must be specified on the DSN parameter.

RECFM

For the Compression Utilities, the default value for the compressed data set is RECFM=VB. For the EXPAND Utility, the default value for the expanded data set is the same as the original uncompressed data set's RECFM. Except for blocking and spanning, the RECFM (F,V,U) cannot be changed.

LRECL

LRECL of the compressed output from the Compression Utilities defaults to LRECL+8 of the original uncompressed data set. For output from the EXPAND Utility, the LRECL value assumed is that of the original uncompressed data set.

BLKSIZE

The Compression Utilities default to the maximum of BLKSIZE of the original uncompressed data set or LRECL+4 of the compressed data set. The EXPAND Utility defaults to the maximum of BLKSIZE or LRECL+4 of a variable-length uncompressed data set. For RECFM=F, FS or U, BLKSIZE defaults to LRECL. For RECFM=FB or FBS, BLKSIZE defaults to the highest multiple of LRECL which does not exceed the original uncompressed BLKSIZE. If the default exceeds the track capacity, you must explicitly code the BLKSIZE.

AMP

The IBM defaults are assumed.

Chapter 5: SUBSYS DD Parameter

CA Compress provides the following two ways to invoke data set compression using the CA Compress subsystem:

- Automatic (transparent) compression of VSAM or physical sequential data sets using the CA Compress Transparency. This is the preferred method.
- Manually invoked compression of non-VSAM data sets by coding the SUBSYS parameter on DD statements in your JCL (or using dynamic allocation). Using the SUBSYS parameter with VSAM data sets causes OPEN to fail with an 0C4 (IBM APAR OW10331) due to a missing DEB extension.

This section contains the following topics:

How it Works

Coding the SUBSYS parameter directs MVS to use the specified subsystem to process the data set. When you specify the CA Compress subsystem, it compresses and expands the data using I/O module ZSURSHRK, just as the Transparency does, so the results are completely compatible.

After the data is compressed, the SUBSYS parameter must be coded in order to access expanded data. If the data need not be expanded, for instance if it is simply being copied or sorted on an uncompressed field, there is no need to code the SUBSYS DD parameter to process the compressed data set.

Here is a simple example of the SUBSYS DD statement parameter:

```
//MYDDNAME DD DSN=MY.DATA.SET,DISP=OLD,
 // SUBSYS=(ZSAM,SHRK,SUPEREXP)
```

The string 'ZSAM' is the subsystem name given to the CA Compress subsystem by the systems programmer at your installation. The string 'SHRK' is a constant which must always be coded. The string 'SUPEREXP' is the name of the CA Compress compression algorithm that you want to use for this data set. When expanding a compressed data set you must specify the compression algorithm name used when the data set was originally compressed.

Coding the SUBSYS JCL Parameter

Be careful when you code the SUBSYS parameter to invoke CA Compress data compression. This section guides you through the process.

SUBSYS Syntax for the CA Compress Subsystem

As documented in the MVS JCL Reference, the general syntax for coding the SUBSYS on a DD statement for any subsystem is:

```
SUBSYS=(subsystemname,parameters)
```

The parameters for the CA Compress subsystem are broken into four pieces: subtype, fdtname, addname, and other-parameters. So, for the CA Compress subsystem, the syntax can be redefined as:

```
SUBSYS=(subsystemname,subtype,fdtname,addname,
        other-parameters)
```

subsystemname

Specifies the name given to your CA Compress subsystem when it was installed on your system. The usual name is 'ZSAM'.

subtype

This parameter must be the constant 'SHRK'.

fdtname

Specifies the CA Compress compression algorithm or File Descriptor Table (FDT) which is to be used for the data set. The following values are valid:

SUPEREXP

Specifies the Super Express compression algorithm.

STDTBLxx

Specifies that CA Compress is to use one of the provided standard File Descriptor Tables (FDT). The 'xx' can be a number from 01 to 06.

fdtname

Specifies the name of the compression algorithm or File Descriptor Table (FDT) created for your data set. The first character of this name must be alphabetic or the job fails with a JCL error.

addname

Specifies the associated ddname which is to be used by CA Compress instead of dynamically allocating the data set. The associated ddname specifies the ddname of another DD statement coded in the step, which must allocate the data set to be processed. The DD statement where SUBSYS= is coded should normally contain only the SUBSYS parameter. All other DD statement parameters for the file should be coded on the associated DD statement. However, DSN and DCB must sometimes be coded on both DD cards because certain products like CICS can validate them during the OPEN process.

Using an associated ddname is the only way to specify certain DD statement parameters which are incompatible on a DD statement where SUBSYS= is specified (that is, DISP=PASS or for SMS-managed data sets). The addname parameter is positional, so a comma must be coded when it is omitted if other parameters follow it.

The DD statement referenced by the associated ddname is looked for when the data set is opened. If it is not found, CA Compress issues message ZSUR002I to inform you of the missing DD. The data set allocated by the associated ddname's DD statement can be of any type that is supported by QSAM.

Note: The associated ddname may also be needed to avoid I/O errors on concatenated SUBSYS data sets after the first. For an example on how to avoid this problem, see the *CA Compress User Guide*.

other parameters

Specifies other parameters which control the operation of CA Compress. If an error is made in coding these parameters, a JCL error results. Each of the following parameters can be coded one time each in any order:

STD or STANDARD—Specifies that CA Compress is to use the standard File Descriptor Table (FDT). This parameter can be omitted if a File Descriptor Table of the name STDTBLxx is specified for the fdtname parameter (described above). See ADD Statement in the chapter Control File Maintenance Utility for more information about the STANDARD parameter.

SDB=YES or SDB=NO – Specifies whether the actual compressed data set should be written with BLKSIZE calculated using System Determined Blocksize (SDB). SDB=NO directs the compress not to optimize but to use the uncompressed BLKSIZE coded in the JCL or set in the Control File entry. If this is a PS data set, SDB= defaults to whatever was specified or defaulted for the Control File entry. If not, the default is YES unless NOSDB was specified on the started task. SDB= does not affect the BLKSIZE parameter of the SUBSYS data set, which must be nonzero to avoid SDB, and the BLKSIZE will continue to look to user programs like the JCL specification. Unless the compressed data is read as compressed with DCB specified, SDB=YES is best.

MVS SUBSYS Restrictions and Special Processing

MVS imposes some restrictions and special processing rules for the use of the SUBSYS DD parameter. These restrictions and special processing rules are documented in the *IBM JCL Reference Manual* and are listed below for your convenience:

- The SUBSYS DD statement parameter cannot be coded for an SMS-managed data set. However, with the CA Compress subsystem's *associated DDNAME* feature, you can circumvent this IBM restriction. The *associated DD statement* referenced using the *addname* subparameter on the SUBSYS parameter can specify an SMS-managed data set.
- The SUBSYS DD statement parameter cannot be coded with SYSOUT=, *, DATA, DDNAME DLM, DYNAM, OUTPUT or QNAME.
- When the SUBSYS DD statement parameter is coded, the parameters COPIES, DEST, FCB and OUTLIM are ignored.
- If DUMMY is specified, the SUBSYS parameters are checked for syntax, and if they are acceptable the data set is treated as a dummy data set.
- If SUBSYS= is specified on an overriding DD statement, a DUMMY parameter on an overridden DD statement is nullified, and a UNIT parameter is ignored.
- SUBSYS does not work correctly with System Determined Blocksize (SDB). SDB is an IBM facility which chooses the optimum BLKSIZE for a data set based on its device type when the user codes BLKSIZE=0 in JCL. Because a SUBSYS data set looks to the system like a SYSIN/SYSOUT data set, SDB computes the BLKSIZE by adding 4 to the LRECL instead of choosing the BLKSIZE appropriate to the actual device. This results in very poor blocking factors and defeats the purpose of compression.

Special Considerations When Using SUBSYS

You must take several factors into account when determining which data sets to place under control of CA Compress using the SUBSYS parameter.

DCB Information When Using SUBSYS

When you code the SUBSYS parameter for the CA Compress subsystem, you can code the other parameters just as you normally would. However, you must always code the DCB parameters for non-VSAM data sets (BLKSIZE, LRECL, and RECFM), even if they already exist (DISP=SHR or DISP=OLD).

There are three restrictions on BLKSIZE and LRECL which you must observe when using CA Compress:

- The BLKSIZE must be at least 12 bytes greater than the LRECL.
- The LRECL cannot be larger than 32,744 bytes.
- Do not code BLKSIZE=0 in order to invoke System Determined Blocksize.

Compression can cause a record to grow by up to 8 bytes, 12 if it was originally fixed length. In order to prevent the logical record length from exceeding the block size, and to prevent the block size from exceeding the maximum value, the BLKSIZE and LRECL restrictions must be adhered to.

Nonlabeled Tapes

When compressing with the SUBSYS parameter on an output file which is a nonlabeled tape, you must use the *addname* subparameter and specify the DCB information on the associated DD statement. Failure to do it this way will result in an S013-34 abend.

Partitioned Data Sets

Invoking CA Compress using the SUBSYS DD statement parameter supports compression of individual members in a PDS, but not of a PDS as a whole. Below are the correct and incorrect ways to compress data in a PDS. The correct way accesses individual members which are compressed and decompressed by CA Compress as if they were sequential data sets.

Correct JCL

```
//GOODPDS  DD DSN=MY.PDS(MYMEMBER),
 //        DISP=SHR,
 //        DCB=(RECFM=FB,BLKSIZE=4080,
 //        LRECL=80),
 //        SUBSYS=(ZSAM,SHRK,SUPEREXP)
```

Incorrect JCL

```
//BADPDS  DD DSN=MY.PDS,
 //       DISP=SHR,
 //       DCB=(RECFM=FB,BLKSIZE=4080,
 //       LRECL=80),
 //       SUBSYS=(ZSAM,SHRK,SUPEREXP)
```

JCL Restrictions

When the *addname* subparameter is not used, CA Compress dynamically allocates the data set, but when *addname* is used, the data set is allocated to the associated ddname before CA Compress begins to process it. Minor differences arise, for instance, in processing the DISP parameter and in when the data set is cataloged. A number of incompatibilities exist between the SUBSYS parameter and other JCL parameters. SMS-controlled data sets must be allocated through an associated ddname. In general, many problems can be solved by using the associated ddname facility. Some specific considerations follow:

- Depending on your IBM maintenance level, if you use the SUBSYS parameter with a VSAM cluster, an 0C4 abend is likely when clearing a field in a missing DEB extension (APAR OW10331). IBM is unwilling to fix this problem, and there is no work around.
- AMP=(...)—This is a VSAM-only parameter. If this parameter is specified on a subsystem DD statement, the subsystem routines do not execute properly and abends result.
- DISP=(...,PASS)—Because CA Compress dynamically allocates the data set, PASS is not allowed on the DD statement where SUBSYS is coded. However, PASS can be coded on an associated DD statement which is specified using the CA Compress subsystem 'addname' parameter.
- LABEL=(...,NL,...)—When compressing with SUBSYS with output to a nonlabeled (NL) tape, an associated DD statement must be supplied in order to specify the DCB attributes of the compressed file. An associated DD statement is specified using the CA Compress subsystem 'addname' parameter. Failure to do this results in a S013-34 abend.
- VOL=REF=*.ddname (where ddname is a subsystem DD statement)—The CA Compress subsystem is not able to update the volume list in the subsystem DD entry, so the volume serial remains the same throughout the life of the job.
- SPACE=(...,...,RLSE)—When an application program closes multiple data sets in a single CLOSE macro call where one or more of the data sets is compressed by CA Compress RLSE can cause an S50D abend. CA Compress issues messages ZSUR007I and ZSUR050I.

  BLKSIZE=0—System Determined Blocksize does not work correctly with SUBSYS, because the proper device type is not recognized for SUBSYS data sets. If you use the associated ddname and code BLKSIZE=0 on both ddnames, the BLKSIZE is computed differently in each case, and an abend is likely.

Chapter 6: Test Compression Facility

The Test Compression Facility (TCF) is a batch program that allows you to determine the best data compression algorithm to use for each file you want compress. Under the control of SYSIN control statements, TCF analyzes data sets and provides statistical projections of the savings (in kilobytes and DASD tracks) to be realized through Huffman and Super Express compression. TCF enables users of CA Compress to determine which individual data sets are the best candidates for processing, and to project the storage savings.

The TCF is an alternative to the Interactive User Interface (IUI), but the IUI is usually more convenient.

This section contains the following topics:

How The Program Works

The TCF reads SYSIN control statements that control the operation of TCF and specify the data sets and/or data set name patterns you want to analyze.

The sample JCL below runs the TCF. This JCL can be found in member TCFRPT in YOUR.CAI.CCVBJCL.

```
//*TCFRPT JOB
//*
//TCF      EXEC PGM=GDAXP001,REGION=1000K
//STEPLIB   DD  DSN=YOUR.CAI.CCVBLOAD,DISP=SHR
//CMDPRINT  DD  SYSOUT=*
//MSGPRINT  DD  SYSOUT=*
//SYSPRINT  DD  SYSOUT=*
//SYSUDUMP  DD  SYSOUT=*
//SYSIN     DD  *
 --- ENTER ANALYZER COMMANDS HERE ---
```

Notes on Using the Program

TCF produces a report showing the data sets included in the scan and the compression percentages achieved for both the Huffman and Super Express (run length code) compression techniques.

Note: Because it can take a long time to scan every catalog and analyze every VSAM and physical sequential data set in your installation, limit the scope of your TCF job before submitting it for processing. See the SCAN, SELECT and EXCLUDE statement sections later in this chapter for more information.

You can instruct TCF to pass control to a user-coded exit program, which receives the statistical results from the test compression for each data set. The exit program can take customized actions using this information.

TCF can be run in simulate mode that shows you the data sets that are selected by TCF without actually reading and test-compressing the data. This allows you to see how many data sets are being selected before spending the system resources required to test compress them all.

TCF Command Language

The TCF command language operates on both sequential and VSAM data sets. TCF processing consists of a catalog scan, data set selection, and compression analysis. Within a single execution of the program, you can run multiple sets of TCF processing. In order to remove redundant data set selections, TCF preprocesses the scans and combines them so that no data set is selected twice for a single analysis operation (EXAMINE statement—see in the following table).

There are three types of operations done in the command language:

TCF Statements	Operation
SCAN	Search catalog(s) and gather a list of data sets using criteria you specify
SELECT and EXCLUDE	Narrow the list of data set names using criteria you specify.
EXAMINE	Test compress the data sets in the final list and produce a report.

TCF requires at least one SCAN statement and one EXAMINE statement.

Command Language Syntax Rules

The following section describes the command language syntax rules.

- Any line which starts with an asterisk ("*") in column 1 is a comment line and is ignored.
- A statement consists of the statement name followed by one or more parameters. For example:

  ```
  SCAN CATALOG=MY.CATALOG,PREEXIT=MYPREXIT
  ```

- Parameter names can be abbreviated by truncating them to the first three characters, if that makes the parameter unambiguous. If three characters are not enough, then additional characters are required until the abbreviation is unambiguous.
- Parameters can be specified in any order. The first parameter must appear on the same line as the statement name.
- A statement can span more than one line.
- A parameter cannot span more than one line unless the statement is continued using Statement Continuation Method B below.
- A blank following a parameter terminates a statement. Characters following the blank are considered comments.
- Statement Continuation Method A: If a comma and a space follow a parameter immediately, then the statement is continued onto the next line. Characters following the space are considered comments. The parameters appearing on the next line can start in any column. With this continuation method, each parameter must be completed on a single line.
- Statement Continuation Method B: If the statement is coded through column 71, and an 'X' is coded in column 72, then the statement is continued onto the next line. The first nonblank in the next line is appended to the character in column 71 in order to form the continued statement. With this continuation method, the line can be split anywhere. In the example below, the PREEXIT=MYPREXIT parameter is continued onto the next line:

  ```
  ----+----1----+----2----+ ..//.. +----7----+----8
      SCAN CATALOG=MY.CATAL       DE,PREEX
         XIT=MYPREXIT
  ```

- The following special characters can be used in data set names, volume names, and catalog names:
  - An asterisk ("*") means *any characters in a single node*.
  - A question mark ("?") means *any one character*.
  - A slash ("/") means *any suffix of characters*.
  - An exclamation point ("!") means *any characters*.

Examples:

The command language syntaxes are described in the following section.

DSN=*

Selects all single-level data set names.

DSN=*.*

Selects all two-level data set names.

DSN=A.*.PROD

Selects all three-level data set names which have an A as the first node, any character or characters as the second node, and PROD as the third node.

DSN=A*.PROD

Selects all two-level data set names which have an A followed by zero to seven other characters as the first node, and PROD as the second node.

DSN=?

Selects all single-character data set names.

DSN=A.TEST??

Selects all two-level data set names that have an A as the first node, and TEST plus two other characters as the second node.

DSN=A/

Selects all data sets that begin with the character A. The data set names can have any number of nodes. The first node can be the letter A, or be a string that starts with A.

DSN=A.TEST/

Selects all data sets that begin with the string A.TEST. Examples:

"A.TEST"

"A.TEST1"

"A.TEST1.TEST2"

"A.TEST.PROD"

DSN=A.*.C?./

Selects all data sets which have a first node of A, any second node, a third node which is exactly two characters in length and the first character is the letter C, and any node or nodes which follow.

DSN=!TEST

Selects all data sets that end with the string TEST

DSN=!TEST!

Selects all data sets that have the string TEST somewhere in it. The string TEST can be at the beginning or the end of the data set name.

DSN=

!TEST!VSAM

Selects all data sets that have the string TEST somewhere in the name and VSAM at the end. Examples:

"A.TEST.VSAM"

"A.TESTVSAM"

"A.TEST1.VSAM"

"A.TEST1.KSDSVSAM"

Command Structures in the Command Language

The command language processor groups the TCF statements into logical blocks called command structures. There can be many command structures in a single TCF run. A command structure consists of the following sequence of statements:

1. One or more SCAN statements. One or more occurrences of the following sequence:
   - Optional SELECT and/or EXCLUDE statements.
   - One or more EXAMINE statements.

   At least one EXAMINE statement must appear in each command structure.

The SCAN statements apply to all SELECT, EXCLUDE, and EXAMINE statements in the command structure.

Unlike the SCAN statements, which are global in scope, the optional SELECT and/or EXCLUDE statements apply only to the EXAMINE statement(s) which immediately follow them.

The following examples show how the command structures work:

Example 1

The command structure below causes TCF to scan catalog ICF.TESTCAT, extract all VSAM ESDS data sets, all VSAM KSDS clusters, and all physical sequential data sets, and test compress them based on a sampling of 30 percent of the logical records found in each data set.

```
SCAN CATALOG=ICF.TESTCAT
```

EXAMINE PERCENT=30

Example 2

The command structure below causes TCF to select all data sets in all catalogs that begin with the prefix 'LABS.TJP' and run complete test compresses on them.

```
SCAN DSN=LABS.TJP./
```

EXAMINE

Example 3

The command structure below extracts all data set names from three catalogs and then analyzes them:

```
SCAN CATALOG=VOL050.USERCAT
SCAN CATALOG=VOL001.USERCAT
SCAN CATALOG=VOL050.TESTCAT
```

EXAMINE

Example 4

The command structure below extracts data set names matching two different data set name criteria from all catalogs and then analyzes them:

```
SCAN DSN=LABS.TJP.VSAMFIL
SCAN DSN=LABS.*.TESTVSAM/
```

EXAMINE SKIP=20,BYPASS=100,EXTRACT=500

Example 5

The command structure below extracts data set names from two catalogs. Then the list is narrowed by selecting data sets which match either of the two SELECT statements and the selected data sets are test compressed using the first EXAMINE statement. Finally, the entire list of data sets is test compressed using the second EXAMINE statement.

```
SCAN CATALOG=VOL050.USERCAT
SCAN CATALOG=VOL001.USERCAT
```

SELECT DSNAMES=(!TJP,!MJA)
 SELECT DSORG=PS,MBYTESRANGE=(300,2000)

EXAMINE SKIP=5,EXTRACT=10000

EXAMINE PERCENT=40

Example 6

The command structure below extracts data set names from two catalogs. This list of data sets is used twice. In the first use, the list is narrowed by selecting data sets that are between 50 and 200 megabytes in size (first SELECT statement). Then, a test compression run is done using 50% of the records in each data set (first EXAMINE statement). In the second use, the original list is narrowed in a different way. Data sets are selected only if they are 201 megabytes or larger (second SELECT statement). Then a test compress is done using 20% of the records in each data set (second EXAMINE statement).

```
SCAN CATALOG=VOL050.USERCAT
SCAN CATALOG=VOL001.USERCAT
```

SELECT MBYTES=(50,200)

EXAMINE PERCENT=50

```
SELECT MBYTES=201

EXAMINE PERCENT=20
```

Example 7

The command structure below extracts data set names matching two different data set name criteria from all catalogs. Then the list is narrowed by selecting VSAM data sets up to 300 MB in size (SELECT statement) and which do not end in *DB2* or *TEST* (EXCLUDE statement). The resulting list of data sets is test compressed using 20% of the records in each data set (EXAMINE statement).

```
SCAN DSN=LABS.TJP.VSAMFIL
SCAN DSN=LABS.TCF.VSAM/

SELECT DSORG=VSAM,MBYTESRANGE=(0,300)
 EXCLUDE DSN=(!DB2,!TEST)

EXAMINE PERCENT=20
```

SET Statement

The following section describes the SET statement.

```
SET  [MODE=LIVE|SIMULATE]
    [,PREEXIT=exitname]
     [,POSTEXIT=exitname]
     [,PERCENT=100|n]
     [,BYPASS=0|n]
     [,SKIP=0|n]R       [,EXTRACT=0|n]
     [,DSNFILL=.|x]
     [,SECURITY=NONE|RACF|TOPSEC|ACF2]
```

The SET statement specifies options for the execution of the command structures that follow it. However, parameters on an EXAMINE statement will override those specified on a SET statement. Any SET statement which specifies MODE=SIMULATE causes the entire run to be in simulate mode even if MODE=LIVE is specified on another SET statement.

You can use the SET statement to establish default values for the compression run which are different from normal TCF default values. The following example shows how the SET statement works with EXAMINE statements:

```
1:      SCAN CAT=ICF.PAYROLL    All of the records

2:              EXAMINE
3:      SET PERCENT=30          Estab new default
4:      SCAN CAT=!IMS
5:              EXAMINE         30% of the records
6:      SCAN CAT=!CICS7:
                EXAMINE PERCENT=20      20% of the records
8:      SCAN CAT=ICF.PROD019:
                EXAMINE         30% of the records
```

The EXAMINE statement in line 2 runs a test compression on all of the records in all of the data sets cataloged in ICF.PAYROLL because TCF processes all records by default. The SET statement in line 3 establishes a new default of running test compression on 30% of the records. The EXAMINE statement in line 5 runs a test compression on a 30% sample of the records in all of the data sets cataloged in catalogs which have a name ending in 'IMS'. The EXAMINE statement in line 7 runs a test compression on a 20% sample of the records in all of the data sets cataloged in catalogs which have a name ending in 'CICS' because PERCENT=20 is specified on it. The EXAMINE statement in line 9 runs a test compression on a sample of 30% of the records in all of the data sets cataloged in ICF.PROD01.

The SET statement is required if you want to use TCF's security interfaces. When used in this way, the SET statement should precede the first EXAMINE statement in the input stream, as in the following example:

```
SET SECURITY=ACF2, ...
```

```
SCAN CAT=ICF.TESTCAT
   EXAMINE PERCENT=20
```

MODE=LIVE|SIMULATE

Specifies whether the test compression is a simulated execution. When MODE=SIMULATE (or MODE=SIM) is specified, a list of data sets is developed by executing the SCAN, SELECT, and EXCLUDE statements, but the test compression is not executed. Messages and reports are produced as if normal processing were occurring, without the compression percentages and resulting data savings being printed. This is useful in determining how big a TCF run you have without going through the work of doing the test compressions.

PREEXIT=modname

Specifies the name of your user-coded preprocessing exit. The purpose of this exit is to allow you to write custom code to select data sets for compression. This is only necessary when TCF's selection process (SCAN, SELECT, and EXCLUDE statements) does not give you enough power to select just the data sets you want to select. Your exit receives control before catalog and format 1 DSCB information is obtained for the data set. You can set a return code to force TCF to bypass the data set. See PREEXIT Pre-Processing Exit in the chapter User Exits for more information.

POSTEXIT=modname

Specifies the name of your user-coded postprocessing exit. The purpose of this exit is to allow you to perform customized processing on the compression statistical data that is available after the test compression is complete. Your exit receives control after each data set is successfully test compressed. If you run TCF in simulate mode, your exit will also receive control, but no compression statistics will be available. See PREEXIT Pre-Processing Exit in the chapter User Exits for more information.

PERCENT=100|n

Specifies the percent of records to be sampled during the test compression. Use this parameter to reduce processing resource requirements for the TCF run. The sampling occurs uniformly throughout the data set. For example, PERCENT=20 samples one in every five records.

Note: Whenever a value other than 100 is coded, the BYPASS, SKIP and EXTRACT parameters are ignored.

BYPASS=0|n

Specifies the number of records to bypass at the start of a data set before test compression is to begin.

Note: It is possible to prevent any records from being selected for test compression through values you specify on BYPASS and SKIP. If no records are selected for test compression, message DCA0074 is issued and the data set is flagged on the report with an error indication.

SKIP=0|n

Specifies that TCF is to process every nth record when selecting records for test compression. For example, if SKIP=4 is specified, records 1, 2 and 3 are skipped and the 4th record is test compressed. Then records 5, 6 and 7 are skipped and the 8th record is test compressed.

This parameter works in concert with the BYPASS parameter. When BYPASS is specified, the number of records specified on BYPASS= is bypassed before the SKIP processing begins.

Note: It is possible to prevent any records from being selected for test compression through values you specify on BYPASS and SKIP. If no records are selected for test compression, message DCA0074 is issued and the data set is flagged on the report with an error indication.

EXTRACT=0|n

Specifies the maximum number of records that are to be test compressed for each data set. Once this number of records has been test compressed, TCF stops processing records for that data set. EXTRACT=0 specifies that there is no limit.

The EXTRACT parameter only counts the records selected for test compression. Records which are bypassed due to BYPASS= and skipped due to SKIP= are not counted toward this total.

DSNFILL=.|x

Specifies the *fill character* to use on the TCF Report which helps you to visually align the data set name with its compression results. The default value is a period. See the TCF Report in this chapter for more information about this report.

SECURITY=NONE|RACF|TOPSEC|ACF2

Specifies the security system that you want TCF to receive authorization from in order to obtain access to each data set.

SCAN Statement

The following section describes the SCAN statement.

```
SCAN   {CATALOG=catalogname|DSNAME={dsname|pattern}}
[,EXCCATS=(catnamelist)]
 [,BEGINDSN=dsname]
 [,PREEXIT=exitname]
 [,POSTEXIT=exitname]
```

The SCAN statement is used to indicate the VSAM/ICF catalog or data set name pattern to be scanned. One or more SET statements can be coded at the top of a command structure. See the Command Structures in the Command Language in this chapter for more information.

Either CATALOG or DSNAME must be specified for the SCAN statement to be valid. It is not valid to specify both on the same SCAN statement.

Narrowing the list of data sets to be processed using the DSNAME parameter on the SCAN statement usually requires less CPU time and less elapsed time than using the CATALOG parameter on the SCAN statement in conjunction with a SELECT statement. This is especially true if your VSAM or ICF catalogs are large. For example, the following two command structures produce the same result, but the second one is generally more efficient.

```
SCAN CATALOG=ICF.TESTCAT
SELECT DSNNAMES=(LABS.TJP./)
    EXAMINE
```

produces the same result as:

```
SCAN DSNAME=LABS.TJP./
    EXAMINE
```

Note: The assumption here is that all data sets named LABS.TJP./ are cataloged to ICF.TESTCAT. For the second command structure to work properly, an alias must exist in the master catalog for the high-level qualifier of the data set name.

CATALOG=catalogname

Specifies the catalog(s) to be searched. If pattern characters are specified, then more than one catalog can be selected on a single SCAN statement. See Command Language Syntax Rules in this chapter for a complete description of the use of the special characters. The CATALOG parameter and the DSNAME parameter cannot be used together.

Note: If the value you specify for CATALOG= does not match any of the catalog names in your installation, TCF will issue an error message and terminates all commands in the command stream.

Note: AB/ is not the same as AB./ even if AB is a complete node. The former requires much more memory and processing time than the latter because the end of the node is not indicated, so that possibilities like ABX or ABY have to be checked for, even if they turn out not to exist. AB./ permits TCF to limit the catalog locate function to just that node.

DSNAME=dsname|pattern

Specifies the data set(s) to be processed. If pattern characters are specified, then more than one data set can be selected on a single SCAN statement. For a complete description of the use of the special characters, see the CA Compress *User Guide*. The first qualifier of the name must not contain any special characters.

The DSNAME parameter and the CATALOG parameter cannot be used together.

Generally, using DSNAME is more efficient than using CATALOG. The CATALOG parameter searches an entire catalog. The DSNAME parameter causes TCF to issue a superlocate to find just those data sets that match the DSNAME parameter.

EXCCATS=catalogname

Specifies up to ten catalog names and/or catalog name patterns that are to be excluded from processing. This parameter is only valid when CATALOG= is specified. For example, the following SCAN statement scans all catalogs which have IMS at the end of their name, but excludes those with TEST or TEMP anywhere in their name.

```
SCAN CATALOG=!IMS,EXCCATS=(!TEST,!TEMP)
```

BEGINDSN=dsname

Specifies the starting point in the collating (sorting) sequence of data set names where you want TCF to begin processing data sets. This is useful when you want to resume processing at the place where the TCF job was prematurely terminated.

PREEXIT=modname

Specifies the name of your user-coded preprocessing exit. The purpose of this exit is to allow you to write custom code to select data sets for compression. This is only necessary when TCF's selection process (SCAN, SELECT, and EXCLUDE statements) does not give you enough power to select just the data sets you want to select. Your exit receives control before catalog and format 1 DSCB information is obtained for the data set. You can set a return code to force TCF to bypass the data set. See PREEXIT Pre-Processing Exit in the chapter User Exits for more information.

The PREEXIT parameter coded on the SCAN statement overrides any PREEXIT parameter coded on a previous SET statement.

POSTEXIT=modname

Specifies the name of your user-coded postprocessing exit. The purpose of this exit is to allow you to perform customized processing on the compression statistical data, which is available after the test compression, is complete. Your exit receives control after each data set is successfully test compressed. If you run TCF in simulate mode, your exit also receives control, but no compression statistics are available. See PREEXIT Pre-Processing Exit in the chapter User Exits for more information.

The POSTEXIT parameter coded on the SCAN statement overrides any POSTEXIT parameter coded on a previous SET statement.

SELECT Statement

The following section describes the SELECT statement.

```
SELECT [DSNAMES=(dsnamelist)]
    [,VOLUMES=(volumelist)]
     [,MBYTESRANGE=(0|n[,n])]
     [,DSORG=ALL|VSAM|PS]
```

The SELECT statement is used to narrow the list of data sets to be processed by a subsequent EXAMINE statement. For a data set to be selected for processing, all tests specified on the SELECT statement must be met. If more than one SELECT statement is coded before an EXAMINE statement, a data set can be selected by any one of the SELECT statements to be processed by the EXAMINE statement. Multiple SELECT statements are ORed together.

Note: If an EXCLUDE statement immediately precedes or follows any SELECT statement, special rules apply. See the SELECT and EXCLUDE Processing Rules in this chapter for a more information.

DSNAMES=dsname

Specifies the data set name(s) to be processed. Up to ten names or name patterns can be specified on a single SELECT statement.

VOLUMES=volume

Specifies the volumes on which the data sets must reside in order to be processed. Up to thirty (30) volumes or volume name patterns can be specified on a single SELECT statement. For multivolume data sets, the first (primary) volume is used for this comparison.

MBYTESRANGE=(0|n[,n])

Specifies the size which the data sets must be in order to be processed. The data set's size is the number of allocated tracks and cylinders converted into megabytes, not the actual space usage. The first number of MBYTESRANGE specifies the minimum size, which the data set must be in order to be included in TCF processing. The second number specifies the maximum size.

```
MBYTESRANGE=(0,600)
 MBYTESRANGE=(300,1200)
 MBYTESRANGE=(,500) <--- Invalid
```

Note: If you code a maximum size, you must also code a minimum size, even if the minimum is zero. Examples:

DSORG=ALL|VSAM|PS

Specifies the data set organization which the data sets must have to be processed.

EXCLUDE Statement

The following section describes the EXCLUDE statement.

```
EXCLUDE [DSNAMES=(dsname list)]
     [,VOLUMES=(volume list)]
     [,MBYTESRANGE=(0|n[,n])]
     [,DSORG=ALL|VSAM|PS]
```

The EXCLUDE statement is used to narrow the list of data sets, which are to be processed by a subsequent, EXAMINE statement. In order for a data set to be excluded from processing, all tests specified on the EXCLUDE statement must be met. If more than one EXCLUDE statement is coded prior to an EXAMINE statement, then a data set can be excluded from processing by any one of the EXCLUDE statements. That is, multiple EXCLUDE statements are ORed together.

Note: If an EXCLUDE statement immediately precedes or follows any SELECT statement, special rules apply. See the SELECT and EXCLUDE Processing Rules in this chapter for more information about these rules.

DSNAMES=dsname

Specifies the data set names to be excluded from processing. Up to ten names or name patterns can be specified on a single EXCLUDE statement.

VOLUMES=volume

Specifies the volumes on which the data sets must reside in order to be excluded from processing. Up to thirty (30) volumes or volume name patterns can be specified on a single EXCLUDE statement. For multi-volume data sets, the first (primary) volume is used for this comparison.

MBYTESRANGE=(0ln[,n])

Specifies the size which the data sets must be in order to be excluded from processing. The data set's size is the number of allocated tracks and cylinders converted into megabytes, not the actual space usage. The first number of MBYTESRANGE specifies the minimum size which the data set must be in order to be excluded from TCF processing. The second number specifies the maximum size.

```
MBYTESRANGE=(0,600)
MBYTESRANGE=(300,1200)
MBYTESRANGE=(,500) <--- Invalid
```

Note: If you code a maximum size, you must also code a minimum size, even if the minimum is zero. Examples:

DSORG=ALL|VSAM|PS

Specifies the data set organization which the data sets must have to be excluded from processing.

SELECT and EXCLUDE Processing Rules

The SELECT and EXCLUDE statements can be coded any number of times for a single EXAMINE statement. The following rules govern TCF's processing of multiple SELECT and/or EXCLUDE statements for a single EXAMINE statement:

- For each SELECT or EXCLUDE statement, all criteria must be met for the statement to select or exclude the data set.
- If only SELECT statements are coded for a given EXAMINE statement, at least one SELECT statement must be true for the data set to be selected for TCF processing.
- If only EXCLUDE statements are coded for a given EXAMINE statement, at least one EXCLUDE statement must be true for the data set to be excluded from TCF processing.
- If both SELECT and EXCLUDE statements are coded for a given EXAMINE statement, the following rules apply:
  - If an EXCLUDE statement tests true, the data set is excluded whether a SELECT statement tests true or not.
  - If no EXCLUDE statements test true, at least one SELECT statement must still test true for the data set to be processed.

EXAMINE Statement

The following section describes the EXAMINE Statement.

```
EXAMINE [PERCENT=100|n]
     [,BYPASS=0|n]
     [,SKIP=0|n]
     [,EXTRACT=0|n]
```

The EXAMINE statement initiates test compressions for the list of data sets selected by the SCAN, SELECT, and EXCLUDE statements which precede it. All records in all data sets are test compressed unless record sampling is specified by the BYPASS, SKIP, and/or EXTRACT parameters on the EXAMINE statement itself, or on any SET statement coded in the command stream.

PERCENT=100|n

Specifies the percent of records to be sampled during the test compression. The reason you would want to use this parameter is to reduce processing resource requirements for the TCF run. The sampling occurs uniformly throughout the data set. For example, PERCENT=20 samples one in every five records.

Note: Whenever a value other than 100 is coded, the BYPASS, SKIP and EXTRACT parameters are ignored.

BYPASS=0|n

Specifies the number of records to bypass at the start of a data set before test compression is to begin.

Note: It is possible to prevent any records from being selected for test compression through values you specify on BYPASS and SKIP. If no records are selected for test compression, message DCA0074 is issued and the data set is flagged on the report with an error indication.

SKIP=0|n

Specifies that TCF is to process every nth record when selecting records for test compression. For example, if SKIP=4 is specified, records 1, 2 and 3 are skipped and the 4th record is test compressed. Then records 5, 6 and 7 are skipped and the 8th record is test compressed.

This parameter works in concert with the BYPASS parameter. When BYPASS is specified, the number of records specified on BYPASS= is bypassed before the SKIP processing begins.

Note: It is possible to prevent any records from being selected for test compression through values you specify on BYPASS and SKIP. If no records are selected for test compression, message DCA0074 is issued and the data set is flagged on the report with an error indication.

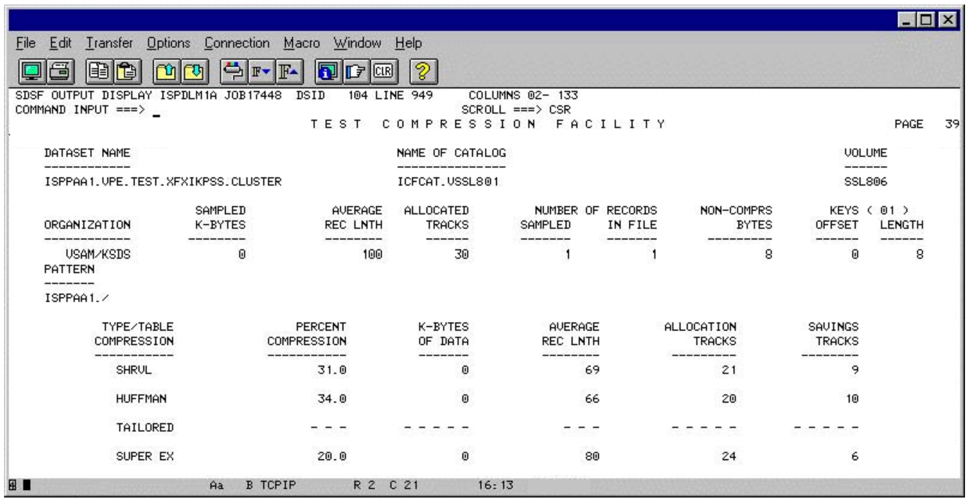
EXTRACT=0|n

Specifies the maximum number of records, which are to be test compressed for each data set. After this number of records has been test compressed, TCF stops processing records for that data set. EXTRACT=0 specifies that there is no limit.

The EXTRACT parameter only counts the records selected for test compression. Records which are bypassed due to BYPASS= and skipped due to SKIP= are not counted toward this total.

TCF Report

The following figure shows the format of the TCF report.



```
SDSF OUTPUT DISPLAY ISPDLM1A JOB17448  DSID    104 LINE 949      COLUMNS 02- 133
COMMAND INPUT ===> _                                            SCROLL ===> CSR
                                         T E S T   C O M P R E S S I O N   F A C I L I T Y                                  PAGE   39

    DATASET NAME                                     NAME OF CATALOG                                                DATASET VOLUME
    ------------                                     ---------------                                                ------
    ISPPAA1.VPE.TEST.XFXIKPSS.CLUSTER                ICFCAT.VSSL801                                                 SSL806

                        SAMPLED             AVERAGE  ALLOCATED      NUMBER OF RECORDS     NON-COMPRS       KEYS ( 01 )
    ORGANIZATION        K-BYTES            REC LNTH     TRACKS    SAMPLED      IN FILE         BYTES    OFFSET   LENGTH
    ------------       --------            --------     ------    -------      -------     ---------    ------   ------
       VSAM/KSDS              0                 100         30          1            1             8         0        8
    PATTERN
    -------
    ISPPAA1./

            TYPE/TABLE                 PERCENT          K-BYTES            AVERAGE          ALLOCATION          SAVINGS
           COMPRESSION             COMPRESSION          OF DATA           REC LNTH              TRACKS           TRACKS
           -----------             -----------          -------           --------           ---------         --------
              SHRVL                       31.0                0                 69                  21                9

              HUFFMAN                     34.0                0                 66                  20               10

              TAILORED                   - - -          - - - - -            - - -           - - - - -        - - - - -

              SUPER EX                    20.0                0                 80                  24                6
```

Field Description and Contents

The following definitions apply to the fields found on the Test Compression Facility Report:

DATA SET NAME

The name of the data set being analyzed.

NAME OF CATALOG

The name of the catalog for the data set.

VOLUME

The name of the volume for the data set.

ORGANIZATION

The Data Set Organization: KSDS, ESDS, or Physical Sequential.

SAMPLED K-BYTES

The estimated number of kilobytes of data sampled.

AVERAGE REC LNTH

The estimated average record length of the records sampled.

ALLOCATED TRACKS

The total number of tracks allocated to the data set (not the number of tracks which contain data).

NUMBER OF RECORDS SAMPLED

The total number of records sampled.

NUMBER OF RECORDS IN FILE

The estimated number of records in the file sampled for VSAM. This field is not applicable for Physical Sequential data sets.

NON-COMPRS BYTES

The total number of noncompressible bytes.

KEYS

The number keys in the record.

KEYS OFFSET

The offset of the keys in the record.

KEYS LENGTH

The length of the keys in the record.

PATTERN

The SCAN or SELECT criteria used to find the record.

TYPE/TABLE COMPRESSION

The name of the compression technique that was used.

PERCENT COMPRESSION

The estimated percentage of the file to be compressed.

KBYTES OF DATA

The estimated number of kilobytes of data after compression.

AVERAGE REC LNTH

The estimated average record length after compression.

ALLOCATION TRACKS

The estimated number of allocated tracks after compression.

SAVINGS TRACKS

The estimated number of tracks saved after compression.

CURRENT RDL STATEMENT

The current record description language.

Chapter 7: VSAM Performance Enhancement

The VSAM Performance Enhancement (VPE) is a general performance enhancer for VSAM, which operates independently from and compatibly with the other components of CA Compress. In most situations, implementing VPE dramatically reduces the wall clock run time of both batch jobs and online transactions, which use VSAM data sets.

This section contains the following topics:

VPE

VPE does the following:

- Eliminates redundant I/Os through buffering.
- Uses memory above the 16 Mb line for any program. It is possible to have files completely loaded into ESA's large virtual storage areas.
- Allows multiple files to share a single buffer pool (Local Shared Resources (LSR)).
- Optimizes sequential buffers using Read Look-Aside and Deferred Write (DFW) techniques for batch jobs.
- Uses Sequential Insert Strategy (SIS) and Deferred Write (DFW) techniques for VSAM files used by CICS.
- Gives batch programs the advantage of VSAM Local Shared Resources (LSR) using a proprietary technique.

It is not necessary to modify your application programs to use VPE. VPE invocation is controlled in 2 ways, both external to the application:

- Control statements in the VPE Rules Table, and/or
- Special DD statements added to the application JCL.

VPE transparently gains control each time a VSAM data set is opened through an interface program, which is installed and removed by a batch job. An IPL is not required.

VPE provides reporting to give visibility to its benefits and to aid in tuning.

How VPE Enhances VSAM Performance

The VPE installation program installs intercepts at OPEN and CLOSE. During OPEN, VPE modifies the VSAM ACB in order to invoke the following techniques to improve VSAM performance:

- Optimized I/O Buffering
- Deferred Write (DFW)
- Sequential Insert Strategy (SIS)
- Read Look-Aside

During CLOSE, VPE performs statistics acquisition and reporting.

Optimized I/O Buffering

VPE utilizes a proprietary technique for optimizing VSAM I/O buffers. VPE creates large numbers of VSAM I/O buffers in virtual storage above the 16 Mb line (when available) in order to dramatically reduce I/O operations.

Deferred Write (DFW)

The performance of many applications suffers from the result of accessing VSAM work files, which cannot be tuned to avoid CI and CA splits. Deferred Write (DFW) allows splits to occur in virtual storage resulting in decreased CPU and DASD utilization when splits occur.

Sequential Insert Strategy (SIS)

VPE invokes VSAM's Sequential Insert Strategy (SIS) automatically and when appropriate when records are added to a VSAM data set in order to improve performance. SIS reduces the number of CA and CI splits when adding multiple records at one point because it splits the CI or CA at the insertion point rather than in the middle of the CI or CA. Because half empty CIs and CAs no longer proliferate, FREESPACE can be reduced without increasing splits.

Read Look-Aside

VPE invokes VSAM's Read Look-Aside feature automatically and when applicable. VSAM's Read Look-Aside feature reduces I/Os by looking first in the existing I/O buffers for the CI being read before reading from disk.

VPE's Use of VSAM's Local Shared Resources (LSR)

VSAM Local Shared Resources (LSR) is inappropriate for some types of VSAM file processing. VPE analyzes the files and processing modes specified by you and modifies the environment accordingly. In certain cases, sequential optimization is done. Algorithms determine the optimum number of buffers to assign to gain the most read-ahead without wasting storage.

Reports Allow VPE Tuning

Statistics and performance reports are produced showing how files were optimized and the percent of I/Os eliminated. These reports allow you to customize the buffer specifications for a particular job to meet schedules and balance resource consumption with run time improvements.

Installing VPE

When VPE is installed using the configuration, which comes on the install tape, by default it does not operate on any of your VSAM files. You can safely install VPE and have confidence that it does not operate on any files until you tell it to do so.

All of the components of VPE are copied from the distribution tape and installed on your machine when you perform the installation procedure described in the *CA Compress Installation Guide*. There are no other installation steps necessary for VPE.

Note: The installation procedure installs the VPE load modules in one of the libraries in your LINKLIST. If you repeat the installation procedure (for example. upgrade, reinstall, and so on), be sure that you do not end up with more than one copy of VPE in your LINKLIST. If different releases are mixed, or if the VPE modules are not all properly installed in a LINKLIST library, abends and other problems can occur.

There are several steps required to implement VPE:

1. Copy the VPE data sets from the distribution tape to the appropriate places on disk. This is the Installation step previously discussed.
2. Modify the parameters, which control the operation of VPE. These are the VPE Rules Table and the VPE Control DD Statements (discussed below).
3. Activate the VPE OPEN interface program by running program VPEHINST.
4. Repeat steps 2 and 3 as required to refine your VPE implementation.

Activating and Deactivating VPE on Your System

The VPE Operating System interfaces are installed and removed by running the VPEHINST program. The interface programs can be installed in an activated or deactivated state. When deactivated, the programs are still installed in the operating system but are nonfunctional.

The VPEHINST program typically runs as part of your IPL sequence to install the VPE Operating System interfaces. The JCL required to run this program is as follows:

```
//VPEJOB JOB ...
 //STEP1  EXEC PGM=VPEHINST,
 //       PARM='ACTION=actionname[,RESTRICT=jobname']
 //
```

The valid values for actionname are:

ACTION=STATUS

Directs VPEHINST to display the status of the VPE Operating System interfaces on the system console.

ACTION=INSTALL

Directs VPEHINST to install the VPE Operating System interfaces.

ACTION=INSTALLD

Directs VPEHINST to install the VPE Operating System interfaces in deactivated state.

ACTION=ACTIVATE

Directs VPEHINST to activate the VPE Operating System interfaces which are currently installed and deactivated.

ACTION=DEACTV

Directs VPEHINST to deactivate the VPE Operating System interfaces which are currently installed and activated.

ACTION=DELETE

Directs VPEHINST to remove the VPE OPEN interface, but to leave the VPE CLOSE interface installed and active. This is desirable when there are jobs currently running with open VSAM data sets and you want the VPE CLOSE interface to gather statistics on them when the CLOSE occurs.

ACTION=FORCDEL

Directs VPEHINST to immediately remove both the VPE OPEN and CLOSE interfaces, even if statistics are lost on any currently open VSAM data sets.

RESTRICT= jobname

Directs VPE to only operate on the job specified. This restriction always takes precedence over the scope of operation specified in the VPE Rules Table and/or the VPE Control DD statements. This parameter is useful when you want to test VPE prior to implementing it for production use. The following VPEHINST example installs the VPE Operating System interfaces such that they only operate when the job name is 'TESTJOB1':

```
//VPEINST JOB ...
 //STEP1   EXEC PGM=VPEHINST,
 //       PARM='ACTION=INSTALL,RESTRICT=TESTJOB1'
//
```

VPE Operation

The operation of VPE is controlled using two methods: Special DD statements in your application JCL, and the VPE Rules Table. You can use either method or both at the same time.

VPE Special Control DD Statements

VPE Control DD statements control the operation of VPE by their presence within a jobstep. With one exception, these special DD statements do not specify data sets. Rather, special DUMMY DD statements in the jobstep and artificial *data set names* invoke VPE and control its operation. VPE does not open the artificial *data set names* you specify on these DD statements. However, be careful in coding the VPE Control DD statements which include the artificial *data set names* because the data sets are allocated and deleted by job control. These DD statements must be completely valid to job control even though they are not opened and used to create a data set. You need to code them like the following:

```
//ddname DD DSN=ddname1.ddname2.ddname3...ddnamen,
 //      UNIT=DISK,SPACE=(TRK,0)
```

ddname

The special VPE ddname

ddname1

The 1st application ddname you are referencing

ddname2

The 2nd application ddname you are referencing

ddnamen

The nth application ddname you are referencing

Unlike the VPE Rules Table, Control DD statements can be coded by anyone who wants VPE operation, without refreshing the VPE Operating System interfaces, and without the involvement of the storage administrator. They are also much more easily and quickly processed by VPE, allowing early exit from the SVC intercept code.

In order for the Special Control DD statements to invoke and control VPE, the ALLOW=YES parameter must be specified (or be allowed to default) on the SYSOPT statement in the VPE Rules Table. Conversely, the Special Control DD statement functionality can be completely disabled by coding ALLOW=NO on the SYSOPT statement in the VPE Rules Table.

Note: The VPEIGNR Control DD Statement always disables VPE in that step even if ALLOW=NO is coded in the VPE Rules Table.

DD=VPEONALV

The VPEONALV DD statement activates VPE for all appropriate VSAM files in the jobstep. In most cases, VPE can correctly select those VSAM files to optimize completely automatically. You can see which files were selected and how they were optimized by VPE on the Run Statistics Report. The syntax of the VPEONALV DD statement is:

```
//VPEONALV DD DUMMY
```

DD=VPEBUFnn

The VPEBUF DD statement specifies the target buffer pool size for the jobstep. The default is 4 Mb. The size of the buffer pool requested is specified in Mb as the last two characters of the ddname. The syntax of the VPEBUFnn DD statement is:

```
//VPEBUFnn DD DUMMY
```

For example, VPEBUF99 sets the target buffer pool size to 99 Mb. A value this large eliminates most I/O.

DD=VPEIGNR

The VPEIGNR DD statement deactivates VPE for the jobstep. This DD statement always functions and takes precedence, even if ALLOW=NO or PRECEDENCE=VPEDD is coded on the SYSOPT statement in the VPE Rules Table. The syntax of the VPEIGNR DD statement is:

```
//VPEIGNR DD DUMMY
```

DD=VPEONnn

The VPEONnn DD statement directs VPE to optimize the files referenced by the ddnames specified as nodes of the DSNAME portion of the DD statement. The *nn* portion of the DD statement is a number from *00* through "09", allowing up to 10 VPEONnn DD statements per jobstep. The syntax of the VPEONnn DD statement is:

```
//VPEONnn DD DSN=ddname1.ddname2.ddname3...ddnamen,
 // UNIT=DISK,SPACE=(TRK,0)
```

If your security system requires a specific value as the first qualifier of the data set name, VPE allows you to code any value there without requiring it to match a DD statement coded in the job. As usual for a real DD statement, the artificial *data set name* coded on this special DD statement must not exceed 44 characters in length. The following example activates VPE for the data sets specified by DD cards VSAM1 and VSAM2:

```
//VSAM1    DD DSN=MY.VSAM.FILE.ONE,DISP=SHR
//VSAM2    DD DSN=MY.VSAM.FILE.TWO,DISP=SHR
//VPEON00  DD DSN=VSAM1.VSAM2,
 //        UNIT=DISK,SPACE=(TRK,0)
```

DD=VPEONGnn

The VPEONG DD statement works the same as the VPEONnn DD statement except that the ddnames coded as nodes of the artificial *data set name* can be ddname prefixes or wild cards. The syntax of the VPEONGnn DD statements is:

```
//VPEONGnn DD DSN=ddname1.ddname2.ddname3...ddnamen,
 //        UNIT=DISK,SPACE=(TRK,0)
```

The following example activates VPE for the data sets specified by DD cards VSAM1 and VSAM2 using the ddname prefix *VSAM*:

```
//VSAM1    DD DSN=MY.VSAM.FILE.ONE,DISP=SHR
//VSAM2    DD DSN=MY.VSAM.FILE.TWO,DISP=SHR
//VPEONG00 DD DSN=VSAM,
 //        UNIT=DISK,SPACE=(TRK,0)
```

DD=VPELSRPB

The VPELSRPB DD statement directs VPE to acquire LSR pool space below the 16 Mb line. This is necessary in applications which are not MVS/XA-ready and use LOCATE mode I/O. This DD statement always functions and takes precedence, even if ALLOW=NO or PRECEDENCE=RULES is specified. The syntax of the VPELSRPB DD statement is:

```
//VPELSRPB DD DUMMY
```

DD=VPEVSTS

The VPEVSTS DD statement is a real SYSOUT DD statement which VPE uses to print error messages and statistics. If it is not included in the jobstep, then VPE dynamically allocates it. The syntax of the VPEVSTS DD statement is as follows, where *n* is a valid SYSOUT class for your JES system:

```
//VPEVSTS DD SYSOUT=n
```

DD=VPEVRPT

The VPEVRPT DD statement directs VPE to produce the VSAM Recommendations Report. See VPE Reports in this chapter for information about the report. In addition, this statement generates warning messages that reflect conditions relating to the IDCAMS definition of clusters or their current status as observed in the catalog. The syntax of the VPEVRPT DD statement is:

```
//VPEVRPT DD DUMMY
```

DD=VPEWRKnn (CICS processing only)

The VPEWRKnn DD statement directs VPE to turn on Deferred Write and Sequential Insert Strategy performance enhancement techniques for the files referenced by the ddnames specified as nodes of the DSNAME portion of the DD statement. The *nn* portion of the DD statement is a number from *00* through *09*, allowing up to 10 VPEWRKnn DD statements per jobstep. The syntax of the VPEWRKnn DD statement is:

```
//VPEWRKnn DD DSN=ddname1.ddname2.ddname3...ddnamen,
 //        UNIT=DISK,SPACE=(TRK,0)
```

If your security system requires a specific value as the first qualifier of the data set name, VPE allows you to code any value there without it requiring to match with a DD statement coded in the job. As usual for a real DD statement, the artificial *data set name* coded on this special DD statement must not exceed 44 characters in length.

VPE Rules Table

Typically, the Storage Administrator maintains the VPE Rules Table. The rules table statements are keyed into a simple text file using any text editor, such as the SPF Editor (Option 2). The statements in this text file are converted to the internal form VPE uses by running the VPERULB program. When VPE is activated during system startup, the internal form of the VPE Rules Table is loaded into Common System Area (CSA) memory, always above the 16 Mb line when it is available.

The VPE Rules Table is deleted from CSA when VPE is uninstalled, so you must run the VPERULB program each time you install VPE. The best way to accomplish this is to include the VPERULB step in your startup PROC for VPE and CA Compress.

Note: Running the VPERULB is necessary only when you use the VPE Rules Table. If you only use the VPE Control DD Statements, it is not necessary to run this program.

VPE Rules Table Syntax Rules

The following section describes the syntax rules of VPE Rules Table.

- Any line which starts with an asterisk (*) in column 1 is a comment line and is ignored by the VPERULB program.
- A statement consists of the statement name followed by one or more parameters. For example:

  ```
  SYSOPT ALLOW=NO,PRECEDENCE=VPEDD
  ```

- The first noncomment line must be a SYSOPT or VSAM statement. The SYSOPT and VSAM statements must appear before all RULE statements.
- A statement can span more than one line.
- A parameter may not span more than one line.
- A parameter followed by a blank terminates a statement. Characters following the blank are considered comments.
- A parameter followed by a comma and a blank continues a statement onto the next line. Characters following the blank are considered comments. The parameters appearing on the next line can start in any column.
- Data set names can be enclosed in quotes or not. An asterisk ("*") in a data set name is a wild card character. It is not node-specific. For example, "A*T" matches DSN=ABCTEST as well as DSN=ABC.NODE.TEST.

Note: A *node* is one portion of the data set name delimited by a period. The nodes of ABC.NODE.TEST are *ABC*, *NODE*, and *TEST*.

VPE Rules Table Source Statements

There are three rule statements. Each statement can have one or more parameters.

SYSOPT

Sets global system options.

VSAM

Sets default VSAM processing options.

RULE

Sets processing options for individual files, jobs, and jobsteps.

The SYSOPT and VSAM statements specify the global and default processing options for your system. The RULE statement specifies exceptions to these global and default options. The flexibility to define your own custom global and default processing options and then override them for specific files, jobs, and jobsteps allows you to minimize the number of statements you need to compose in order to implement VPE just the way you want.

SYSOPT Statement

The following section explains the SYSOPT statement.

```
SYSOPT  [ALLOW=YES|NO]
[,PRECEDENCE=VPEDD|RULES]
```

The SYSOPT statement sets global system options. There can be at most one SYSOPT statement. It can be omitted if desired. These options set the default values for your system. They are global in nature and apply to all jobs, steps, and VSAM files unless overridden by more specific rules or special VPE DD cards.

ALLOW=YES|NO

Specifies whether the special VPE DD cards are active or not. If ALLOW=YES is specified (the default), then the special VPE DD cards will affect the operation of VPE. If ALLOW=NO is specified, then any special VPE DD cards will not affect the operation of VPE. Exceptions: The VPEIGNR and VPEVIGNR Control DD cards always disable VPE in that jobstep.

PRECEDENCE=VPEDD|RULES

Specifies which of the two VPE controlling mechanisms takes precedence when there is a conflict between them. If PRECEDENCE=VPEDD is specified (the default), then the special VPE DD statements present in the application JCL take precedence over any conflicting rules in the VPE Rules Table. If PRECEDENCE=RULES is specified, then the opposite is true. This parameter only makes sense if ALLOW=YES is specified.

VSAM Statement

The following section explains the VSAM statement.

```
VSAM    [POOL=n]
      [,BUFFRLOC=ABOVE|BELOW]
      [,SIS=NO|YES]
      [,VSAMREC=NO|YES]
      [,TGTBUF=nnM]
      [,MINBUF=nnM]
      [,MINRESV=nnnK]
```

The VSAM statement sets global system options. There can be at most one VSAM statement. It can be omitted if you want. These options are used to tune VPE. Like the SYSOPT statement, these options set the default values for your system. They are global in nature and apply to all jobs, steps, and VSAM files unless overridden by more specific rules or special VPE DD cards.

POOL=n

Specifies the LSR pool number which VPE is to use for I/O buffers. The default value is 3.

This is the pool number and not the number of pools. VSAM supports 16 LSR pools for data components and 16 for index components. Both sets of pools are numbered 0 through 15. POOL= specifies which of these pairs of pools are to be used.

BUFFRLOC=ABOVE|BELOW

Specifies from where the LSR buffer space is to be obtained, above or below the 16 Mb line.

Note: In either case, a VPE RULE statement or special VPE DD statement can override this global value for the data sets affected by them.

SIS=NO|YES

SIS=YES directs VPE to globally turn on VSAM's Sequential Insert Strategy (SIS) for all applicable ACBs.

SIS=NO (the default) directs VPE to not do so.

VSAMREC=NO|YES

VSAMREC=YES directs VPE to globally produce the VSAM Recommendations Report for every VSAM data set open.

VSAMREC=NO (the default) directs VPE to not do so.

Note: A VPE RULE statement or special VPE DD statement can override this global value for the data sets affected by them.

TGTBUF=nnM

directs VPE to attempt to obtain the specified number of megabytes for the VSAM I/O buffer. If VPE cannot obtain this much space, it reduces the request until the space request succeeds, or until the minimum buffer size is reached (see the MINBUF=nnM parameter below). The default target buffer size is 9 Mb if the LSR buffers are above the 16 Mb line, and 4 Mb if they are below it.

MINBUF=nnM

Works in conjunction with the ABENDNOMIN=YES parameter of a RULE statement. MINBUF=nnM specifies the minimum VSAM I/O buffer size which is acceptable. The default is zero. If a nonzero value is specified for MINBUF=nnM and ABENDNOMIN=YES is specified, VPE abends the job step if the minimum buffer space cannot be obtained.

MINRESV=nnnnK

Specifies the minimum amount of memory to reserve below the 16 Mb line for the application to use. The default setting is 300K. VPE does not enhance sequential file buffering if this amount of memory is not available. When LSR buffers are being acquired below the 16 Mb line, this value reduces the size of the VSAM I/O buffers if this amount of space is not available.

RULE Statement

The following section explains the RULE statement.

```
RULE    (Scope of the rule)
    [INCLUDE|XCLUDE]
     [,JOB=cccccccc|STEP=cccccccc|DDNAME=cccccccc|DSN=cccccccc]
     (Job/Step specified parameters)
     [,POOL=n]
     [,TGTBUF=nnM]
     [,MINBUF=nnM]
     [,ABENDNOMIN=NO|YES]
     [,MINRESV=nnnK]
     (DDNAME/DSN specific parameters)
     [,POWERFACT=nn]
     [,SIS=NO|YES]
     [,DFW=NO|YES]
     [,FORCEMODE=SEQ|LSR]
     [,BUFND=nn]
     [,BUFNI=nn]
     [,BUFSP=nn]
     [,VSAMREC=NO|YES]
```

The RULE statement specifies VPE processing options for jobs, steps, and data sets. The values specified on a RULE statement overrides the corresponding values specified in the SYSOPT and VSAM statements.

INCLUDE|XCLUDE

Specifies whether this RULE statement is including jobs, jobsteps, or data sets, or excluding them.

JOB=cccccccc|STEP=cccccccc|DDNAME=cccccccc| DSNAME=cccccccc

Specifies the job name, step name, DDNAME, or DSNAME of the job, step, or data set being included in or excluded from VPE processing using this RULE statement. The values specified can contain one or more asterisks, which represent wild card indicators. Note that for data set names wild card characters are not node-specific. For example: "A*T" matches DSN=ABCTEST and DSN=ABC.NODE.TEST.

Note: A *node* is one portion of the data set name delimited by a period. The nodes of ABC.NODE.TEST are *ABC*, *NODE*, and *TEST*.

JOB/STEP Specific Parameters

These parameters are valid only when the RULE statement specifies JOB= or STEP=.

POOL=n

Specifies the LSR pool number which VPE is to use for I/O buffers for the job(s), step(s), or data set(s) affected by this RULE statement. The default value is 3.

This is the pool number and not the number of pools. VSAM supports 16 LSR pools for data components and 16 for index components. Both sets of pools are numbered 0 through 15. POOL= specifies which of these pairs of pools is to be used. This value need not match the value specified on the POOL parameter of the VSAM statement.

TGTBUF=nnM

Directs VPE to attempt to obtain the specified number of megabytes for the VSAM I/O buffer for the job(s), step(s), or data set(s) affected by this RULE statement. If VPE cannot obtain this much space, it reduces the request until the space request succeeds, or until the minimum buffer size is reached (see the MINBUF=nnM parameter below). The default target buffer size is 4 Mb.

Note: This parameter overrides the TGTBUF parameter on the VSAM statement.

MINBUF=nnM

Works in conjunction with the ABENDNOMIN=YES parameter. MINBUF=nnM specifies the minimum VSAM I/O buffer size which is acceptable for the job(s), step(s), or data set(s) affected by this RULE statement. The default is zero. If a nonzero value is specified for MINBUF=nnM and ABENDNOMIN=YES is specified, VPE abends the job step if the minimum buffer space cannot be obtained.

ABENDNOMIN=NO|YES

ABENDNOMIN=YES directs VPE to abend the jobstep when the minimum buffer space specified by MINBUF= cannot be obtained. Use this parameter whenever it is unacceptable for a jobstep to run without buffering.

MINRESV=nnnnK

Specifies the minimum amount of memory to reserve below the 16 Mb line for the application to use for the job(s), step(s), or data set(s) affected by this RULE statement. The default setting is 300K. VPE does not enhance sequential file buffering if this amount of memory is not available. When LSR buffers are being acquired below the 16 Mb line, this value reduces the size of the VSAM I/O buffers if this amount of space is not available.

VPE Tuning Tip: If you find that TGTBUF= (or even MINBUF=) requirements cannot be met, reduce MINRESV=.

Data Set Specific Parameters

These parameters are valid only when the RULE statement specifies DDNAME= or DSNAME=:

POWERFACT=nn

Directs VPE to obtain additional buffers for the data set(s) specified on the RULE statement. This parameter provides a means of *distributing* the available buffer space on a file-by-file basis. POWERFACT= does not affect the size of the buffer pool (see TGTBUF=). The default value is 1.

SIS=NO|YES

SIS= directs VPE to activate VSAM's Sequential Insert Strategy for the data set(s) specified on the RULE statement. The default value is SIS=NO.

DFW=NO|YES

DFW= directs VPE to activate Deferred Write for the data set(s) specified on the RULE statement. The default is DFW=NO.

Note: Setting SIS=YES and DFW=YES is equivalent to the CICSWRK control DD.

FORCEMODE=SEQ|LSR

FORCEMODE= directs VPE to force either sequential performance enhancements or LSR optimization, if possible, for the data set(s) specified on the RULE statement. There is no default for this parameter. When FORCEMODE= is omitted, VPE uses an algorithm to automatically select one type of optimization or the other.

BUFND=nn

BUFND= specifies the number of buffers VPE is to acquire for the VSAM data component. This parameter overrides VPE's automatic computations.

BUFNI=nn

BUFNI= specifies the number of buffers VPE is to acquire for the VSAM index component. This parameter overrides VPE's automatic computations.

BUFSP=nn

BUFSP= specifies the size of the VSAM buffer space. This parameter overrides VPE's automatic computations.

VSAMREC=NO|YES

VSAMREC= directs VPE to produce the VSAM Recommendations Report.

Usage Notes

VPE's Operating System interfaces receive control during VSAM OPEN and CLOSE. During each VSAM file OPEN, VPE will process each RULE statement looking for a match on job, step, or data set. The first RULE statement which matches is the one which is used for the VSAM file being opened. The sequence of the RULE statements determines how your implementation of VPE performs. Code the most specific RULE statements prior to the more general ones. For example, data set specific RULE statements should appear before step specific ones.

There is no limit placed on the number of RULE statements you can use. However, the more RULE statements you code, the more processing is done by VPE during VSAM file open time. For best results, keep the number of RULE statements below 500.

Because the RULE statements can be coded to only affect certain jobs, steps, or data sets, you can use them to safely experiment with different combinations of parameter values. Then, when you have determined good parameter values which should apply to all data sets by default, you can move the parameter settings to the SYSOPT and VSAM statements.

If the ACB for the VSAM file being opened specifies DIR processing and the file has more than one record in it, VPE normally attempts to implement LSR processing for the file. If you know that the file is actually processed sequentially, FORCEMODE=SEQ forces VPE to implement sequential processing enhancements for the file.

If the VSAM file being opened is empty or has only one record, or if the file's ACB specifies SEQ processing, VPE normally attempts to implement sequential processing enhancements for the file. If you know that the file is loaded by the program and then updated in DIR mode, and if the update is quite large, specify FORCEMODE=LSR to get the greater benefit of LSR enhancements for the file.

Advanced Topics

The following topics are some special considerations you should make in certain situations.

VPE Implementation Considerations

There are some VPE implementation considerations:

- Region Size
- Sequentially Accessed VSAM Files
- SHAREOPTION=4 Files
- MACRF=RLS files
- Checkpoint/Restart
- Job Swaps

Region Size

In most cases, VPE functions without region size changes. However, a larger region enhances sequential access and may be required if LSR below the line is used. VPE writes messages, which indicate if a larger region would enhance buffering. If VPE does not ask for a larger region size, increasing the region does not improve performance.

Note: In order to avoid making changes to each job's region size, you can set IEALIMIT using an SMF exit.

LSR above the line can be used. The additional storage used below the line by VPE for code and tables is minimal.

Region Computations

VPE's region computations involve the amount of space available below the line at the time of first OPEN plus the following VPE parameters. These parameters are set in the VPE Rules Table and/or by VPE Control DD statements:

- MINRESV—Specifies the minimum amount of memory to reserve below the 16 Mb line for the application to use. The default setting is 300K.
- TGTBUF—Directs VPE to attempt to obtain the specified number of megabytes for the VSAM I/O buffers. This defaults to 9 Mb if the LSR buffers are above the 16 Mb line, and to 4 Mb if they are below it.
- MINBUF—Specifies the minimum VSAM I/O buffer size, which is acceptable. The default value for MINBUF is zero bytes. MINBUF can be used to generate error messages or to prevent jobs from running when sufficient buffer space is not available.

VPE optimizes buffer space on files opened for load or sequential only access (high RBA = 0 or 1 or ACB specifies sequential only access) using space from below the line. VPE uses the following calculation to obtain a value for the upper limit of size of sequential buffers to assign:

Case 1: LSR below the line:

```
LIMIT = (AVAIL REGION BELOW 16MB - LSR POOL SIZE / 2).
```

Case 2: LSR above the line:

```
LIMIT = (AVAIL REGION BELOW 16MB - MINRESRV)/2.
```

In Case 1, the LSR pool size is set to the smaller of TGTBUF or the available region minus MINRESRV. In Case 2, the entire MINRESRV is available for the application. In Case 1, a minimum of half of the region specified by MINRESRV is available.

Note: This amount of region is not actually taken, but is merely a calculated upper limit. A message is written if the minimum calculated LSR pool is not available or if the sequential limit is reached prior to fully optimizing all files.

Sequentially Accessed VSAM Files

If a VSAM file has a high RBA of 0 (that is, the file is empty), has only one record in it, or if the ACB opened for the file has sequential processing specified, VPE assumes the file is in a load state. A buffer space target is created such that full track read/write operations are performed for the data portion of the file and each index level has an index buffer. It is possible some applications load these files and then update them. If the update is large, use the VPE rule statement FORCEMODE=LSR to speed these jobs up more than the default sequential buffering.

SHAREOPTION=4 Files

This VSAM processing option allows multiple CPUs or address spaces to have concurrent write access to a data set. Unless your environment requires this type of access, it is generally inadvisable. Specifying SHAREOPTION=4 requires VSAM to perform a physical I/O to refresh the buffer contents each time the record is read or written, so buffering these files is useless and VSAM does not permit LSR for such data sets. Review your cluster definitions to ensure that SHAREOPTION=4 is still appropriate.

MACRF=RLS (Record Level Sharing)

Record Level Sharing is incompatible with LSR and with BUFND, so VPE cannot optimize such OPENs. VPE excludes these data sets from optimization.

Checkpoint/Restart

Checkpoint/Restart can be used with data sets when VPE is being used, but processing restarts at the data set's high-used RBA, rather than the point where the checkpoint was taken. This occurs because VPE uses LSR and this is how LSR affects Checkpoint/Restart. If your application currently depends on Checkpoint/Restart, become familiar with the difference in restart processing for files using LSR before using VPE with that application. See IBM's *Checkpoint / Restart Guide*, which contains information about restart processing for VSAM files.

Job Swaps

VPE increases the amount of virtual storage used by your application. The amount of virtual storage used in a job step is one of the criteria that the MVS System Resource Manager uses to determine which job to swap out when there is too much contention for memory resources. This function is a very important part of MVS's ability to ensure optimum response for time-critical systems such as CICS, as it provides the means to ensure that priority systems obtain the memory resources they require for good response times.

Unfortunately, this same evaluation criteria works against optimum performance during the off-prime hours when online systems are largely idle and the CPU is doing more batch than interactive work. Under these circumstances, the bias in favor of online systems and lower multiprogramming levels can lead to bottlenecks and poor memory utilization. This typically results in high overall CPU utilization, long elapsed run times, high swap counts for large batch jobs, and relatively light I/O contention (because jobs spend a great deal of time swapped out). Should you find that VPE has relieved your I/O bottleneck only to create a memory problem, ask your MVS tuner about raising domain minimums for your performance group. This can be done by command to test the effect and made permanent using changes to IEAICS*xx* parameters. If you are still not satisfied, contact CA Technical Support for assistance.

Optimizing VSAM Performance by Adjusting VSAM Parameters

Although VPE automatically alleviates many of the performance problems which occur with poorly defined files, you need to correct flagrant violations and problems. This is especially true for large or heavily accessed files. Poorly defined files always have a negative effect on processing, even when VPE reduces the effect through intelligent buffering. For example, a file with excessive index levels requires excessive reads to access a record without VPE. With VPE, the excessive reads are exchanged for excessive accesses to the core buffers. This results in wasted CPU processing time accessing the buffers. Storage is still wasted containing the buffers. In a memory constrained system, the I/O overhead of paging and swapping may equal the cost of saved index reads. There is no substitute for properly tuned systems.

The options specified when a VSAM data set is created have a great effect on both its space usage and overall performance. For some options the defaults are adequate, but for others they are a poor choice. VPE generates warning messages for poorly defined files to aid in optimizing VPE performance. The discussion which follows describes situations which cause VPE to produce these messages and how you can correct the problems by adjusting parameters on the IDCAMS Define Cluster statement. CISZ (Data Component CI Size)

VPE flags all data component CI sizes of less than 2048 bytes. These small CI sizes result in reduced space utilization. Larger data component CI sizes result in improved space utilization. In addition, small data component CI sizes increase the number of physical reads required to process a file sequentially. These small sizes are generally a holdover from 3330 disk devices and channels with speeds of 1.5 Mb-per-second.

CISZ (Data Component CI Size)

VPE flags all data component CI sizes of less than 2048 bytes. These small CI sizes result in reduced space utilization. Larger data component CI sizes result in improved space utilization. In addition, small data component CI sizes increase the number of physical reads required to process a file sequentially. These small sizes are generally a holdover from 3330 disk devices and channels with speeds of 1.5 Mb-per-second.

SPACE (Data Component CA Size)

Data component CA size is controlled by the space allocation specified for the data component. Generally, it is one cylinder, but it can be as small as one track. VSAM sets the data component CA size to the lesser of the primary allocation, the secondary allocation, and one cylinder. VSAM creates sequence sets such that they hold one compressed key per data component CI in a data component CA. This means that the data component CA size controls the number of index records VSAM retrieves on one read (one index component CI) as well as the number of index levels that are required to point to the sequence sets. Because obtaining more index records per read benefits both random and sequentially accessed files, it is generally best to use cylinder CAs. This means all but very small files should be allocated in cylinders.

An exception to the above guideline is files that have high browse update activity in CICS. In this case, a small data component CA size can reduce the number of browse enqueue locks because the number of index entries in the sequence set is smaller. Therefore, fewer entries are enqueued by browse. However, to achieve optimal performance, replace the browse operations with *Get Key Greater than or Equal*. This replacement eliminates the sequence set enqueue.

CISZ (Index Component CI Size)

The lowest level of the index is called the sequence set. It contains one pointer (compressed key) for each data component CI in a data component CA. This means the index component CI size must be at a minimum equal to the compressed key size times the number of data component CIs in a data component CA plus the overhead.

The index set is all levels of the index above the sequence set. The highest level of the index set must fit in a index component CI. So, if the total number of data component CAs is greater than the number of keys which can be held in one index component CI, the index has more than one index level. When the indexes are not in buffers, a physical read must be performed for each index level to access a data record. Without an explicit definition for the index component CI size, VSAM automatically selects a CI size based on assumptions it makes about key compression. Frequently, these assumptions over-estimate the amount of compression present. Therefore, VSAM creates an index component CI that is too small to address all data component CIs within one CA. Because the remaining data component CIs cannot be used, space is wasted. You can correct this problem by increasing the index component CI size to reduce the amount of space used by the file.

In addition to wasting space, small index component CI sizes can cause VSAM to add additional levels to the index set. The additional levels increase the number of I/Os required to locate data. LSR reduces the impact of this problem by keeping index component CIs in storage. However, LSR processing is not appropriate for certain applications, specifically SHAREOPTIONS=4 and sequential processing. In these cases, the number of physical I/Os required to locate your data can have a great impact on performance. As a rule of thumb, files should have no more than one index level per 500 cylinders of data, plus one each for the base index and sequence set. VPE issues a message informing you how many index levels the file has when this is exceeded.

When in doubt, it is better to have an index component CI that is too large than one that is too small. For most files, index space is a small fraction of data space. If the index component CI is too large, little space is wasted. If it is too small, major overhead is added to the file access and a significant amount of data space can be wasted.

The factors to be considered when selecting an index component CI size are:

- The key size and compression.
- The data component CI size (index and data component CI sizes should be different to avoid buffer contention).
- The number of data component CIs per CA.
- The data component CA size.
- The overall file size (total number of CAs).

Because an assessment of actual key compression is not possible without scanning the full sequence set and the penalty for undersized index component CIs is large, VPE issues a warning when the index component CI size specified is less than a factor of the number of CIs per CA multiplied by the full key length. VPE uses factors similar to those internal to VSAM, but more conservative. If you receive this message, go to the next higher CI size or use IDCAMS to examine the indexes to determine if any are actually full. If the index change has a positive effect, increase it again until no further change takes place. If it has no effect, ignore any further occurrences of this message. Remember there are NO circumstances under which having an index component CI one size too large causes any performance degradation.

SHAREOPTIONS

Specifying SHAREOPTIONS=4 (SHAREOPTIONS=(n,4) or SHAREOPTIONS=(4,n)) informs VSAM that the file is being used by multiple systems. This forces a physical READ or WRITE every time a record is accessed, so no benefit is achieved by using LSR or extra buffers. VSAM fails the OPEN if SHAREOPTIONS=4 and LSR are both specified. So, if VPE encounters this option, it excludes this file in the buffer pool and issues a warning. Do not use SHAREOPTIONS=4 unless you need it.

Note: If a cluster has SHAREOPTIONS=(n,4) specified, any of its alternate indexes that were created with the option of UPGRADE are assumed to be SHAREOPTIONS=(n,4) even if they were not defined that way. The same is true for a cluster that is not SHAREOPTIONS=(n,4), if any one of its upgrade AIXs were created with SHAREOPTIONS=(n,4).

IMBED

IMBED causes VSAM to place the sequence set in the first track of each CA and to replicate the sequence set around the track until it is full. This option normally allows one seek to satisfy the read for both the sequence set and the data it references, but it increases the amount of space required to contain the file by a minimum of about seven percent and reduces the file integrity. It can be up to 50 percent, if the subcylinder allocations are used. The latter occurs because VSAM must now update the indexes multiple times.

A typical problem with IMBED is records that are sometimes not found. This occurs because not all of the sequence sets on the track are updated. This only happens when the file is updated from two regions at once without the use of SHAREOPTIONS=4; but this is not uncommon, as few shops can live with the poor performance of SHAREOPTIONS=4. The use of IMBED is not recommended. If performance dictates, place the index and data portions of the cluster on separate packs. VPE warns you if IMBED is specified for a file that is larger than 200 cylinders.

WRITECHECK

This option causes VSAM to read each record written. It is of no use with modern DASD. VPE warns you if your file is defined with WRITECHECK.

REPLICATE

This option causes index records to be replicated around the track using IMBED for sequence sets. If sufficient buffers exist to hold the entire index set in memory (which is generally the case), no performance benefit is gained from REPLICATE. This option wastes DASD and invites problems. VPE warns you if your file is defined with REPLICATE.

SPEED

SPEED is a performance option that helps reduce the amount of time it takes to load a VSAM file. It has no hidden cost, so it is a good idea to use it. If you do not specify SPEED, RECOVERY is automatically used. RECOVERY preformats the entire data set allowing load jobs to restart. VPE issues a message if the cluster was defined without SPEED.

FREESPACE

This parameter reserves space in either the CI or CA at load time so that new records or new CIs are usually written without splits taking place. FREESPACE works when file updates are nonclustered in a key range. It can work with clustered updates, but the space is allocated to the entire file to help one CI or CA and thus it is wasted. If many updates/additions take place with similar keys, large amounts of free space are needed to prevent splits. In fact, the maximum value (100,100), which puts one record in a CI and one CI in a CA, cannot be enough.

CI splits occur when not enough free space is available to contain a record, which is being inserted. Without Sequential Insert Strategy (SIS), VSAM must move half the data in the CI to a new CI before continuing with the insert operation. These splits result in CPU and I/O overhead and should be avoided. In addition, logically ascending data is no longer physically adjacent so any sequential access to the file is slowed down.

If CI splits occur over the period of several days, the file needs to be reorganized. If they occur in a period of a day or less, increase the CI FREESPACE parameter and reorganize the file. CI free space is a percentage of the CI size and must be large enough to insert a whole record.

Whereas CI splits represent minimal to moderate overhead, the amount of processing incurred by a CA split is much greater. A CA split occurs when there is not enough room to contain the new CI created during a CI split. When this happens, VSAM must create a new control area, move 50 percent of the data from the old CA to the new one, update the sequence set, update part of the index set, and then continue with the original CI split. With Sequential Insert Strategy (SIS), this process is similar except the old CA is split at the CI being split instead of in the middle. With cylinder CAs (no SIS), both half a cylinder of data is moved and a minimum of one level of index is located and updated before the original CI split can occur. Like CI splits, CA splits call for reorganizations and FREESPACE.

As mentioned above, CA splits are of greater concern than CI splits. Generally it is a matter of how much free space you can afford to allocate to a file. Putting free space in a file increases the number of data CIs and CAs, thereby increasing index CI size requirements and levels as well as the file's space allocation. Splits are probably the number one cause of poor CICS response time. All the queuing that takes place while CICS is nondispatchable uses a lot of CPU time, requires additional I/O, and makes CICS less responsive. Although CICS puts a file in LSR, it does not activate Deferred Write (DFR), meaning CICS waits on physical I/Os during splits. VPE can help by turning on Sequential Insert Strategy (SIS) and Deferred Write (DFW) for files that CICS has put into LSR as well as to identify which files contain splits.

A split in an index component CI or CA indicates the index component CI is undersized. In addition to increasing the index CI size, you can give the index component its own FREESPACE.

Multiple Extents

Multiple extents occur either when the file needs more space than is available in the primary allocation or when the pack your file is on is so fragmented that multiple extents are required to obtain the space you ask for in the primary allocation. Both of these conditions can increase seek time. Correct this problem by increasing the size of the primary allocation, or if fragmentation is the problem, reorganize the pack. VPE issues a warning when a file contains multiple extents.

Tuning VPE's Buffer Size

When you first run VPE on a file, start with the default buffer size and review the performance statistics generated in the VSAM Recommendation Report to determine how effective that size is for your application. Run the job several times, with different buffer sizes. While you are incrementally decreasing the buffer size, you will eventually reach the point where a continued decrease of the buffer pool size for a particular job will cause the percent of I/O requests which are being satisfied from the buffer pool to decrease dramatically. For optimum performance, do not go below this value. While you are incrementally increasing the buffer size, a point will be reached where adding more buffer space does very little for the look-aside ratio. For most applications, a look-aside ratio as high as 90 percent can be obtained on all files. Some applications will get even higher ratios with the addition of more buffers, but beyond the 90 percent point the decrease in run time will usually be small. In a memory-constrained environment, the use of too large a value can cause a performance decrease, due to additional paging in the system.

Buffer size changes affect the number of times your job is swapped out. When it is out, it is not doing anything. If swapping is not going up and run times are still out of proportion with CPU consumption, check the following:

- Excessive Program Loads—Correct excessive loads from STEPLIB or JOBLIB by canceling the JOB after it has run a while and inspecting the use count in the CDEs.
- Excessive OPENS or CLOSES—Correct this problem by performing a GTF trace. This trace identifies both OPEN/CLOSE and LOAD problems.
- Enqueue Conflicts—Examine your dumps or use a third party program to identify enqueue problems.

Assembly Programming Limitations

VPE looks out for the use of certain assembly programming techniques which are not supported by VSAM when LSR pools are in use. VPE excludes such files from VPE's optimizations because VPE uses LSR pools. The affected assembly programming techniques are:

- CBIC—Control Blocks In Common (specified on the ACB macro).
- ICI—Improved Control Interval access (specified on the ACB macro).
- UBF—User Buffering (specified on the ACB macro).

VPE Reports

Examples of VPE reports are displayed for the following:

- VPE Initialization and Setup Statistics
- VSAM Recommendation Report
- Performance Statistics Report

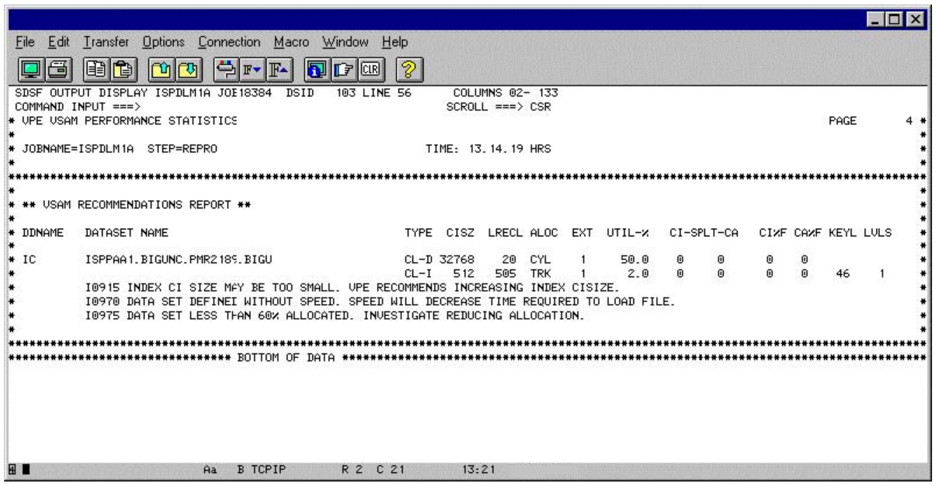
VPE Initialization and Setup Statistics

The following sample shows the VPE initialization and setup statistics report.

```
SDSF OUTPUT DISPLAY ZSAMSTC  STC01199  DSID   101 LINE 0       COLUMNS 02- 81
 COMMAND INPUT ===>                                            SCROLL ===> 0011
********************************* TOP OF DATA **********************************
*
********************************************************************************
*
* VPE VSAM PERFORMANCE STATISTICS
*
* JOBNAME=ZSAMSTC   STEP=ZSAMSTC            DATE: 08/12/12  TIME: 23.37.26 HRS
*
*
* VPE VSAM PERFORMANCE STATISTICS
*
* JOBNAME=ZSAMSTC   STEP=ZSAMSTC            DATE: 08/12/12  TIME: 23.37.26 HRS
*

********************************************************************************
*
* VPEHINST-I1567 PARMS->> ACTION=INSTALL
********************************************************************************
*
* VPEHINST-I1200 SVC 19 (OPEN) AND 20 (CLOSE) SUCCESSFULLY INTERCEPTED.
*                VSAM PERFORMANCE ENHANCER IS NOW INSTALLED.
*                REMINDER - ALL SVC ROUTINES MUST BE DEINSTALLED IN LIFO ORDER.
********************************************************************************
*
* VPEHINST-I1111 SVC TBL VALUE FOR SVC 19 IS AT 0142C950 SET TO 00DD8600 TO RESTORE.
********************************************************************************
*
* VPEHINST-I1111 SVC TBL VALUE FOR SVC 20 IS AT 0142C958 SET TO 00DC4000 TO RESTORE.
********************************************************************************
*
* VPEHINST-I1110 VPEHKS19(OPEN) IS AT 869E7DD0. IT HAS BEEN ACTIVE   SINCE 08/12/12
 ********************************************************************************
```

VSAM Recommendation Report

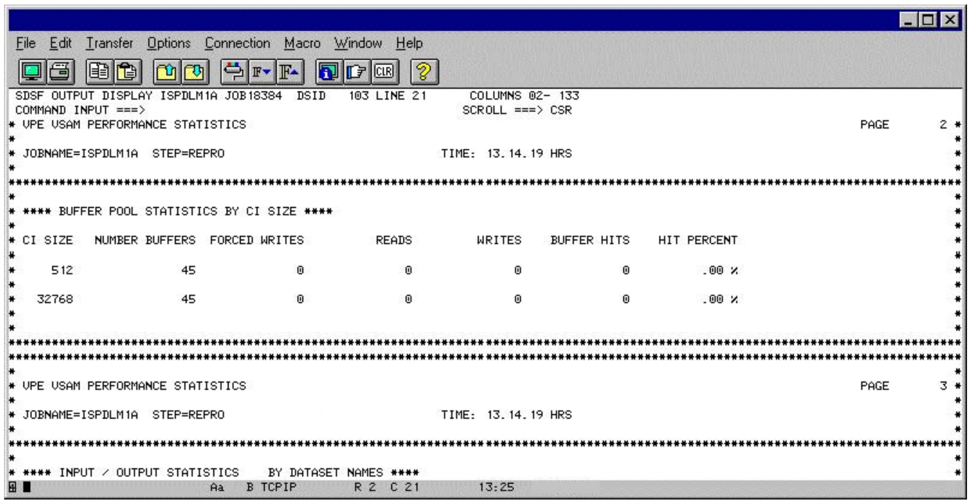
The following image shows the VSAM recommendation report.



```
 SDSF OUTPUT DISPLAY ISPDLM1A JOE18384  DSID   103 LINE 56      COLUMNS 02- 133
 COMMAND INPUT ===>                                            SCROLL ===> CSR
* VPE VSAM PERFORMANCE STATISTICS                                                                                           PAGE      4 *
*                                                                                                                                       *
* JOBNAME=ISPDLM1A  STEP=REPRO                             TIME:  13. 14. 19 HRS                                                        *
*                                                                                                                                       *
*****************************************************************************************************************************************
*                                                                                                                                       *
* ** VSAM RECOMMENDATIONS REPORT **                                                                                                     *
*                                                                                                                                       *
* DDNAME   DATASET NAME                                      TYPE  CISZ  LRECL ALOC  EXT  UTIL-%   CI-SPLT-CA   CI%F CA%F KEYL LVLS    *
*                                                                                                                                       *
* IC       ISPPAA1.BIGUNC.PMR2189.BIGU                       CL-D 32768    20  CYL    1     50.0    0     0      0    0                 *
*                                                            CL-I   512   505  TRK    1      2.0    0     0      0    0    46     1     *
*          I0915 INDEX CI SIZE MAY BE TOO SMALL. VPE RECOMMENDS INCREASING INDEX CISIZE.                                                *
*          I0970 DATA SET DEFINEI WITHOUT SPEED. SPEED WILL DECREASE TIME REQUIRED TO LOAD FILE.                                        *
*          I0975 DATA SET LESS THAN 60% ALLOCATED. INVESTIGATE REDUCING ALLOCATION.                                                     *
*                                                                                                                                       *
*****************************************************************************************************************************************
******************************** BOTTOM OF DATA *****************************************************************************************

                           Aa   B TCPIP        R 2  C 21        13:21
```

Performance Statistics Report

The following image shows the performance statistics report.



```
 SDSF OUTPUT DISPLAY ISPDLM1A JOB18384  DSID   103 LINE 21      COLUMNS 02- 133
 COMMAND INPUT ===>                                            SCROLL ===> CSR
* VPE VSAM PERFORMANCE STATISTICS                                                                                           PAGE      2 *
*                                                                                                                                       *
* JOBNAME=ISPDLM1A  STEP=REPRO                             TIME:  13. 14. 19 HRS                                                        *
*                                                                                                                                       *
*****************************************************************************************************************************************
*                                                                                                                                       *
* **** BUFFER POOL STATISTICS BY CI SIZE ****                                                                                           *
*                                                                                                                                       *
* CI SIZE   NUMBER BUFFERS  FORCED WRITES         READS         WRITES    BUFFER HITS    HIT PERCENT                                  *
*                                                                                                                                       *
*     512              45              0              0              0              0         .00 %                                 *
*                                                                                                                                       *
*   32768              45              0              0              0              0         .00 %                                 *
*                                                                                                                                       *
*                                                                                                                                       *
*****************************************************************************************************************************************
*****************************************************************************************************************************************
*                                                                                                                                       *
* VPE VSAM PERFORMANCE STATISTICS                                                                                           PAGE      3 *
*                                                                                                                                       *
* JOBNAME=ISPDLM1A  STEP=REPRO                             TIME:  13. 14. 19 HRS                                                        *
*                                                                                                                                       *
*****************************************************************************************************************************************
*                                                                                                                                       *
* **** INPUT / OUTPUT STATISTICS    BY DATASET NAMES ****                                                                               *
                           Aa   B TCPIP        R 2  C 21        13:25
```

Chapter 8: Exclusion Facility

Excluding certain jobs, modules, and programs from CA Compress processing improves performance by reducing both I/O and CPU time. More importantly, exclusion can prevent programs from compressing already compressed data.

The CA Compress Exclusion Facility is controlled by a sequential data set or a member of a PDS that is pointed to by the SYSIN DD of the CA Compress started task. The DCB attributes of the file are LRECL=80, RECFM=FB, and any BLKSIZE equal to a multiple of 80. It contains definition statements for the CA Compress address space. The statements include nine table names and their associated record entries.

The tables include the names of jobs, programs, and modules for which CA Compress processing should not take place. If a job, program, or module name is in these tables, GET for a Compress data set retrieves compressed data, and a PUT stores the record as is, without compression.

This section contains the following topics:

Exclusion for VSAM Backup/Restore Processing

You can use the EXCLUDE-JOB, -PGM, and -MOD tables to prevent BACKUP/RESTORE facilities from doing compression or expansion. The backup functions of CA FAVER, DFDSS, and CA Disk retrieve data from a compressed cluster without expansion and store data compressed. However, their restore functions use VSAM record management to store the compressed backup data back into the cluster at restore time. CA Compress is invoked when the VSAM cluster is opened for output. Unless the restore functions of such products are excluded, compressed data from backups is compressed again.

Exclusion for Physical Sequential Transparency Processing

For similar reasons, the EXCLUDE-JOB-PST, EXCLUDE-PGM-PST, and EXCLUDE-MOD-PST tables prevent compression and expansion of PST data sets. For instance, it is not uncommon to sort or select certain compressed records based on uncompressed fields, without actually expanding them. In such cases, processing uncompressed not only adds overhead, but forces SORT control cards to change.

Because the exclusion facility causes compressed data sets to be processed without CA Compress, be sure when coding DCB parameters that you specify the compressed attributes. Otherwise, CA Compress does not realize that the data set is compressed, and other errors can occur.

Exclusion to Prevent Control-Interval (CI) Processing and EXCP

CA Compress does not expand or compress data when invoked by a program using EXCP or Control-Interval (CI) processing. The tables *EXCLUDE-JOB-SORT, *EXCLUDE-PGM-SORT and *EXCLUDE-MOD-SORT force programs performing CI access, such as SYNCSORT, to use record management access so that the data is compressed and expanded correctly. The Physical Sequential Transparency reallocates PST data sets before giving control to SORT and other programs, in order to make them use BSAM instead of EXCP. This enables CA Compress to compress and expand logical records.

The Exclude File

The CA Compress Exclude File contains three types of information: table statements, record entries, and comment statements.

Table Statements—A table statement is specified by the prefix character "*" in column 1, followed by a table name. The table names can be specified only once and must be syntactically correct, but can be in any order. The names and functions of the statements are described in the following section.

EXCLUDE-JOB

Excludes jobnames from VSAM compression and expansion.

EXCLUDE-MOD

Excludes VSAM compression and expansion for subroutine modules called from a processing program.

EXCLUDE-PGM

Excludes VSAM compression and expansion for programs invoked by an EXEC statement within a job or proc step.

EXCLUDE-JOB-SORT

Forces programs performing CI access to use record management access.

EXCLUDE-MOD-SORTForces programs performing CI access to use record management access.

EXCLUDE-PGM-SORT

Forces programs performing CI access to use record management access.

EXCLUDE-JOB-PST

Excludes jobnames from transparent physical sequential compression and expansion.

EXCLUDE-MOD-PST

Excludes transparent physical sequential compression and expansion for subroutine modules called from a processing program.

EXCLUDE-PGM-PST

Excludes transparent physical sequential compression and expansion for programs invoked by an EXEC statement within a job or proc step.

Records

A record entry follows a table statement, and must begin in column 1.

Comment

A comment statement is the character "*" in the first column, followed by a blank.

The following code shows a sample of the default CA Compress Exclude File:

```
EDIT       .CCVBJCL(EXCLUDE) - 01.00        Columns 00001 00072
Command ===>                                              Scroll ===> CSR
 ****** ***************************** Top of Data *******************
000010 * MAINTENANCE
000020 * 07/07/08 PAA 001783 .... UPDATED PRODUCT NAME
000030 *
000100 * THIS TABLE IS A LIST OF JOBS, PROGRAMS AND MODULES AND THE
000200 * OCCASIONS UNDER WHICH THEY SHOULD BE EXCLUDED FROM THE
000300 * COMPRESSION/EXPANSION OF CA COMPRESS
000400 *
000410 * PROGRAM NAMES MUST START IN COLUMN 1!
000420 *
000430 *EXCLUDE-JOB
000440 JOBNAME
000450 *EXCLUDE-PGM
000460 EMCSNAP          EMC TIMEFINDER
000461 SIBBATCH         IBM SNAPSHOT
000462 GVRESTOR         FAVER
000470 ADRDSSU          DFDSS
000480 ADSAR008         CA DISK AUTORESTORE
000490 ADSMI002         CA DISK BATCH
000500 *EXCLUDE-MOD
000501 EMCSNAP          EMC TIMEFINDER
000510 SIBBATCH         IBM SNAPSHOT
000600 GVRESTOR
000700 ADRDSSU
000800 ADSST100
000900 ADSAR008
001000 ADSMI002
001100 *EXCLUDE-JOB-PST
```

Expiration Date of 86060

Another way to use the Exclusion Feature is by specifying the EXPDT parameter (or the EXPDT subparameter of LABEL) on the DD statement with a value of 86060. When EXPDT=86060 is specified, CA Compress excludes the data set from compression or expansion because CA Compress is not invoked. An example is shown below:

```
// DDNAME DD DSN=compressed.data set.name,DISP=SHR,
// LABEL=EXPDT=86060
```

Exclusion by DDNAME @ZSM@XCL

Because EXPDT=86060 can conflict with your tape management system, or because EXPDT=86060 can be inconvenient for other reasons, CA Compress now supports exclusion by user-specified ddname, in one of two formats.

To invoke exclusion for all compressed data sets in the step, specify:

```
//@ZSM@XCL DD DUMMY
```

To specify exclusion of specific ddnames, specify them in the nodes of the DSN as follows. You can exclude as many ddnames as you can fit in the 44-byte DSN:

```
//@ZSM@XCL DD UNIT=SYSALLDA,SPACE=(TRK,0),DSN=hlq.ddn1.ddn2
```

For example, to exclude ddnames SYSUT1 and SYSUT2, where MYTSOID is a valid high level qualifier:

```
//@ZSM@XCL DD UNIT=SYSALLDA,SPACE=(TRK,0),
//      DSN=MYTSOID.SYSUT1.SYSUT2
```

Invoking Exclusion in Assembler Macros

It may be necessary to force exclusion regardless of user JCL or because you need the compressed data set attributes or the compressed data. You can disable CA Compress processing at OPEN, CLOSE, OBTAIN, RDJFCB, or catalog management by loading X'DEDFADE1' into Register 15 before issuing the macro.

It is your responsibility to use this facility consistently. Be careful to exclude both OPEN and CLOSE or neither, and be sure to treat RDJFCB and OPEN TYPE=J consistently. Failure to observe this restriction leads to abends and other unpredictable results.

Chapter 9: Safeguards

Compressed data should not be accessed by application programs when CA Compress is not active, because user programs will experience abends or other problems due to unexpected data.

This exposure is not great for sequential data sets, principally because update in place is not supported, and so access for update while CA Compress is not up simply results in a valid uncompressed data set. Support for sequential data sets using the SUBSYS parameter offers no protection, but customers seldom experience this problem, especially if they use FDTs created by the IUI, which enable CA Compress to distinguish between compressed and uncompressed records if both are present.

For VSAM, however, update in place is commonplace. In contrast to the physical sequential case, keys exist and may not be in the same physical location on the record in compressed and uncompressed records. For these reasons, CA Compress includes the Safeguards facility to protect non-SMS VSAM data sets defined under the CA Compress Transparency.

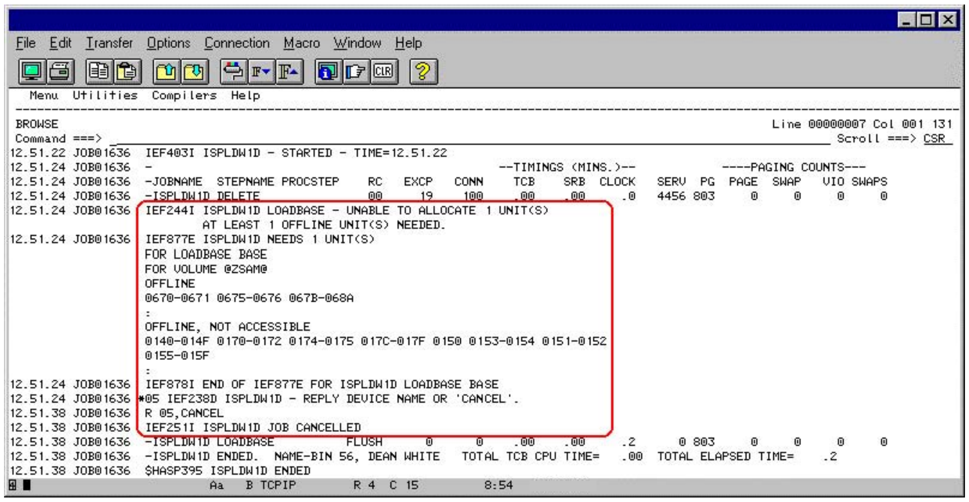
CA Compress Safeguards protect compressed VSAM data sets from inadvertent access under two conditions:

- When the CA Compress subsystem is not active.
- When the CA Compress subsystem is being abnormally terminated or shut down by request. If any compressed data sets are being accessed at shutdown time of the CA Compress subsystem, a warning message is issued.

This section contains the following topics:

How Safeguards Protect Data

Safeguards tell the started task to add a candidate volume to a CA Compress VSAM data set the first time that the data set is accessed when CA Compress is running. If it is a non-SMS data set, the started task adds the candidate volume if it is not present. This candidate volume prevents the allocation of the compressed data set when the CA Compress subsystem is not active. When CA Compress is active, it monitors all VSAM Catalog Management requests (SVC 26 calls) and removes the candidate volume so the job can allocate, open, and access the data set. Because this is not done if the started task is not running, allocation fails at data set allocation time with the following messages to the operator:


File Edit Transfer Options Connection Macro Window Help
Menu Utilities Compilers Help

```
BROWSE                                                                                    Line 00000007 Col 001 131
Command ===>                                                                                          Scroll ===> CSR
12.51.22 JOB01636  IEF403I ISPLDW1D - STARTED - TIME=12.51.22
12.51.24 JOB01636  -                                                  --TIMINGS (MINS.)--            ----PAGING COUNTS---
12.51.24 JOB01636  -JOBNAME  STEPNAME PROCSTEP    RC   EXCP    CONN    TCB    SRB  CLOCK   SERV  PG  PAGE  SWAP   VIO SWAPS
12.51.24 JOB01636  -ISPLDW1D DELETE               00     19     100    .00    .00     .0   4456 803     0     0     0     0
12.51.24 JOB01636  IEF244I ISPLDW1D LOADBASE - UNABLE TO ALLOCATE 1 UNIT(S)
                           AT LEAST 1 OFFLINE UNIT(S) NEEDED.
12.51.24 JOB01636  IEF877E ISPLDW1D NEEDS 1 UNIT(S)
                   FOR LOADBASE BASE
                   FOR VOLUME @ZSAM@
                   OFFLINE
                   0670-0671 0675-0676 067B-068A
                   :
                   OFFLINE, NOT ACCESSIBLE
                   0140-014F 0170-0172 0174-0175 017C-017F 0150 0153-0154 0151-0152
                   0155-015F
                   :
12.51.24 JOB01636  IEF878I END OF IEF877E FOR ISPLDW1D LOADBASE BASE
12.51.24 JOB01636 *05 IEF238D ISPLDW1D - REPLY DEVICE NAME OR 'CANCEL'.
12.51.38 JOB01636  R 05,CANCEL
12.51.38 JOB01636  IEF251I ISPLDW1D JOB CANCELLED
12.51.38 JOB01636  -ISPLDW1D LOADBASE           FLUSH      0       0    .00    .00     .2      0 803     0     0     0     0
12.51.38 JOB01636  -ISPLDW1D ENDED.  NAME-BIN 56, DEAN WHITE   TOTAL TCB CPU TIME=   .00  TOTAL ELAPSED TIME=    .2
12.51.38 JOB01636  $HASP395 ISPLDW1D ENDED
                              Aa    B TCPIP        R 4   C 15        8:54
```

The message states that the CA Compress volume (volume serial "@ZSAM@") is not available to the system. The operator must respond to this message by canceling the job and rerunning it after the CA Compress subsystem is activated.

Safeguards Detailed Description

To add safeguards to non-SMS VSAM data sets, CA Compress adds a special candidate volume to the catalog record for the data component of the data set. This volume has a serial number of "@ZSAM@". This serial number in the catalog is not apparent to users as long as the CA Compress subsystem is active. For example, an IDCAMS LISTCAT of the cluster does NOT show this volume in the listing when CA Compress is active. Nor do allocations of multiple volumes attempt to use this volume when CA Compress is active.

The implications of CA Compress adding Safeguards in this manner are as follows:

- After a data set is defined to CA Compress, the Safeguards are added to the data set the next time that it is accessed. The Safeguards are added without user involvement. When Safeguards are added, CA Compress writes a message to the operator, and it writes a message to SYSPRINT unless the started task specifies NOSGPRT to PGM=ZSUR through the PARM parameter.
- After the Safeguards are added, the data set can only be accessed when CA Compress is active on the CPU that is attempting the access. Special care must be taken when sharing an ICF catalog that contains CA Compress data sets across CPUs. CA Compress must be active on the CPU attempting access to the data set or the access fails.

Safeguards Utility

The Safeguards Utility is designed to provide a simple, easy way of adding or removing the CA Compress Safeguards. The Utility allows you to specify the CA Compress Control File that contains the names of the VSAM data sets under CA Compress's control and provides control statements that contain either a specific data set name or a pattern data set name to be used. The utility is useful when converting from an earlier release because it adds whatever Safeguards are missing from a data set. It can also be used to remove the Safeguards if it is necessary to run an earlier version of CA Compress that does not support them.

When the Safeguards Utility is invoked, it reads the SYSIN data set. The input can be either a pattern name or a specific data set name. If a specific data set is named, the Utility attempts to add or remove the Safeguards information from that data set without regard for its existence within the CA Compress Control File allocated to the VSAMFILE DD statement. If a pattern is named, it is stored for later processing. When the end of input on SYSIN occurs, the utility reads the VSAMFILE data set if there were any pattern data set names, trying to match each name from VSAMFILE with any of the patterns. If the match is successful, the Utility attempts to add or remove the Safeguards.

- The JCL for the Safeguards Utility is in YOUR.CAI.CCVBJCL(SGUJCL).
- The PARM can be either ON to add Safeguards to the selected data sets or OFF to remove them from the selected data sets.
- STEPLIB defines the library that contains the CA Compress load modules.
- VSAMFILE defines the CA Compress Control File that contains the names of the data sets you wish to process.

  The control statements are fixed format, with the data set or pattern name starting in column 1. A pattern name contains one or more special characters ("/", "?", "*" or "!"). For more information about these characters, see the CA Compress *User Guide*. A discrete data set name does not contain any special characters. To process all of the data sets in the VSAMFILE data set, use a pattern of / in column 1.

Chapter 10: Physical Sequential Transparency

The Physical Sequential Transparency (PST) supports compression and expansion of Physical Sequential data sets without application program or JCL changes. As in the case of VSAM data sets, the user defines selected PS data sets or patterns in the CA Compress Control File using the Interactive User Interface or Control File Maintenance Utility. CA Compress automatically intercepts activity against appropriate data sets in order to write compressed data to the compressed data sets and return uncompressed data from them to application programs.

This section contains the following topics:

Full Transparency to Application Programs

Support for application programs is fully transparent to the user. No special exits, parameters or interfaces are required for SORT and other utilities. Unlike the SUBSYS implementation, PST does not require you to code DCB parameters in the JCL unless the application requires it without CA Compress. GDGs, tape processing, and concatenation of PST and uncompressed data sets are fully supported as long as the processing is sequential using QSAM or BSAM. CA Compress causes SORT, which ordinarily uses EXCP, to consider PST data sets to be SYSIN/SYSOUT or SUBSYS data sets and to process them using BSAM.

Full Interactive User Interface and Control File Maintenance Utility

The Interactive User Interface (IUI) analyzes and implements Physical Sequential data sets transparently through the Control File, as it does for VSAM. The IUI is able to recognize data sets compressed using SUBSYS and in most cases can return the compression algorithm to the user, skipping analysis and compression and implementing the data set as already compressed.

The Control File Maintenance Utility (CFMU) supports the PST by means of two new Control File record types, PSDSNAME and PSPATTERN, with a number of new keyword parameters on the ADD and ALTER control statements to supply DCB and other information needed to process sequential data sets.

Compatibility with Previous Releases and the SUBSYS JCL Parameter

Data sets compressed with the transparency are completely compatible with data sets previously compressed using the SUBSYS parameter - byte for byte the compressed format is identical. Data sets already compressed with the SUBSYS parameter can be defined as already compressed and can be read and written immediately with no change.

Control File records and cross memory services to support the PST are similar to but completely distinct from those which support the VSAM Transparency. Older releases of CA Compress running on different systems can even share the same Control File compatibly, except that the earlier release cannot recognize the PST data sets as compressed. Data sets defined to the PST can be uncompressed by the previous release using the SUBSYS parameter, which provides backward compatibility.

Implementation Considerations

Implementing a Physical Sequential data set commonly involves analyzing the data set, choosing a compression method and defining it to the Control File, just as in the case of the VSAM Transparency, with certain minor differences. You can run the CFMU yourself to define the data set, but the IUI analyzes the data set and can normally tell whether it is already compressed or not, and with which algorithm, which helps you to insure that the data set is defined correctly.

Deferred and Immediate Implementation

CA Compress cannot support update in place for Physical Sequential data sets, and it does not support adding compressed records to a data set containing uncompressed data. For these reasons, Scheduled Implementation is inappropriate for PST data sets, but immediate and deferred compression are still supported by specifying the date compression is to take effect (EFFDATE=yyddd), or ANYDATE to specify that the data set should be immediately treated as compressed.

DCB Attributes

VSAM KSDS and ESDS record formats are essentially the same, whether the user considers the data set fixed or variable length. However, in the case of non-VSAM data sets there is a sharp distinction among fixed, variable, and undefined record formats. Compression changes these attributes, but applications expect to handle the data in its original uncompressed format, and so the uncompressed attributes must be preserved in the Control File for CA Compress OPEN processing.

Four parameters enable you to specify DCB attributes for the data set using the IUI or the CFMU: DCBMODEL, RECFM, LRECL, and BLKSIZE. DCBMODEL specifies a cataloged data set from which to take DCB parameters, and the other three, if coded, supply or override individual values.

The DCBMODEL data set can be any PS or PO data set except a compressed data set not defined as PST, because for such a data set no uncompressed attributes are available. If a compressed non-PST data set is specified for DCBMODEL, CA Compress issues a diagnostic message and rejects the statement.

Automated Cleanup of Uncataloged Data Sets

Unlike VSAM data sets, Physical Sequential data sets are often defined by GDGs or similar schemes - instead of remaining constant, the data set name for a given application keeps changing as new generations are created and old ones fall away. This can easily lead to uncontrolled growth of the Control File, so the CFMU provides the ERASEUNCAT keyword to specify that a PS dsname entry should be automatically deleted from the Control File after it is uncataloged. To avoid performance problems and to prevent entries from being lost due to inadvertent deletion of the data set, the Control File is not purged until at least the day after the data set is uncataloged.

For your convenience, CA Compress tries to pick an intelligent default. If you define a PS pattern with GDG=ONLY, CA Compress realizes that only GDG data sets are permitted and selects ERASEUNCAT=YES. For GDG=NO or GDG=YES, which permit non-GDG data sets, CA Compress defaults to ERASEUNCAT=NO for safety. In many cases, however, such as IMS log data sets, you probably want to override this choice with ERASEUNCAT=YES.

Implementing Uncompressed Data Sets with the IUI

While the IUI is building the Work List, it determines whether each PS data set is already compressed. If it is uncompressed, you implement it much like a VSAM data set. You can explicitly analyze it or you can pick a compression method and go directly to implementation.

If the IUI is running in Scheduled mode, it implements the data set online by adding a Control File entry with an effective date of today with the DCB attributes of the uncompressed data set. The data set is compressed the next time it is loaded.

If the IUI is running in Choice mode, you can specify all the values supported by the CFMU, including GDG, ERASEUNCAT, and NON-COMP. If you specify a nonzero effective date, the IUI implements the data set online with the values you specify, and the data set is compressed the next time it is loaded. If you specify ANYDATE, the IUI generates JCL to reload and compress the data set immediately, in the same way that it supports VSAM immediate compression.

Implementing Compressed Data Sets with the IUI

If the data set is already compressed, the IUI marks it COMPRESD on the Work List and analysis is not permitted. If the IUI is able to determine the FDT name or compression algorithm, implementation can be done only with that name. If the IUI cannot determine the FDT, you can enter the correct FDT.

Remember that if a data set is already compressed, you cannot really select a compression method. The compression selection has already been made, so you can only tell CA Compress what it is if the IUI is unable to determine it. To select a different Compression method, you must uncompress and reimplement the data set.

Limitations and Restrictions

The following are limitations and restrictions of Physical Sequential Transparency:

- Only Sequential Access Using QSAM or BSAM
- Concatenation Restrictions
- Limited DCB Exit List Support
- Relatively High Overhead for Sequential Processing

Only Sequential Access Using QSAM or BSAM

PST data sets can only be processed sequentially, using QSAM or BSAM, and update in place is not allowed. However, even though the IBM subsystem interface is used, BSAM NOTE and POINT macros are supported, so ISPF BROWSE and EDIT, ISPF option 3.3 and similar functions are supported.

Access methods such as BDAM and EXCP are not supported for PST data sets. By adding SORT and other common utilities which use EXCP to the EXCLUDE-SORT exclusion tables, you can tell CA Compress to force these programs to think a PST data set is a subsystem data set and to use BSAM. For this to work, however, the program must be able to handle subsystem data sets. Any program which works correctly with the SUBSYS JCL parameter should support PST data sets.

Concatenation Restrictions

PST data sets cannot be concatenated with SYSIN (//ddname DD *) or other subsystem data sets. In this case, message ZSUR407I is issued and the entire concatenation is processed without compression or expansion, which will probably lead to errors. Likewise, if you specify LABEL=EXPDT=86060 on any data set in a concatenation, the exclusion is applied to the entire concatenation.

Limited DCB Exit List Support

RDJFCB exits x'07' and x'13' are fully supported. User label exits are supported at OPEN and CLOSE, but not at EODAD or EOV. Because CA Compress must control the DCB attributes, it replaces the user's DCB Open exit. It does not support EOV exits, user totaling exits, or other exits which might be affected unpredictably by Compress.

Relatively High Overhead for Sequential Processing

Compression is relatively inexpensive and easy to justify with random access, which is typical in the case of VSAM, because savings due to compression occur for each record, whether you ever read it or not, and overhead is exacted only for the relatively few records actually processed. However, sequential processing incurs overhead for every record because in most cases all are read whenever the data set is processed.

Moreover, programs like SORT, which normally can optimize I/O, are forced to use the subsystem interface so that CA Compress can compress or expand each logical record, and this adds substantially to I/O overhead.

For these reasons, not all sequential data sets should be compressed. Good examples are multivolume tapes written once and seldom read, or DASD data sets created at night and seldom accessed during the day.

Chapter 11: User Exits

To enhance its power and flexibility, CA Compress enables you to direct its processing at certain points by means of user exits, as described below.

CA Compress offers the following user exits:

- Transparency User Exit
- Control File Maintenance Utility Security Interface
- Test Compression Facility Pre-Processing, Post-Processing, and Security Exits

These user exits are discussed in detail below.

This section contains the following topics:

Transparency User Exit

The Transparency User Exit enables you to gain control during compression and expansion in order to avoid or recover from errors. You can receive control from CA Compress for each record before and after compression and before and after expansion. Through the CA Compress/2 parameter list, described in the chapter *CA Compress/2*, you have access to the record in both its compressed and uncompressed state, as well as to its length and other information. By setting the return code at PRECMP and PREXPD, you can prevent compression and expansion where appropriate. After taking corrective action, you can recover at POSTCMP and POSTXPD from compression and expansion errors resulting in I/O errors and messages such as SHR014I, SHR015I, and SHR010I.

The exit is powerful, and you should have a good understanding of the material in the chapter *CA Compress/2*, and of the parameters you receive in the exit before you manipulate them in any way. Errors in the user exit may lead to I/O errors or even data loss, so always test any changes carefully before enabling the exit for any production data set. You receive addressability to the ddname, the data set name, and the TCB, which gives you ready access to the jobname, so you can use these values to control your processing. You can use RACF or an equivalent product to control access to the module name, ZUXITMOD.

Enabling the User Exit

The user exit is enabled by the UEXIT parameter of the Control File Maintenance Utility ADD and ALTER statements. UEXIT is coded just like the EXCLUDE parameter, whose syntax is found in the descriptions of the ADD and ALTER statements in the chapter *Control File Maintenance*.

If UEXIT is not specified, or if the exit is unavailable when CA Compress tries to load it, compression and expansion are performed without it. If the exit is specified but is unavailable, CA Compress issues message ZSUR296I, described below.

The expansion phases can only receive control if the preceding GET was successful, because CA Compress attempts expansion only when the record has been successfully read. Because compression is performed before writing the record, the PUT has not yet been done when the compression phases receive control, so errors in the exit may cause I/O errors when CA Compress tries to write the record.

Using the User Exit

The Transparency User Exit must be called ZUXITMOD and reside in STEPLIB, JOBLIB, or the linklist. If you place the exit in linklist, it can be accessed to process any data set for which UEXIT has been specified, perhaps in cases you do not intend. For this reason, avoid the linklist and use STEPLIB or JOBLIB until you confirm that the exit works as you intend in every case

The exit must be reentrant, because I/O can take place concurrently on several data sets and RPLs. It can have any RMODE but is always given control in AMODE 31 to insure that it can gain access to all the parameters it receives from CA Compress.

CA Compress calls the User Exit using standard linkage conventions. The Registers and their descriptions are given in the following section.

Register 1

Points to the parameter list

Register 13

Points to a 72 byte save area. It is immediately followed by another, which you can use instead of GETMAINing one yourself

Register 14

Contains the return address and caller's AMODE

Register 15

Contains the address of the entry point receiving control

CA Compress saves and restores Registers 0 thru 9 and ignores Register 15, so the exit can freely use them as work registers without preserving their contents. The exit must restore Registers 10 through 14 when it returns to the caller. CA Compress ignores Register 15 on return and takes the return code from the first word of the parameter list, as documented in the following section.

Coding the User Exit

The first 4 words of the user exit define the routines that are to get control at each of the 4 phases of the user exit - PRECMP, POSTCMP, PREXPD, and POSTXPD. If any of these addresses is zero, that phase does not get control. The routines follow the 4 word prolog.

To simplify coding, CA Compress supplies a prolog macro, ZXITPLOG, in the installation library. ZXITPLOG builds the following prolog logic:

1. The 4 word entry point list, addressing the routines you specify for each phase, or zero for those you do not specify.
2. Register equates, with comments.
3. A dsect and a USING statement for the parameter list.
4. Dsects for the RPL and ACB to permit you to address their fields symbolically.
5. The module name, date, time, and user-specified identification field.

The Parameter List

The parameter list passed to all 4 phases of the user is addressed by Register 1 and has the following format:

Name	Offset	Description
ZUXRC	0	Zero on entry to PRECMP and PREXPD. CA Compress/2 Return Code on entry to POSTCMP and POSTXPD. Set this field to set the return code, as documented in Return Codes in this chapter.
ZUXCLIST	4	Address of CA Compress/2 Parameter List as described in the chapter *CA Compress/2*, except that the RDW is never included unless the RDL begins with V2-4.
ZUXDDNAM	8	Address of User ddname.
ZUXDSN	12	Address of DSNAME.
ZUXURPL	16	Address of User RPL - refers to uncompressed record
ZUXCRPL	20	Address of Compress RPL - refers to compressed record
ZUXTCB	24	Address of TCB

Name	Offset	Description
ZUXDEBUG	28	Address of Debug Byte. If the byte is nonzero, PST tracing is active, and if zero, it is not. Use this byte to determine whether you want to issue diagnostic messages, and you can change this byte in order to control tracing in module ZSURSHRK for this data set.
ZUXSPA	32	User Exit Scratch Pad Area. This area is 96 bytes long, and the exit can use it for any purpose. Because CA Compress does not use this area after I/O is complete, its contents are valid from one phase to the next.

Return Codes

All phases receive the return code in ZUXRC, the first word of the parameter list, and they can change it there as appropriate. The incoming and outgoing return codes and their meanings for each phase are as follows:

PRECMP

On entry, ZUXRC is always zero. The exit can determine that the record cannot be successfully compressed, perhaps because it is shorter than the noncompressible area defined by the RDL. To avoid the failure in compression in this case and message SHR015I, you can set ZUXRC to -1 (x'FFFFFFFF') to stop compression and write the uncompressed record to the compressed output data set. If you stop compression at PRECMP, POSTCMP is never entered, because compression is never called.

POSTCMP

On entry, ZUXRC contains the return code from compression, either zero or a positive return code indicating that compression failed. You can recover from the error, after making any needed corrections, but any failing message from CA Compress/2, such as SHR015I, has already been issued. Set ZUXRC to zero in order to write the compressed record or to -1 (x'FFFFFFFF') to copy the uncompressed record without compression. If ZUXRC is zero and you set a positive return code, CA Compress forces an I/O error, just as if compression had failed.

PREXPD

On entry, ZUXRC is always zero. The exit may determine that the record is already uncompressed and want to avoid expanding it again and causing errors. To stop expansion and write the compressed record to the output data set, set ZUXRC to -1 (x'FFFFFFFF'). If you stop expansion at PREXPD, POSTXPD is never entered, because expansion is never called.

POSTXPD

On entry, ZUXRC contains the return code from expansion, either zero or a positive return code indicating that expansion failed. You can recover from the error, after making any needed corrections, but any failing message from CA Compress/2, such as SHR010I, has already been issued. Set ZUXRC to zero in order to write the uncompressed record or to -1 (x'FFFFFFFF') to copy the compressed record without expansion. If ZUXRC is zero and you set a positive return code, CA Compress forces an I/O error, just as if expansion had failed.

Control File Maintenance Utility Security Interface

The Control File Maintenance Utility interfaces with your installation's security system: RACF, Top Secret, or ACF2. You need specific types of authority in order to create or access compression control records in the Control File. You can also write a user exit to modify that processing.

How the User Security Exit Works

The User Security Exit is invoked for each access the Utility makes on any compression profile contained in the Control File. The exit can allow, disallow, or make no access determination. In addition, the exit can change the data set name and the requested access level, to be used by the CA Compress Security Interface for access determination. If the exit makes no access determination, or if the exit does not exist, the installation's access control facility is invoked through the System Authorization Facility (SAF) interface. If your installation's access control facility does not support SAF, then the exit must make a determination or the access is allowed.

The access already defined within your access control facility automatically determines a user's level of access. If you are attempting to ADD or DELETE a Control File entry, you need authority that corresponds to RACF's ALTER access. To issue an ALTER for a Control File entry, you need UPDATE access. To report on an entry, you need at least READ access. These levels of access can be modified by the user exit.

Using the Security Exit

The Security Exit must be named ZSUSEC00 and reside in a load module library that is accessible by the Control File Maintenance Utility. The exit module may reside in a library in the LINKLIST or in the JOBLIB or STEPLIB concatenation. The exit may be link edited with any valid combination of AMODE and RMODE and should be at least serially reusable.

Two default exits are supplied. One allows all accesses. The other allows all accesses for the REPORT statement but invokes the SAF interface for ADD, DELETE and ALTER. The assembler source for these exits is contained in the YOUR.CAI.CCVBSAMP data set. The exit names are ZSUSEC01 (allow all accesses) and ZSUSEC02 (check SAF).

Linkage Conventions of the Exit

CA Compress employs standard MVS linkage conventions when invoking the Security Exit. At entry to the exit:

Register	Description
Register 1	contains the address of the parameter list
Register 13	contains the address of a 72 byte save area
Register 14	contains the return address and caller's AMODE
Register 15	contains the Exit's entry point address

All registers (except 15) must be saved at entry and restored when the exit returns to the Control File Maintenance Utility. Register 15 must contain a return code indicating the security exit's determination.

The exit is called in problem state, user protect key, and the AMODE defined by the link edit attributes.

Return Codes

The following return codes in register 15 cause CA Compress to take the indicated action:

0

The exit has allowed the access. CA Compress does not invoke SAF for access determination. The access is allowed.

4

The exit has made no determination about access. CA Compress invokes SAF for access determination. Access is determined by SAF.

8

The exit has disallowed access. CA Compress does not invoke SAF for access determination. The access is disallowed.

The Parameter List

The Parameter list passed to the security exit follows standard MVS conventions. Register 1 contains the address of a list of addresses to parameter values:

x'00'

The count of the parameters that follow.

x'04'

The access level requested. The field is 2 bytes in length and contains a binary value: a 1 for READ access, a 2 for UPDATE access, and a 3 for ALTER access. This field can be changed by the exit to reflect a different level.

x'08'

The statement for which the processing is requested. The field is 8 bytes long.

x'0C'

The entry type of the following parameter. The field is 8 bytes long. This can be either DATA SET or PATTERN.

x'10'

The entry name. This field is 44 bytes long and contains the data set name or the pattern name to be accessed.

x'14'

The JOBNAME. This field is 8 bytes long.

Test Compression Facility User Exit

TCF supports the use of optional user exits at various phases in its processing. These are provided so that each installation can tailor the use of TCF beyond what can be accomplished based on parameter specifications.

Assembler language conventions are followed in passing parameters to the exit modules. Register 1 points to a list of addresses, each of which points to a specific parameter as defined for the exit. In some cases, the exit module returns a half word code through the designated parameter to indicate to TCF what further actions are to be taken.

PREEXIT Pre-Processing Exit

The PREEXIT gets control after TCF obtains the next sequential data set name from catalog management, but before it obtains the detailed catalog information and format 1 DSCB. Its function is to decide whether or not to allow TCF to compress the data set. PREEXIT gets control before any selection/exclusion testing, which would be performed based on SELECT or EXCLUDE statements in the job stream, so SELECT and EXCLUDE statements can override the PREEXIT's decision to process the data set. The parameter list that is passed to the exit is shown in the following table.

Type	**Size**	**Description**		
OUTPUT	Halfword	Result Code	0	Allow the data set to be processed.
			8	Bypass the data set.
INPUT	CL44	Data set Name-The name of the data set currently being considered for inclusion in TCF processing.		
INPUT	CL44	Catalog Name-Where the data set is cataloged.		

POSTEXIT Post-Processing Exit

The POSTEXIT gets control after TCF test compresses the data set and computes statistics.

The parameter list that is passed to the exit is shown in the following table.

Note: The statistical fields contain binary zeroes when TCF is run in simulation mode.

Type	**Size**	**Description**
INPUT	CL44	Data set Name-The name of the data set currently being processed.
INPUT	Halfword	Volume Count-The total number of volumes to which the data set is cataloged. This is also the number of volumes that is presented below in the Volume List parameter.

Type	Size	Description	
INPUT	nCL6	Volume List-A list of the volumes to which the data set is cataloged. The number of volumes appearing in this list can be determined by examining the Volume Count parameter above.	
INPUT	CL3	Data set Organization-This is a literal value indicating the data set organization. The valid values are 'VS' for VSAM data sets and 'PS' for physical sequential data sets.	
INPUT	XL80	Statistical Information - The statistical information that TCF generated for the data set is passed as a block of information. It begins on a fullword boundary and contains the following information:	
		FW	Total number of data records in the data set.
		FW	Total number of compressed records.
		FW	Total number of data bytes in the data set.
		FW	Total number of data bytes in the compression sample.
		FW	Total number of bytes after Super Express compression sample.
		FW	Total number of bytes after Huffman compression sample.
		FW	Estimated total number of bytes saved using Huffman.
		FW	Estimated total number of bytes saved using Huffman.
		FW	Estimated total number of bytes saved using Super Express.
		FW	Estimated number of tracks saved using Huffman.
		FW	Estimated number of tracks saved using Super Express.
		FW	Minimum data record size encountered.
		FW	Maximum data record size encountered.
		FW	Average data record size encountered.
		FW	Minimum record size after Super Express compression.
		FW	Maximum record size after Super Express compression.

Type	Size	Description	
		FW	Average record size after Super Express compression.
		FW	Average record size after Super Express compression.
		FW	Maximum record size after Huffman compression.
		FW	Average record size after Huffman compression.
		HW	Percent of compression using Super Express (that is, 534 = 53.4%).
		HW	Percent of compression using Huffman (that is, 534 = 53.4%).

Security Interface and Exit

The TCF comes with interface routines to the ACF2, RACF and Top Secret security systems. If you use one of these, you can invoke the appropriate interface by coding the SECURITY parameter on the SET statement. TCF calls the security system you specified on the SET statement before obtaining the catalog information for the current data set. If you do not have read authority for the data set, message DCA0040 is issued and the data set bypassed.

Important Note for ACF2 Users: TCF's ACF2 interface dynamically attempts to obtain access to ACF2's CVT. If this cannot be accomplished, message DCA0071 is issued indicating the CVT could not be dynamically obtained. When this error occurs, you must link edit the ACF2 CVT with the TCF ACF2 security interface module (GDAXP014) in order for TCF to gain access to ACF2's *user call* interface. Without this special link edit, TCF issues message DCA0070 and TCF's security checking is deactivated for the duration of the job. Below is some sample JCL to perform this special link edit:

```
//LKED    EXEC PGM=HEWL,PARM=(LIST,MAP),REGION=320K
//SYSPRINT DD  SYSOUT=A
//SYSUT1   DD  UNIT=SYSDA,SPACE=(1024,(50,20))
 //SYSLIB   DD  DISP=SHR,DSN=TCF.Rnn.LOADLIB
//ACF2     DD  DISP=SHR,DSN=ACF2.LOADLIB
//SYSLMOD  DD  DISP=SHR,DSN=DCA.Rnn.LOADLIB
//SYSLIN   DD  *
SETCODE AC(0)
 INCLUDE ACF2($ACFGCVT)
 INCLUDE SYSLIB (GDAXP014)
 ENTRY GDAXP014
NAME GDAXP014
```

Note: The JCL above shows the module name for the ACF2 CVT as $ACFGCVT. Your release of ACF2 can use ACF$GCVT instead. Attempt the link edit with the JCL shown. If you get an unresolved reference for $ACFGCVT, resubmit the job with the alternate module name.

Glossary

algorithm

A finite set of well-defined rules for the solution of a problem in a finite number of steps, for example, a specific set of steps used to compress or expand a record.

analysis file

The data set used by the CA Compress Interactive User Interface to record the results of compression analysis and implementation performed by the IUI.

BDA

Byte Distribution Analysis. The process of analyzing the distribution of characters in a data set in order to construct the Huffman tables used to compress and expand the data set.

buffer

Storage allocated to temporarily hold input or output data.

CA Compress started task

Running in its own address space, the started task supports the SVC intercepts, cross-memory services, and other facilities required to support compression transparently for application programs.

CA Compress/2 subroutines

Compression and expansion subroutines, used by CA Compress, which can also be called by users. These subroutines are especially useful for supporting compression in products with data that CA Compress cannot support transparently.

CA Compress/2 utilities

Utilities that can be used to perform compression and expansion when the CA Compress subsystem is unavailable.

CFMU

Control File Maintenance Utility. A batch utility provided with CA Compress for maintaining the Control File.

checkbyte

An extra byte calculated and added to the compressed data. If it does not exactly match the expected result at expansion, the compressed data has been damaged, perhaps through overlay or truncation when written.

compression algorithm

A finite set of rules used to collect and assign values to the characters found in a string of data so that the data takes less space.

compression overhead

The processing required to compress data as it is transferred from the application to the data set.

control file

The data set used by CA Compress to record the data sets controlled by CA Compress and how CA Compress should handle them.

discrete data set

A single data set, as opposed to a pattern defining a number of data sets.

dynamic allocation

1. Assignment of system resources to a program at the time the program is executed rather than at the time it is loaded into main storage.
2. The IBM facility provided through SVC 99 to accomplish this purpose.

exclusion feature

A facility enabling the user to supply tables of jobs and programs for which compression and expansion should not take place in order to prevent unnecessary compression and expansion or double compression.

expansion overhead

The processing required to expand data as it is transferred from the data set to the application.

express

The CA Compress string compression algorithm that was replaced by the Super Express algorithm.

FDT

File Descriptor Table. A load module that also exists as records on the Analysis and Control Files. It contains information required to compress and expand the data set, including its Huffman compression and expansion tables.

FDTLIB

The load library in which the Interactive User Interface stores FDTs, and from which it and the CFMU copy FDTs to the Control File.

file analysis

The process by which CA Compress presents the various compression choices and the compression percentages possible.

huffman

A compression algorithm that uses tables to replace each character by a variable-length bit string. The characters expected to occur most often are assigned the shortest bit lengths.

ICB	Integrity Check Block. A 3-byte field preceding the compressed data by which CA Compress recognizes the algorithm used to compress the data.
implementation	Placing a data set under CA Compress control.
interactive user interface (IUI)	The ISPF interface of CA Compress.
linklist	The list of load module libraries chosen by the installation to be searched by default after any STEPLIB or JOBLIB libraries.
LSR	Local Shared Resources. The IBM facility for permitting buffers to be shared by VSAM data sets in a job step. Because I/O to DASD is greatly reduced for random processing in most cases, performance is much improved even for a single data set.
PASSTHRU	A facility to enable CA Compress processing without actually performing compression or expansion.
pattern	A name with variable characters used to implement a particular compression algorithm for all data sets matching the pattern.
pattern analysis	The process by which CA Compress will search for data sets to analyze based on name.
pattern matching	The process of implementing a data set when it matches a pattern in the Control File, rather than explicitly by name.
RDL	Record Definition Language. The language supplied with CA Compress to define compression on each record. RDL is usually generated by default, but can be modified by the user.
RDW	Record Descriptor Word. The first 4 bytes of a variable length non-VSAM record or segment, which defines its length.
safeguards	A CA Compress facility for preventing inadvertent access to compressed VSAM data sets when the CA Compress subsystem is not active.

SAM-SI	Sequential Access Method-Subsystem Interface. The IBM access method used to support the SUBSYS JCL keyword. It is also invoked by Physical Sequential Transparency. Its limitations require certain special considerations when implementing compression using the SUBSYS JCL parameter.
scheduled	A method of implementing compression at a specified future event either when the VSAM data set is next loaded or when it is opened for output.
SDB	System Determined Blocksize, in which the user codes BLKSIZE=0 to cause the system to select an optimal block size for the device.
SHRVL	A CA Compress compression algorithm that achieves high compression for relatively high CPU overhead on certain types of data.
space release	The facility for freeing unused space from compressed VSAM data sets.
standard tables	Six compression tables distributed with CA Compress.
SUBSYS	An IBM JCL DD parameter for non-VSAM data sets that invokes a subsystem to process the ddname on which it is coded.
subsystem	A facility running under MVS that performs a certain function. JES2 and JES3, for example, handle job entry and throughput, and SMS does storage management.
super express	The CA Compress string compression algorithm. It compresses repetitive characters without using tables.
TCF	Test Compression Facility.
VPE	VSAM Performance Enhancement.
VPE rules	Rules supplied to VPE to direct its optimization of data sets.
VPE rules table	The table of rules built by VPE in extended CSA and addressed through the entry for VPE in the subsystem control table (SSCT). VPE does not run as a true subsystem or started task, but in this way VPE can recognize rules built before it is activated.

Index