CA Compress™ Data
Compression for IMS

User Guide
Version 18.0.00
This Documentation, which includes embedded help systems and electronically distributed materials, (hereinafter referred to as the “Documentation”) is for your informational purposes only and is subject to change or withdrawal by CA at any time. This Documentation is proprietary information of CA and may not be copied, transferred, reproduced, disclosed, modified or duplicated, in whole or in part, without the prior written consent of CA.

If you are a licensed user of the software product(s) addressed in the Documentation, you may print or otherwise make available a reasonable number of copies of the Documentation for internal use by you and your employees in connection with that software, provided that all CA copyright notices and legends are affixed to each reproduced copy.

The right to print or otherwise make available copies of the Documentation is limited to the period during which the applicable license for such software remains in full force and effect. Should the license terminate for any reason, it is your responsibility to certify in writing to CA that all copies and partial copies of the Documentation have been returned to CA or destroyed.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CA PROVIDES THIS DOCUMENTATION “AS IS” WITHOUT WARRANTY OF ANY KIND, INCLUDING WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NONINFRINGEMENT. IN NO EVENT WILL CA BE LIABLE TO YOU OR ANY THIRD PARTY FOR ANY LOSS OR DAMAGE, DIRECT OR INDIRECT, FROM THE USE OF THIS DOCUMENTATION, INCLUDING WITHOUT LIMITATION, LOST PROFITS, LOST INVESTMENT, BUSINESS INTERRUPTION, GOODWILL, OR LOST DATA, EVEN IF CA IS EXPRESSLY ADVISED IN ADVANCE OF THE POSSIBILITY OF SUCH LOSS OR DAMAGE.

The use of any software product referenced in the Documentation is governed by the applicable license agreement and such license agreement is not modified in any way by the terms of this notice.

The manufacturer of this Documentation is CA.

Provided with “Restricted Rights.” Use, duplication or disclosure by the United States Government is subject to the restrictions set forth in FAR Sections 12.212, 52.227-14, and 52.227-19(c)(1) - (2) and DFARS Section 252.227-7014(b)(3), as applicable, or their successors.

Copyright © 2014 CA. All rights reserved. All trademarks, trade names, service marks, and logos referenced herein belong to their respective companies.
CA Technologies Product References

This document references the following CA Technologies products:

- CA Compress™ Data Compression for IMS (CA Compress Data Compression for IMS)

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

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CA Compress Data Compression for IMS Overview</td>
</tr>
<tr>
<td>Express Overview</td>
</tr>
</tbody>
</table>

## Chapter 2: Database Compression

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>How Database Compression Works</td>
</tr>
<tr>
<td>File Description Table</td>
</tr>
<tr>
<td>Database Compression Considerations</td>
</tr>
<tr>
<td>JCL Requirements</td>
</tr>
<tr>
<td>Database Compression Controls</td>
</tr>
<tr>
<td>Share an FDT between Two or More Segments</td>
</tr>
<tr>
<td>Enter or Update the RDL for a Database Segment</td>
</tr>
<tr>
<td>Control the Size of a Compressed Segment</td>
</tr>
<tr>
<td>First-use Recommendation</td>
</tr>
<tr>
<td>How to Test an IMS Database Compression</td>
</tr>
<tr>
<td>View Results of Previous Compression Tests</td>
</tr>
<tr>
<td>Analyze the Database</td>
</tr>
<tr>
<td>View Results of Database Analysis</td>
</tr>
<tr>
<td>Specify Parameters for Data Sample Extraction</td>
</tr>
<tr>
<td>Generate JCL for Database Compression Test</td>
</tr>
<tr>
<td>Run Database Compression Test</td>
</tr>
<tr>
<td>How to Refine RDL for Segments</td>
</tr>
<tr>
<td>Compress an IMS Database with CA Compress Data Compression for IMS Controls</td>
</tr>
<tr>
<td>Delete an IMS Database From the Control File</td>
</tr>
<tr>
<td>How to Change a Database Controlled by CA Compress Data Compression for IMS</td>
</tr>
<tr>
<td>Refresh FDTs without Bringing IMS Down</td>
</tr>
<tr>
<td>Recreate a Lost or Damaged FDT</td>
</tr>
</tbody>
</table>

## Chapter 3: Database Compression without the Interactive Dialog

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>How to Compress a Database without the Interactive Dialog</td>
</tr>
<tr>
<td>User Constraints</td>
</tr>
<tr>
<td>Database Integrity</td>
</tr>
<tr>
<td>How to Define a Database to CA Compress Data Compression for IMS</td>
</tr>
<tr>
<td>How to Perform Database Compression</td>
</tr>
<tr>
<td>How to Compress a Database without the Interactive Dialog using Express</td>
</tr>
<tr>
<td>Add the COMPRTN Parameter by Editing Existing DBD Source Macros</td>
</tr>
</tbody>
</table>
Chapter 4: Reports

Common Product Reports ................................................................. 55
FDT Index Listing Report ................................................................. 55
Authorized User List Report .......................................................... 57
IMS Reports ..................................................................................... 58
Index List Report ............................................................................. 58
Stored Data Report ......................................................................... 59
Extracted Data Sample Report ......................................................... 70
Byte Distribution Analysis Report .................................................... 73
Test Compression Report ................................................................. 76

Appendix A: Record Definition Language ............................................. 97
RDL General Overview .................................................................... 97
Performance Considerations When Using RDL ................................. 98
How the RDL Operates ................................................................... 99
RDL Terminology ........................................................................... 99
Field ............................................................................................... 99
Field Type ..................................................................................... 100
Field Length .................................................................................. 100
RDL Syntax Rules ........................................................................... 101
RDL Specification Field Type Codes ............................................... 101
Field Types C1, C2, and C3—Character Data ................................... 102
Field Type GA—Garbage Data (Permanently Unused Fields) .......... 103
Field Type L—Insert Tally of Actual Length .................................... 104
Field Types MA and MB—Pattern Matching .................................... 104
Field Type N—Fields Exempted From Compression ......................... 106
Field Type PD—Packed Decimal Data ............................................. 107
Field Types S and X—Set of Expected Values ................................. 108
Field Type UN—Undefined Fields ................................................... 110
Field Types V, VP, and VZ—Calculating a Value for the Variable Symbol ................................................................. 111
Field Types ZL and ZR—Zoned Decimal Data .................................. 113
RDL Specification Field Length Descriptors .................................... 114
RDL Repetition Groups ................................................................. 116
RDL Condition Groups ................................................................. 117
Alternative Condition Groups ......................................................... 118
Default Condition Group ................................................................. 120
RDL Position Function ................................................................. 121
General Restrictions on RDL Use .................................................... 123
RDL Defaults .................................................................................. 124
RDL Default Specifications for Sequential Files ................................................................. 125
RDL Default Specifications for ISAM Files ........................................................................ 125
RDL Default Specifications for VSAM Files ...................................................................... 126

Glossary ................................................................................................................................. 127
Chapter 1: Introduction

This section contains the following topics:

CA Compress Data Compression for IMS Overview (see page 9)
Express Overview (see page 10)

CA Compress Data Compression for IMS Overview

CA Compress Data Compression for IMS transparently compresses selected segment information for an IMS database before storing it on DASD. When IMS reads the information back from DASD, it automatically decompresses the information, while the application programs and IMS utilities all function normally.

CA Compress Data Compression for IMS lets you analyze the space savings and the required overhead for existing databases before deciding whether to implement compression. The Record Definition Language (RDL) is used to specify the types of data fields and their layout within a segment to allow optimum compression for each data element.

CA Compress Data Compression for IMS interfaces contain unique features, but provide performance capabilities comparable to other available data compression products. The interfaces that use Huffman compression routines can consistently realize compression rates of 50 to 60 percent. Compression yields can be as high as 70 to 80 percent.

CA Compress Data Compression for IMS uses a check byte comparison to help ensure complete data integrity.

CA Compress Data Compression for IMS operates under control of the IBM z/OS operating system. CA Compress Data Compression for IMS is implemented as an IMS Segment Edit/Compression exit routine and is completely transparent to IMS application programs while providing:

- Compression for both fixed-length and variable-length segments
- Data compression only (key sequence fields remain uncompressed)
- Compression of one or more segment types of a physical database

Note: The Compress/2 subroutines (SHRINK, EXPAND, and CLOSE) and utilities (SHRINK and EXPAND) are currently supported. However, this support will be deprecated in a future release. For more information about how to use these subroutines and utilities, see the CA Compress Data Compression 5.5 documentation bookshelf.
Express Overview

The Express component provides data compression and expansion facilities for IMS users. The program operates under control of the IBM z/OS without application program modification. In a z/OS environment, Express can be used with the IMS Parallel DL/I mode of database control.

The Express component offers lower compression rates while limiting CPU overhead. Express uses a check byte comparison to help ensure complete data integrity.

Express is implemented as an IMS Edit/Compression User Exit and is transparent to IMS application programs while providing the following:

- Compression for both fixed-length and variable-length segments
- Data compression only; key fields (sequence fields) remain uncompressed
- Compression of one or more segment types of a physical database
- No JCL changes are required when using Express

You can implement Express using the facilities of the Interactive Dialog, an ISPF interface, or by a simple modification of the IMS DBD source macros.

Note: If your facility does not use ISPF version 2.0 or higher, see How to Compress a Database without the Interactive Dialog (see page 53).
Chapter 2: Database Compression

This section contains the following topics:

- **How Database Compression Works** (see page 11)
- **File Description Table** (see page 12)
- **Database Compression Considerations** (see page 12)
- **JCL Requirements** (see page 13)
- **Database Compression Controls** (see page 13)
- **First-use Recommendation** (see page 18)
- **How to Test an IMS Database Compression** (see page 19)
- **Compress an IMS Database with CA Compress Data Compression for IMS Controls** (see page 27)
- **Delete an IMS Database From the Control File** (see page 29)
- **How to Change a Database Controlled by CA Compress Data Compression for IMS** (see page 29)
- **Refresh FDTs without Bringing IMS Down** (see page 30)
- **Recreate a Lost or Damaged FDT** (see page 31)

**How Database Compression Works**

To compress a database using CA Compress Data Compression for IMS, follow these steps:

1. **Select a database for compression** (see page 12).
2. **Test database compression** (see page 19).
   
   The compression ratio depends on the data stored in the database and on the accuracy of RDL that describes the characteristics of the data. CA Compress Data Compression for IMS lets you test the compression on a data sample before you compress the database.

   Perform a series of test compressions and tailor RDL to achieve the best compression ratio.
3. **Create DBD for database compression** (see page 27).
   
   Based on the results of the test compressions, select segments to compress and create a compression DBD.
4. **Unload and reload the database** (see page 27).
   
   Unload the database using the old DBD and then reload the database using the newly created compression DBD.
File Description Table

A File Description Table (FDT) contains complete information that is needed to change the form of the file from uncompressed to compressed and conversely. FDT is a short 1-through 8-KB sequential data set. Each file processed must have an FDT. FDT stores information such as code tables, edited record definitions, or file characteristics.

After a file is compressed, the data contained in the file appears unintelligible because its coding structure is modified during file compression. The FDT contains the information required to return the file or individual records from the file to its original uncompressed form. Control of the FDT is equivalent to control of the file. You can use the FDT control as a part of the security system to limit access to compressed files.

You can convert FDT to a load module stored as a member of a load library PDS with the FDTLOADR Utility. Converting FDTS to load modules can reduce JCL coding in application programs.

More information:

FDTLOADR Utility (see page 47)

Database Compression Considerations

Databases are not automatically good candidates for compression. Consider the following when choosing databases for compression:

- Databases with random primary access are better candidates, because random access corresponds better with normal online operation.
- Consider carefully a database whose primary access is sequential. The run time of the program may increase to what could be considered an unacceptable level.
- A database is compressed on a segment-by-segment basis, not a whole. Segments containing fewer than 20 bytes of compressible area may cost more in compression and expansion overhead than the amount of DASD space would justify.
JCL Requirements

To access compressed records in an online environment or when executing batch programs that access compressed segment data, the CA Compress Data Compression for IMS load library (hlq.CIMTLOAD) requires APF-authorization. Include the hlq.CIMTLOAD load library and the FDT library in the STEPLIB concatenation.

When executing a utility where a valid IMS environment does not exist for databases that also use CA Compress Data Compression for IMS, specify the IMSUT.LOAD library first in the STEPLIB concatenation. This library contains the special version of the IMSHRINK program that enables compression support for this environment.

Note: For more information about enabling compression support for a non-IMS environment, see the Installation Guide.

Database Compression Controls

We recommend that you consider the following information before specifying compression controls:

■ Sharing an FDT between two or more segments (see page 13)
■ Using an RDL for a database segment (see page 16)
■ Controlling the size of a compressed segment (see page 17)

Share an FDT between Two or More Segments

You can use single FDT by multiple segments to conserve resources. In the shared FDT environment, the base segment uses its own RDL; the sharing segments use the RDL of the base segment. Therefore, when you modify the RDL of a base segment, the RDL of every segment that shares the FDT changes too.

To find whether a segment shares the FDT of another segment, see the Stored Data Report for a database. If the report indicates that a segment shares the FDT of another segment, the report identifies the base segment of the FDT and the database where it is found. If the report indicates that the segment is the base for a shared FDT, the report lists all of the segments sharing the FDT.

Typically, an FDT is shared only between segments in the same database. We recommend that you do not share an FDT between databases, though it is possible to share an FDT created for a segment in another database.
The following methods are available for FDT sharing:

- **Method SHR=SFS** (see page 15)—This method is valid for the segments that have the same field structure.
- **Method SHR=YES** (see page 15)—This method is valid for the segments that have the same basic characteristics, but the structure may be different.

**Follow these steps:**
1. Select Option 3, Modify User Parameters, from the IMS Support Services Menu.
   The User Parameter Maintenance Menu panel appears.
2. Select Option 4, FDT User Parameters.
   The IMS User FDT Parameter Maintenance panel appears, displaying a list of the FDT control parameters for each segment in the database.
3. Specify FDT sharing settings in the following fields:
   - FDT NAME
   - SHR
   - SRCE DBD
   - SRCE SEG
   Press Enter each time after you change a segment to save your update and continue.
   Press the End PF key after you updated all segments to leave the FDT parameter maintenance panel.
   The FDT share settings are saved.
Method 1—SHR=SFS

For segments that have the same field structure, specify the following FDT sharing parameters:

- Specify an SHR=SFS for all segments that share the FDT.
- For base segment:
  - Enter the name of the FDT to be shared in the FDT NAME field.
  - Enter the name of the segment (SEG NAME) in the SRCE SEG field.
  - If the segment is in another database, enter the name of the DBD in the SRCE DBD field. Otherwise, the field is blank.
- For other segments:
  - Enter the name of the FDT to be shared in the FDT NAME field.
  - Enter the name of the base segment in the SRCE SEG field.
  - If the base segment is in another database, enter the name of the DBD in the SRCE DBD field. Otherwise, the field is blank.

Method 2—SHR=YES

Use the SHR=YES for segments that have the same basic characteristics but different field structure. The segments must have the same length noncompressible key area, and the remainder of the RDL for the segment is defined as C1VER. If these two conditions exist, the segments can share the RDL:

\[ Nx, C1VER \]

\[ \times \]

Specifies the noncompressible key area.

When using the SHR=YES method, specify the following FDT sharing parameters:

- Specify SHR=YES for all segments.
- For the base segment, do not change the value of the fields SRCE DBD and SRCE SEG.
- For other segments:
  - Enter the name of FDT in the FDT name field.
  - Enter the name of the base segment in the SRCE SEG field.
  - If the base segment is in another database, enter the name of the DBD in the SRCE DBD field. Otherwise, this field is blank.
Enter or Update the RDL for a Database Segment

You can enter a prepared custom RDL for a database segment.

**Important!** Do not change the RDL for a database segment until you have verified that the segment does not share an FDT with another segment.

**Follow these steps:**

1. From the IMS Support Services Menu, select Option 3, User Parameters Maintenance.
   
   The IMS User Parameter Maintenance Menu appears.

2. Enter the name of DBD you want to process in the DBD NAME field and press Enter.
   
   **Note:** If this is the first function you have selected for this session, the DBD name field is blank. If other functions have been performed, this field shows the name of the last DBD referenced.

   The DBD name is set up.

3. Select Option 3, RDL User Parameters.
   
   The IMS User RDL Maintenance Selection panel displays a list of the compressible segments in the database, together with the information about the segment key and the number of compressible bytes the segment contains.

4. Enter S in the select field in front of the segment you want to modify and press Enter.
   
   The IMS User RDL Parameter Maintenance panel appears, displaying the current RDL for the segment.

   **Note:** The RDL for the segment key is protected and cannot be altered. The remaining RDL is for the compressible area.

5. Enter or modify the RDL for the compressible area. You can use up to 11 lines of 78 bytes each for the segment RDL.

   **Note:** To discard the changes, press End PF key before you save the changes.

6. Press Enter.
   
   The new RDL is saved, and the list of segments appears again. The select field in front of the segment contains an asterisk (*) if the segment has been selected during this session. You can select segments multiple times for RDL maintenance in the same session. You can select another segment for RDL maintenance or can end the RDL maintenance task.

   **Note:** If you have made any errors in specifying the RDL, an error message appears 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 Help PF key. Correct the error in the RDL and press Enter.
Control the Size of a Compressed Segment

You can control the size of a compressed segment using the PAD facility or by specifying a minimum size for the segment, or both. The PAD is applied first and then checked against the MIN SIZE specification.

Follow these steps:

1. Select Option 3 from the Support Services Menu.
   The IMS User Parameter Maintenance Menu appears.

2. Select Option 4, FDT User Parameters.
   The IMS User FDT Parameter Maintenance panel appears, displaying a list of segments in the database.

3. Specify the segment size in the following fields:
   - PAD LEN
   - PAD TYPE
   - MIN SIZE

   Press End PF key.
   The segment size settings are stored and the IMS User Parameter Maintenance panel appears.

Padding Facility for Compressed Segments

The padding facility tells CA Compress Data Compression for IMS to add padding to the segment after it has been compressed to the smallest possible size. The padding is done only for segment insert. Padding helps reduce VSAM CI splits. You can use any of the following padding types:

- By adding a specific number of bytes to the compressed segment.
  To use this type, specify the following values:
  - PAD TYP = N
  - PAD LEN = number (1 through 255 bytes)
By padding the compressed segment by a percentage of the number of bytes saved during compression. The following formula is used to calculate the percentage:

\[ A + \left( \frac{(B - A) \times P}{100} \right), \]

- **A**
  - Specifies the length of the compressed segment.

- **B**
  - Specifies the length of the uncompressed segment.

- **P**
  - Specifies the percentage.

To use this type, specify the following values:

- `PAD TYP = P`
- `PAD LEN = number` (1 through 99)

**MINSIZE**

Specifies the minimum length of a compressed segment, beyond which the routines do not compress segments. Using the MINSIZE specifications helps reduce the probability of VSAM CI splits during replacement operations and the variation in length of compressed segments. During replacement operations, the segments are less likely to increase in length. You can change the MINSIZE value any time without reloading the database.

Specify 0 to compress the segment to its smallest possible size. A non-zero MINSIZE should be greater than the number of bytes from the beginning of the segment through the end of the segment key, if any, plus 5 bytes.

If omitted, MINSIZE defaults to 0 and no lower limit is imposed on the length of compressed segments.

**First-use Recommendation**

For the first-time use, we recommend the following:

- Specify one database to compress, rather than trying to compress several databases at one time.
- Use the default values for most steps.
- Use the data sample that contains 10 percent of the total occurrences of each segment to compress.
How to Test an IMS Database Compression

Before you attempt a database compression using CA Compress Data Compression for IMS, test the database compression settings using a data sample. Testing database compression on a data sample helps you judge the effectiveness of the compression settings for each database segment and customize the compression settings.

This procedure does not compress the actual database, but simulates the compression process on the extracted sample. The results of the database compression test are printed to your default SYSOUT class. A Test Compression report (see page 76) indicates the compression percentage that you can expect using the current compression controls.

Typical compression results using default Record Definition Language (RDL) range from about 30 to 50 percent, and may vary even further, due to the nature of the specific database. If your compression test results are in or near this range, and you find the total compression results acceptable, take the next step, and compress your database. If the compression test results are unacceptable, do any of the following:

- Specify another database for testing compression.
- Customize the RDL to improve compression.

We recommend that you first gain experience on a few databases using the default values before you attempt any fine-tuning. All results of the definition process to date are saved in the Interactive Dialog Control File, and you can fine-tune and implement compression on the database later. Remember that fine-tuning the RDL for a segment can increase compression, but can also increase the CPU overhead.

To test a database compression, follow these steps:

1. (Optional) View results of previous compression tests (see page 20)
2. Analyze the database (see page 20)
3. View results of database analysis (see page 21)
4. Specify parameters for data sample extraction (see page 21)
5. Generate JCL for database compression test (see page 24)
6. Run database compression test (see page 26)
7. (Optional) Refine RDL for segments (see page 26)
View Results of Previous Compression Tests

Before you run a database compression test, determine if the database has already been analyzed and tested, so that you can compare results of the previous and current test runs. The Index List report (see page 58) lists all of the databases that are currently defined to the Interactive Dialog Control File with the information about database analysis and results of compression tests.

To view results of previous compression tests, select option I from the IMS Support Services Menu.

A list of the recorded databases appears. To produce a printed copy of the list, press the End PF key.

Analyze the Database

Database analysis introduces a database to CA Compress Data Compression for IMS and provides you with detail information about the structure and content of a database. Use this information to specify proper parameters for data sample extraction.

Follow these steps:

1. Select Option 1, Support, from the Interactive Dialog Primary Services Menu.
   The IMS Support Services Menu appears.
2. Select Option 1, Analyze a Database, from the IMS Support Services Menu.
   The Database Analysis panel appears.
3. Provide the following information and press Enter:
   - The name of the DBD to be analyzed.
   - The name of the DBDLIB that contains the DBD load module. The standard DBDLIB name or the name of the last DBDLIB that you used displays initially. If the DBDLIB name is the same, scroll down to the database title field.
   - The title of the IMS database. We recommend that you provide a meaningful title for the database.

The Interactive Dialog obtains and analyzes the DBD load module and stores the results in the Interactive Dialog Control File.

Note: When the analysis is complete, a message appears in the upper right-hand corner of the screen, notifying you on the successful completion or an error. If the meaning of the message is not clear, press the Help PF key and a complete message appears. If the complete message requires more space than is available on one screen, a second message directs you to press the Help PF key again to display subsequent panels.
View Results of Database Analysis

When the database analysis is performed, the information about the database is stored in the Interactive Dialog Control File. The Stored Data report (see page 59) shows the results of the database analysis.

Follow these steps:

1. Select Option 1, Support, from the Interactive Dialog Primary Services Menu.
   The IMS Support Services Menu appears.
2. Select Option 2, Generate a Database Report, from the IMS Support Services Menu.
3. Enter the DBD name of the IMS database that is the base for the report and press Enter.
   The Stored Data Report appears.
4. Press the End PF key.
   The Report Printing Options panel appears.
5. Specify the report printing options as follows and press Enter:
   ■ Enter YES in the HARDCOPY OPTION field.
   ■ Enter any valid SYSOUT class in the SYSOUT CLASS field.
   ■ Enter the number of copies that you want to print in the NUMBER OF COPIES field.
   The Stored Data report is printed to the specified SYSOUT print class.

Specify Parameters for Data Sample Extraction

Based on the database information provided in the Stored Data Report and the SMU11 or reorganization report, you can specify criteria for data sample extraction. The following criteria determine the data sample:

■ The beginning and end of the area in the database from where you extract the sample
■ The quantity of occurrences to select for the sample
The following extraction parameters specify the data to include in the sample:

**BYPASS**

Specifies the number of database records (in thousands) to bypass before the extraction of the data sample begins.

**Default:** 0

**Example:** Enter 10 to start the extraction after 10,000 occurrences of the segment have been bypassed.

**EXTRACT**

Specifies the number of database records to extract to form the data sample. The following values are available:

**ALL**

Specifies that the extraction progresses all the way through the database after the extraction has begun. This is the default.

**number**

Specifies the number of database records (in thousands) to be extracted to form the data sample.

**Example:** Enter 10 to stop the extraction after 10,000 occurrences of the segment have been extracted.

**SKIP**

Specifies the number of database records to skip between each database record that is extracted.

**Default:** 0

**Examples:**

- Enter 0 to extract all records once extraction has begun.
- Enter 1 to extract half of the segments (skips every other record).
- Enter 3 to extract 25 percent of the segments (skips three records between each extracted record).

**Note:** The percentage values are based on the segments between the beginning and ending boundaries specified for the extract. Only if you use the default values for the BYPASS and EXTRACT parameters the percentage reflects the entire database. Increasing the value of the SKIP parameter decreases the quantity of segments extracted.
To receive valid and useful data sample for compression test with a reasonable use of resources, consider the following when you specify the data sample extraction parameters:

- The data sample should contain 10,000 occurrences of a segment.
- If a segment has fewer than 20 bytes of compressible data, specify the EXTRACT parameter with a value of 000.
- If a segment has fewer than 10,000 occurrences, specify the EXTRACT parameter with a value of ALL.
- If a segment has from 10,000 through 20,000 occurrences, extract at least 10,000 by specifying the EXTRACT parameter with a value of ALL, and the SKIP parameter with a value of 1. You can also extract ALL occurrences by leaving the SKIP parameter at its default value of 0.
- If a segment has more than 20,000 occurrences, extract at least 10,000 occurrences and use the SKIP or BYPASS parameters to spread the extraction across the entire database.
- If a segment has more than 50,000 occurrences, limit the extraction to 20 percent of the occurrences.

Follow these steps:

1. Select option 1, Support, from the Interactive Dialog Primary Services Menu.
   The IMS Support Services Menu appears.
2. Select option 3 from the IMS Support Services Menu.
   The IMS User Parameter Maintenance Menu panel appears.
3. Enter the name of the database and option 2 and press Enter.
   The IMS Data Sample Extract Parameter Maintenance panel appears, displaying the root segment name.
4. Specify the extraction parameters and press the End PF key.
   For the first-time use, we recommend that you specify the default values as follows:
   - Specify 0 for BYPASS or SKIP.
   - Specify ALL for EXTRACT.
   The IMS User Parameter Maintenance Menu appears.
5. Select option 5, Data Set Names.
   A list of required ddnames that were recorded when the DBD was first analyzed appears. All of the DBDs that were directly or indirectly associated with the current DBD were examined.

6. Enter or correct the data set names for each ddname to extract the sample data directly from the database using DL/I or HSSR calls, and press the End PF key. You can enter one or more data set names.
   The database is identified, and the IMS User Parameter Maintenance Menu panel appears.

**Generate JCL for Database Compression Test**

The database compression test is performed by executing the following JCL members:

- Extract JCL—Extracts the data sample.
- Byte distribution and test compression JCL—Examines byte distribution and database compression controls.

**Follow these steps:**

1. Select option 4, Generate JCL for Testing, from the IMS Support Services Menu.
   The IMS JCL Generation Menu panel appears.

2. Enter the DBD name and press Enter.
   The IMS JCL Generation Menu for the specified database type appears.
3. Enter the database identification information, specify parameters for JCL generation as follows and press Enter:

   **JOB CARDS**

   Specifies the JOB statement to use with the test jobs.
   
   **Limits:** four lines

   **MBR NAME PREFIX**

   Specifies the prefix to use when creating the JCL members in the GENLIB. The first member is the extract JCL, the second member contains the byte distribution and the test compression JCL. Generating the JCL for both members is done only once for each extraction selection.
   
   **Limits:** seven characters

   **JCL TYPE**

   Specifies the type of the data sample extraction. The following values are available:
   
   **1**
   
   Extracts data sample from database using batch DL/I calls.
   
   **2**
   
   Extracts data sample from database using BMP DL/I calls.
   
   **3**
   
   Extracts data sample from database using batch HSSR calls.
   
   **Note:** You can select this option only if your installation has installed the HSSR facility.
   
   **X**

   Exits JCL generation session without generating any JCL.

An IMS Data Extraction JCL Generation panel for the specified database type and extract option appears.

4. Enter or verify the information required for each selection.

   - For option **1**, Extract using Batch DL/I calls, enter the following:
     
     - PSB name and PCB number
     - Library (PSBLIB or ACBLIB) to retrieve the PSB from
     - Correct PSBLIB, ACBLIB, and RESLIB data set names

   **Note:** PSB language must be COBOL or Assembler. PL/I is not supported. The PCB must have sensitivity to every segment in the database.
For option 2, Extract using BMP DLI calls, enter:
- PSB name and PCB number
- IMS Control Region ID
- Correct RESLIB data set name

Press Enter.
The second IMS Data Extraction JCL Generation panel appears.

5. Enter the required allocation information for the extracted sample data set and press Enter.
Two JCL members are generated.

Run Database Compression Test

To run database compression test, execute the two JCL members you created in the previous step as follows:

1. Execute the extraction JCL.
2. Execute the byte distribution and test compression JCL.

The results of the database compression test are printed to your default SYSOUT class.

Note: You can rerun byte distribution and test compression JCL after modifying the RDL to obtain better compression results. In this case, resubmit the JCL from the second member only.

Now that you have tested compression settings on a data sample, you can analyze the test results. If your compression test results are not acceptable, you can specify another database for compression test or customize the compression settings. If you find the total compression results acceptable, you can implement compression on your production database.

How to Refine RDL for Segments

When you run a compression test, CA Compress Data Compression for IMS builds a record of compressions achieved, compares the compression record to other test runs, and stores the RDL information in the Control File. You can view the results of test compression in the Test Compression (see page 76) report.

Retain these reports when you perform repeated testing. When the Best Test field on the Test Compression Report does not reflect the current test number, refer to the report for the best test to re-enter the RDL producing the best results for the segment.
You can respecify RDL for various segments and perform the testing procedure again. If you change the RDL for a segment, the **Byte Distribution Analysis** (see page 73) (BDA) for the segment is invalidated and the procedure must be re-executed. The process can be repeated as often as needed until you have obtained the compression that you want to achieve for the database.

As you fine-tune the RDL for the segments in the database, follow these steps:

1. Specify the RDL you have devised for each segment. Choose Option 3, RDL User Parameters, from the User Parameter Maintenance Menu.
2. Specify any FDT sharing to be used. Choose Option 4, FDT User Parameters, from the User Parameter Maintenance Menu. See Sharing an FDT Between Two or More Segments in this chapter for more information.
3. Execute the JCL to calculate byte distribution and perform test compression.
4. Evaluate the results of the test compression.

---

**Compress an IMS Database with CA Compress Data Compression for IMS Controls**

Follow these steps:

1. Select Option 5, Implement Database Under CA Compress Data Compression for IMS Controls from the IMS Support Services Menu. The IMS Database Implementation Menu appears.
2. Enter the DBD of the database and press Enter. The IMS Database Implementation Menu appears.
3. Select Option 1, Select Segments for Compression. The IMS Segment Selection panel appears, displaying the FDT SHARE status (a default value of NO displays for each segment for a given database), and an area for comments.
4. For each segment in the list, enter one of the following:
   - F—Compress using an FDT compression
   - E—Compress using Express compression
   - C—Cancel any previous compression for a given segment
   Press End PF key. The segment selection is saved and the IMS Database Implementation Menu appears.
5. Select Option 2, Generate Modified DBD Source Macros. The DBD Generation panel appears.
6. Specify the following information and press Enter:
   ■ Source DBDLIB
   ■ Source DBD Name
   ■ CA-Compress DBD Name
   ■ (Optional) Reorganization DBD Name

   **Note:** The purpose of the optional Reorganization DBD Name macros is to permit the reorganization of the database without having to expand and compress the segments.

   The SEGM macro is modified for each flagged segment. For an FDT compression, SEGCC macros are created and inserted for each segment for which an FDT was generated. The modified DBD macros are stored in GENLIB.

7. Exit the Interactive Dialog.

8. Assemble the DBD macros for the database.

   **Note:** To make the SEGCC macro accessible to assemble the new DBD, concatenate the CA Compress Data Compression for IMS sample library to the SYSLIB DD statement of your DBD Assembly proc.

9. Unload the database specifying the old DBD.

10. Move the new sources to the Production Library as follows:
   ■ For the FDT compression, move the new FDTs and DBD macros into the Production Library.
       a. Move the FDT modules and CA Compress Data Compression for IMS load modules into IMSVS.RESLIB or into a separate library of your choice. If you do not place them in the IMSVS.RESLIB, modify the JCL of every job that accesses the database. The modification consists of concatenating the libraries that contain the FDTs and load modules to a STEPLIB or JOBLIB DD statement.
       b. Move the new DBD module into the appropriate DBD libraries and perform an ACB GEN to do a DBD BUILD.
   ■ For the Express compression, move the new DBD module into the appropriate DBD libraries and perform an ACB GEN to do a DBD BUILD.

11. Reload the database using the new DBD macros and FDTs.

   **Note:** If the necessary FDTs are not accessible to the compressed database, a 3997 ABEND occurs.

   When the reorganization is complete, the database is under the control of CA Compress Data Compression for IMS or Express.

   **Note:** The actual compression is performed during the reload procedure.
Delete an IMS Database From the Control File

If you determine that compressing the database is not desirable, there are two possible follow-up steps. You can leave the database information in the Control File for reference purposes, or you can delete the database from the Control File.

**Note:** You do not cause any harm by deleting information about a database that has already been compressed. However, we recommend that you do not delete the test results for the data sets that have already been or will be compressed using CA Compress Data Compression for IMS.

**Follow these steps:**

1. Select Option 8, Delete Obsolete Database Information, from the IMS Support Services Menu.
   The Delete Database from CA Compress Control File panel appears.
2. Enter the DBD name of the database to delete, and press Enter.
   The IMS DBD Data Deletion Confirmation panel appears.
3. Enter Y in the Confirm Delete field and press Enter.
   The information about the database is erased from the control file.

   **Note:** If you entered the wrong DBD name, enter N in the Confirm Delete field and press Enter or the End PF key to cancel the delete.

How to Change a Database Controlled by CA Compress Data Compression for IMS

When changes are made to the structure of an IMS database that has been already compressed and under the control of CA Compress Data Compression for IMS, adjust the compression control parameters. The process of adjusting the compression control parameters depends on the nature of the changes.

When the changes to the database affect the CA Compress Data Compression for IMS-controlled segments or the physical structure of the database changes, follow these steps:

1. **Generate a Stored Data report** (see page 21) with the information about the database that is currently stored in the Control File.
2. Create an uncompressed test version of the database in its new structure. This can be accomplished with a database reorganization or restructure which used the normal CA Compress Data Compression for IMS DBD to unload the data and the new structure DBD without any CA Compress Data Compression for IMS controls to do the restructure or restore.
3. Delete the information about the database from the Control File after the Stored Data Report has been generated.

4. Analyze the new database structure as if it has never been processed previously.

For the following kinds of changes, modify the CA Compress Data Compression for IMS version of the DBD for the macro:

- For changes in the length of a segment under the control of CA Compress Data Compression for IMS, correct the bytes parameter in the SEGM macro for the segment as follows:
  - For variable-length non-DEDB segments, the maximum length required is 12 bytes larger than the defined length.
  - For fixed-length segments compressed with a variable RDL, such as C1VER, or DEDB segments, the length remains the same.

  **Note:** For fixed-length segments compressed with a fixed RDL, such as C1Fxx, execute the previous four steps to build a new FDT.

- For changes to any of the segments not under the control of CA Compress Data Compression for IMS, regenerate the DBD using the Interactive Dialog DBD generation service.

---

**Refresh FDTs without Bringing IMS Down**

Refreshing FDTs while IMS is still running can help you when adding actual data to a segment that was defined but had no occurrences in the database.

**Important!** Perform this procedure with extreme care. Using the wrong FDT can cause data loss.

**Follow these steps:**

1. Stop all the databases that use the FDT.
2. Restart all the databases that use the FDT. This causes a fresh copy of the FDT to be loaded from the FDT library.
Recreate a Lost or Damaged FDT

FDTs are generated into the Interactive Dialog FDTLIB. The information used to create FDTs and a copy of the FDT are stored in the Interactive Dialog Control File.

As each FDT is created, it is assigned a control number that is created and maintained by the Dialog. The control number, more commonly referred to as the Integrity Check Block (ICB), identifies a specific version of the FDT, no matter how many times that FDT is created. Each FDT has a unique ICB which becomes part of each segment or record compressed. The ICB contains a 14-bit binary number; given the length of this binary number, the Dialog can generate and identify 16,383 unique FDTs.

The Production FDT Services of the Interactive Dialog let you list all FDTs and regenerate any FDTs that the original load module has been destroyed for.

Follow these steps:
1. Select option 8 from the Primary Services Menu of the Interactive Dialog.
2. Select option I to Generate a Stored FDT Index List.
3. Determine the FDT ID of the FDT by finding the name of the FDT on the list. Because several versions of an FDT might exist, verify the following:
   - The DBD name of the file code
   - The date of creation
   - The name of the creator
4. Select option 1 from the Primary Services Menu.
5. Enter the FDT ID. The information regarding the FDT displays.
6. Enter Y or N as appropriate to confirm that the FDT is correct. The FDT is now generated and placed in the Interactive Dialog FDTLIB.
7. Move the new FDT to the library where your production FDTs reside.
Chapter 3: Database Compression without the Interactive Dialog

This section contains the following topics:

How to Compress a Database without the Interactive Dialog (see page 34)
How to Compress a Database without the Interactive Dialog using Express (see page 53)
How to Compress a Database without the Interactive Dialog

The applications programs require only minor JCL additions (for example, STEPLIB DD statements to FDT library and IMS modules) to access compressed database segments.

The whole process of implementing CA Compress Data Compression for IMS without the Interactive Dialog comprises the following phases:

1. **Database Prepass** (see page 36)—The database segments to compress are defined with the standard Record Description Language (RDL). The IMSPASS utility executes as an IMS application program to generate a File Descriptor Table (FDT) for each segment type.

2. **Database Compression** (see page 48)—The database to compress is unloaded, the IMSHRINK interface is specified as the compression routine in a DBD generation and the database is reloaded. Database reload invokes the user IMSHRINK exit, resulting in a database with compressed segments. IMS execution invokes IMSHRINK (based on DBD specification) for retrieve, insert, replace, or load operations, as shown in the following illustration.
User Constraints

General constraints are as follows:

- The PSB referenced in IMSPASS cannot show PL/I as the program language.
- Compressed segments cannot be processed by a user-supplied edit exit. Only one COMPRTN is allowed per SEGM macro. This is an IMS constraint.
- The entire segment sequence field, from the start of the segment through the end of the key, must be left uncompressed. This is done through the type N RDL specification.
Database Integrity

CA Compress Data Compression for IMS helps minimize the loss of database integrity or the possibility of abnormal termination in a production environment by performing the following:

- Self-checking and error analysis features to trap specification errors during execution of the IMSPASS utility, DBD generation, or the IMS reorganization or reload utility.
- Including redundancy-checking logic to verify that each expanded record (segment) is identical to the record before compression.

Assuming initial database integrity and error-free application programs, further errors should not occur.

How to Define a Database to CA Compress Data Compression for IMS

To define a database, perform the following actions for each segment type to be compressed:

1. Specify the characteristics of the segment type using the Record Definition Language (RDL).
   
   Each segment type can be defined in fine detail, to the field level or less. The level of detail depends on the knowledge of segment content and an evaluation of the trade-off between compression and overhead. The segment key fields are described as fields that are not compressed. The remainder of the segment type is described to whatever extent you need to balance compression obtained and execution performance.

2. Execute the IMSPASS utility.
   
   This utility creates the File Descriptor Table (FDT) for the segment type. The FDT contains compression and expansion data that CA Compress Data Compression for IMS uses to encode or decode a segment. If a segment shares an FDT, do not execute IMSPASS for the segment.

   IMSPASS is executed once for each segment type to be compressed. Trial compression statistics are then analyzed, and a determination is made regarding whether coding RDL compression specifications in finer detail is required to obtain better compression.
3. (Optional) Review the RDL to improve compression. To permit efficient experimentation, the prepassed segments read by IMSPASS can be written to a sequential data set (OUTDB). The File Prepass Utility then reads this data set to evaluate the effectiveness of the revised segment description and to create the FDT for the segment.

4. Convert the FDT to load module format on the FDT library using the FDTLOADR utility. When the compression is satisfactory, the FDT is loaded onto the FDT library with the FDTLOADR Utility.

The following diagram illustrates the process of defining a database to CA Compress Data Compression for IMS.

**Database Definition to CA Compress Data Compression for IMS**
How to Describe Segments with RDL

Use the Record Definition Language (RDL) to describe the segment types of the database to be compressed. RDL lets you describe various data field types within a segment to indicate explicitly the compression technique to use. The RDL specification for the segment type is input to the IMSPASS utility as the data set with DDNAME RECDEF.

The following basic characteristics of the segment must be defined with RDL:

- The location and length of the segment key (sequence field) that is not to be compressed. The segment key is defined as a type N field.
- The length of the data portion of the segment that is to be compressed using the Huffman encoding technique. The remainder of the segment is defined as a type C field.

Note: RDL lets you define more detailed description of the segment type. All field type compression specifications are permitted by CA Compress Data Compression for IMS, except pattern matching (codes MA, MB). For information about RDL syntax, see the Appendix A: Record Definition Language (see page 97).

Fixed-Length Segments Definition

To define fixed-length segments, specify the following:

Nx,C1Fy

x

Specifies the key length.

Limits: up to four digits

y

Specifies the data length.

Limits: 1 through 16,384

If the segment is not keyed, omit the "N" value.
Example: Fixed-length Segment

Consider the following segment:

<table>
<thead>
<tr>
<th>key</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>120</td>
<td>136</td>
</tr>
</tbody>
</table>

To define this segment with RDL, specify the following:

N120,C1F136

The previous syntax specifies no compression for 120 bytes and character encoding for a fixed length of 136 bytes.

Variable-Length Segments Definition

The RDL specification for variable-length segments includes definition of the 2-byte length indicator (LL), using the value in the length indicator to calculate the length of the data portion. To define variable-length segments, specify the following:

V2-f,Nx,C1FVS.

f

Specifies the offset of the first position beyond the key (x+2).

x

Specifies the key length.

Limits: up to four digits

If the segment is not keyed, omit f=2 and the "N" values.

Example: Variable-length Segment

Consider the following segment:

<table>
<thead>
<tr>
<th>LL</th>
<th>key</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>

To define this segment with RDL, specify the following:

V2-102,N100,C1FVS

The previous syntax specifies a 2-byte length field, no compression for 100 bytes, and character encoding for variable length (calculated as segment length minus 102).
IMSPASS Utility

The IMSPASS utility executes the following:

- Reads and edits the RDL statements describing the segment type.
- Validates the RDL specifications by a prepass of a user-specified sample of segments.
- Performs trial compression (optional) and calculate compression statistics for a user-specified sample of segments. Compression is performed, but the database is not altered.
- Generates the File Descriptor Table (FDT) for the segment type.

A separate IMSPASS step must be executed for each segment type to be compressed. The utility executes as a batch IMS application program. IMSPASS execution is controlled by an input file with keyword parameters, most likely an input stream data set:

**imsnames**

I/O: both

Specifies standard IMS JCL requirements for batch program execution, defining the database to DFSLI000.

**PARMFL**

I/O: I

Specifies the IMSPASS Parameter File specifying execution options.

**RECDEF**

I/O: I

Specifies the RDL statements describing the segment type.

**TABL00**

I/O: O

Specifies the FDT for the segment type prepassed. A sequential data set.
OUTDB
I/O: O
Specifies the sequential output data set (optional) of prepassed segments. This file can be input to the File Prepass Utility with different RDL specifications to determine the optimal compression method.

IMSPASS forces the following DCB specifications for this data set:
- RECFM=U is used for both fixed-length and variable-length segments. The RDL input to the File Prepass Utility specifies the actual record (segment type) format.
- BLKSIZE reflects the largest segment length of the data set. The IMSPASS output indicates the actual maximum length.

OUTFILE
I/O: O
(Optional) Specifies a dummy data set required only if trial compression statistics are accepted (you specify the C= parameter). The DD statement is as follows:

//OUTFILE DD DUMMY, DCB=(LRECL=x,BLKSIZE=y)

x
Specifies the maximum segment size + 8.

y
x + 4

PRINT
I/O: O
Specifies a print data set (SYSOUT) to record RDL input, prepass and compression statistics, and error messages.

DCB=(LRECL=121,RECFM=FB, . . . )

SYSUDUMP
I/O: O
Specifies a print data set (SYSOUT).
**IMSPASS Parameters**

The IMSPASS Parameter File (PARMFL) is an 80-character input record that contains the following:

- Keyword parameters separated by commas appearing in columns 1-71.
- More than one record may be input. Continuation is indicated by a comma following the last parameter on the record.
- The last record is recognized by the presence of data in column 71 (except for a comma), or the lack of a comma following the last parameter of the card. The BEGIN parameter is an exception.

**SEG=segname**

(Required) Specifies the name of the segment. The name of the segment must correspond to the SEGM macro segname specification.

**P=nn|ALL**

(Required) Specifies the number of records (in thousands) to be prepassed. The following values are available:

- **ALL**
  - Prepasses the entire database.
- **nn**
  - Prepasses the specified number (in thousands).
  
  **Example**: P=003 prepasses 3000 records or the entire database, whichever occurs first.

**C=nn|ALL**

(Optional) Specifies the number of records (in thousands) to be subjected to trial compression. We recommend that you use this parameter to obtain compression statistics and verify RDL specifications beyond syntax check. The following values are available:

- **ALL**
  - Specifies the entire database.
- **nn**
  - Specifies the number of records (in thousands).

  **Example**: C=002 prepasses 2000 records or the entire database, whichever occurs first.
PCBNAME=dbname | PCB=n (mutually exclusive)

(Optional) Specifies the name of the DBD from the PCB.
Specify the relative number of the PCB in the PSB. If omitted, PCB=1 is assumed.

Note: If the PSB was generated with CMPAT=YES, a dummy PCB is the first PCB in
the PSB and must be accounted for when using PCB=n.

OUT=P | C

(Optional) Specifies the number of records to write to the OUTDB data set. If
omitted, no records are written.

P
Writes the number of records corresponding to the number of records
prepassed (OUT=P).

C
Writes the number of records corresponding to the number of records "trial"
compressed (OUT=C).

SKIP=n

(Optional) Specifies the number of input records to skip during IMSPASS execution
to obtain a better cross-section of the segment type.

Limits: decimal value from 1 to 9999.

Example: SKIP=1 specifies the processing of records 2, 4, 6, and so on. SKIP=2
specifies processing of records 3, 6, 9, and so on.

SKIPTO=n / BEGIN=key (mutually exclusive)

(Optional) Specifies where to start prepass.

SKIPTO=n

Specifies the number of input records to skip before beginning prepass.

Limits: decimal value from 1 to 99999

Example: SKIPTO=1000 indicates start prepass at record 1001.

BEGIN=key

Specifies the key at which prepass is to begin. Verify that the following rules
apply:

- This parameter follows the PCB or PCBNAME parameter.
- The key specified has the same length as the segment key.
- Code the key in character format (use multipunch for packed decimal
  representations). IMSPASS obtains the key length from IMS control blocks.
  If coding of the key requires use of column 71 of the input record,
  continuation is automatic. Do not code a comma to indicate continuation.
Example: IMSPASS Execution JCL

```plaintext
/*IMSPASS JOB
/*
****************************************************************************
/* *
/* THE FOLLOWING JCL IS PROVIDED FOR IMS USERS TO *
/* BUILD THE FILE DESCRIPTOR TABLE (FDT)
/* *
****************************************************************************
//IMSPASS EXEC PGM=DFSRRC00,PARM='DLI,IMSPASS, ...'
//STEPLIB DD DISP=SHR,DSN=ca-loadlib   (1)
   other private libraries DD statements
   and standard IMS execution DD statements
//DFSRESLB DD DSN=ims-reslib,DISP=SHR
//PRINT DD SYSOUT=*  
//SYSUDUMP DD SYSOUT=* 
//TABL00 DD DISP=(,CATLG,DELETE),DSN=user.FDT.FILE,  
//UNIT=unit,VOL=SER=volser,SPACE=(TRK,(2,1))  
//RECFDEF DD *
   record definitions   (2)  
/*
//OUTDB DD DISP=(,CATLG,DELETE),DSN=user.OUTDB.FILE, 
//UNIT=unit,VOL=SER=volser,SPACE=(TRK,(n,n))  
//OUTFILE DD DUMMY,DCB=(LRECL=nn,BLKSIZE=nn)  
// ddname DD DISP=SHR,DSN=user.DBD   (3)
//PARMFL DD *
   SEG=segname,P=xxx, ... , ...   (4)  
/*  
Notes:

(1) The DSN assigned to the CA Compress Data Compression for IMS load library.

(2) RDL specification.

(3) The DDNAME and DSN of the DBD library for the specified segment.

(4) The IMSPASS parameter statement.
```
How to Share an FDT

Sharing the File Description Table (FDT) between segment types helps you save CSA storage (5 KB to 24 KB per segment type):

- For RDL statements associated with an FDT that apply to several segment types and their data characteristics are similar, specify the following parameter in their SEGCC macros:

  $$FN=\text{fdtname}$$

  *fdtname*

  Specifies the name of the common FDT.

  This technique is often employed when a large database has been partitioned into smaller, more manageable databases, each with essentially the same segment types. For example, if there were seven compressed segment types in each of 20 similar databases, we require only seven FDTs instead of 140.

- For RDL statements associated with an FDT that applies to segments with different structure (for example, fixed and variable, different lengths, key lengths), specify the following parameter in the SEGCC macro:

  $$FN=(\text{fdtname}, \text{SHR})$$

  The FDT must contain a type C1 field in its associated RDL. Only the type C1 data characteristics are extracted from the FDT, and these characteristics approximately matches the nonkey portion of the sharing segment.

  For example, if the FDT describes fields containing a mix of packed-decimal and character data, the current segment should have more or less the same mix. If the segment contained binary data, expansion rather than compression might result.

A shared FDT need not correspond to any single segment type, but may be a merge of several segment types. To construct such an FDT, follow these steps:

1. Collect a representative data sampling from the various segments by running the IMSPASS utility for each, with the following specification:

   ```
   OUT=P specified in PARMFL
   DISP=(MOD,KEEP) on the OUTDB DD statement
   ```

2. Run the File Prepass Utility using the OUTDB data set from step 1 as the input data set INFILE. Specify the RDL statement, $N_x, CIVER$, for the RECDEF, where $x$ is the approximate average sequence field length. The average rather than the maximum sequence field length can be used, because for shared FDTs the RDL is not used to determine where compression starts.

3. Run the FDTLOADR Utility.

4. Unload the databases with the old DBD and old FDTs if any.
5. For each segment type, code the SEGCC parameter
   \( FN=(\text{fdtname}, \text{SHR}) \).

6. Reload the databases with the new DBD and FDT.

**Review RDL with the OUTDB Data Set (Optional)**

You can review and tailor your RDL specifications as needed using the OUTDB option in the IMSPASS utility. The revision of RDL with the OUTDB data set helps save the overhead of repeated accesses of the database.

The OUTDB sequential data set becomes the input data set (INFILE) to the File Prepass Utility. Descriptions for the P and C values of the PARM field are the same as described for the PARMFL DD statement of the IMSPASS Utility. After determining the proper RDL to use, execute IMSPASS again to create an FDT. Do not use the TABL00 file from the File Prepass Utility. For this reason, TABL00 is defined as a temporary data set.

To review RDL with the OUTDB data set, specify the following in the IMSPREPASS utility:

- the OUTDB DD statement
- the OUT parameter of the IMSPASS PARMFL.

**Example: File Prepass Utility JCL for RDL Revision**

This example shows the File Prepass Utility JCL for RDL revision:

```plaintext
//**IMSPASS2 JOB
//*
//===***********************************************************************************
//** *THE FOLLOWING JCL IS PROVIDED FOR IMS USERS TO REVISE THEIR RDL SPECIFICATION*
//** ************************************************************************************
//PREPASS EXEC PGM=SHRINK,PARM='P=ALL,C=ALL
//PREPLIB DD DISP=SHR,DSN=hlq.loadlib
//PRINT DD SYSOUT=* 
//SYSUDUMP DD SYSOUT=* 
//INFILE DD DISP=OLD,DSN=user.outdb.file
//OUTFILE DD DUMMY
//TABL00 DD DISP=(,PASS),DSN=$FDTFILE,UNIT=unit,SPACE=(2048,(3,1))
//RECDEF DD *
// revised RDL specification
//*

Note:

(1) The DSN assigned to the CA Compress Data Compression for IMS load library.
FDTLOADR Utility

The FDTLOADR utility converts the File Description Table (FDT) sequential data sets to a load module format. This conversion lets you group FDTs within a partitioned data set (PDS), where each FDT is a separately named PDS member.

Conversion of FDTs to load module format offers the following advantages:

- The need for a separate TABLxx DD statement for each FDT used in an application program is eliminated.
- Only a single STEPLIB DD statement identifying the PDS containing all of the FDTs used in a job step is required.

Thus, JCL coding is reduced in application programs processing more than one compressed data set, and the storage of FDTs is centralized for ease of control and maintenance.

FDTs in load module format are accessed from application programs in a slightly different manner than FDTs in sequential data set format. Converting existing FDTs to load module format requires minor modifications to application programs that are already coded to process FDTs in sequential data set format.

When executing the FDTLOADR utility, consider the following rules:

- The IMSPASS TABL00 data set corresponds to the TABL00 data set of the FDTLOADR Utility.
- The FDT load modules must be stored in a special FDT load module library (SYSLMOD), that is required when executing CA Compress Data Compression for IMS.
- The module name "fdtname" must match the fdtname defined in the FN field of the SEGCC macro.
- In MVS, this load module is loaded into Common Storage Area Example (CSA). The names assigned to FDTs must be unique among all possible occupants of CSA.

FDTLOADR Utility Syntax

This utility has the following syntax:

```
//CONVFDT EXEC PGM=FDTLOADR
//STEPLIB DD DISP=SHR,DSN=hlq.loadlib
//SYSLIN DD UNIT=SYSDA,SPACE=(TRK,(2,1))
//SYSUT1 DD UNIT=SYSDA,SPACE=(1024,(50,20))
//SYSPRINT DD SYSOUT=*
//SYSDUMP DD SYSOUT=* 
//SYSLMOD DD DISP=SHR,DSN=user.FDT.LOAD(fdtname)
//TABL00 DD DISP=(OLD,DELETE,KEEP),DSN=user.FDT.FILE
```
How to Compress a Database without the Interactive Dialog

This utility has the following DD statements:

**SYSLIN**

Specifies the linkage editor primary input data. The FDTLOADR utility populates the data set and passes it to the linkage editor.

**SYSUT1**

Specifies the optional temporary work data set.

**SYSPRINT**

Specifies the data set where the utility prints messages.

**SYSDUMP**

Specifies the dump data set.

**SYSLMOD**

Specifies the PDS library and the member name where the FDT is stored in load module format. The example presumes that the PDS exists and is cataloged.

*user.FDT.LOAD*

Specifies the PDS library where the FDT is stored in load module format.

Substitute the name of the PDS with your value.

*fdtnname*

Specifies the member name for the FDT. Substitute the member name of the FDT in load module format with your value.

*Note:* FDT member names must be unique within a PDS library containing FDTs. However, we recommend that FDT member names be unique throughout all PDS libraries containing FDTs.

*Note:* The TABL00 DD statement defines the FDT in sequential data set format for conversion to load module format. Substitute the correct data set name that was specified in the execution of the File Prepass Utility that created the FDT. The example presumes that the FDT in sequential data set format was cataloged when created and should be deleted on conversion to load module format.

**How to Perform Database Compression**

To perform database compression, execute the following steps for each database to be compressed:

1. Unload the database using the IMS reorganization or unload utility, DFSURGU0, using the current DBD with no COMPRTN defined.

   Standard IMS procedures are used to unload the database to be compressed.

2. Generate DBD using an expanded SEGM macro and a CA Compress Data Compression for IMS SEGCC macro for each segment type to be compressed.
Use standard IMS procedures to assemble the DBD control statements. Verify that the library containing the CA Compress Data Compression for IMS SEGCC macro is referenced at assembly time.

3. Reload the database using the IMS reorganization or reload utility, DFSURGLO, and the new DBD. IMSHRINK is invoked as the compression exit to produce compressed segments.

Execution of the database reorganization or reload utility, DFSURGLO, follows standard IMS procedures with additional DD statements. The STEPLIB that contains the IMSHRINK and FDT load modules must be in the concatenation.

**Note:** For information about the IMSHRINK installation, see the *CA Database Management Solutions for IMS for z/OS Installation Guide*. 
The following diagram illustrates the process of database compression.

**IMSHRINK Operations**

- Uncompressed database
- Current DBD
- IMS reorganization or unload utilities
- Unloaded database
- FDT library (STEPLIB)
- CA Compress library (STEPLIB)
- SEG, SEGCC, DBD control statements
- DBD generation utility
- New DBD
- IMS reorganization or reload utilities
- Compressed database
**SEG Macro**

The SEG macro defines the segment type to be compressed. Code the SEG macro with the COMPRTN parameter for each segment type to be compressed exactly as follows:

(symbol) SEG NAME=segname, COMPRTN=(IMSHRINK, DATA, INIT), BYTES=[(max, min)],...

**Note:** If a segment is of variable length, alter the BYTES parameter as follows:

- **max**
  - Add 12 to the maximum value.

- **min**
  - Set to the length of the sequence field plus 4.

**SEGCC Macro**

The SEGCC indicates CA Compress Data Compression for IMS parameters to be used. Code one SEGCC macro for each SEG macro that specifies IMSHRINK as the compression routine.

The SEGCC macro has the following syntax:

segname
SEGCC FN=fdtname, MINSIZE=n, STAT=YES, PAD=(N, xxx) (fdtname, SHR)
NO (P, xx)

**segname**

Specifies the SEG macro. The value corresponds to the NAME=segname parameter of a SEG macro.

**FN**

(Optional) Specifies whether to override the default FDT load module name to be associated with this segment. If omitted, the FDT load module name is assumed to be the same as the segname. When the default name is unacceptable due to name conflicts, FN must be coded. Code the SHR subparameter if this segment is to use an FDT created for a different segment type by the IMSPASS utility. In this case, it is not necessary to run IMSPASS for this current segment.
MINSIZE

Specifies the minimum length of a compressed segment. Though it is not required, you can specify MINSIZE to reduce the probability of VSAM control interval splits during replacement operations. A MINSIZE specification can reduce the variation in length of compressed segments; during replacement operations, the segments are less likely to increase in length. A starting value for MINSIZE can be chosen from the trial compression statistics: a value between the minimum and average compressed segment length could be used. MINSIZE may be changed at any time without reloading the database. If omitted, MINSIZE defaults to zero and no lower limit is imposed on the length of compressed segments. If a MINSIZE other than 0 is specified, it should be larger than the number of bytes from the beginning of the segment to the end of the segment key plus 5 bytes.

STAT

Specifies whether compression statistics are to be written to the system output writer at the time of the first IMS "close" for each segment type.

YES

Writes the compression statistics to the system output writer at the time of the first IMS close. This is the default.

NO

Does not write the compression statistics.

PAD

Specifies how much padding should be added to the end of a segment on segment insert only. This can be used to help reduce VSAM CI splits.

N,xxx

Specifies a fixed number of bytes.

Limits: 1 through 255

P,xx

Specifies a percentage. The percentage is calculated as follows:

\[ A + \left( \frac{((B - A) \times P)}{100} \right) \]

- \( A \) — Specifies compressed segment length.
- \( B \) — Specifies uncompressed segment length.
- \( P \) — Specifies the required percentage.

Padding is not automatically included. If you omit it, no padding is added.

Limits: 1 through 99
Example: DBD Control Statements

```
DBD NAME=filename,ACCESS=VSAM, ...
SEGMENT NAME=SEGX,COMPRTN=(IMSHRINK,DATA,INIT), ...
SEGMENT NAME=SEGY,COMPRTN=(IMSHRINK,DATA,INIT), ...
.. (Additional SEGMENT macros. Not all segments need to be compressed.)
DBDGEN
..
.. FINISH SEGX SEGCC STAT=NO (1)
SEG Y SEGCC FN=FDT2 (2)
.. (Additional SEGCC macros)
END
```

Notes:

(1) FDT name is the same as segment name.
(2) FDT name is given explicitly.

How to Compress a Database without the Interactive Dialog using Express

The applications programs require only minor JCL additions (STEPLIB DD statements to CA Compress Data Compression for IMS Express modules) to access compressed database segments. Express does not use FDTs and therefore does not have a SEGCC macro.

To implement Express without the Interactive Dialog, add the COMPRTN parameter to the segment macro that defines a segment to be compressed.

The COMPRTN parameter has the following format:

,COMPRTN=(IMEXPRES,DATA,INIT)

Add the COMPRTN Parameter by Editing Existing DBD Source Macros

To activate the compression routine for a segment, edit the DBD source macro and add the ,COMPRTN= parameter to each SEGMENT to be compressed.
Add the COMPRTN Parameter Using a Data Dictionary to Generate DBD Source Macros

If you are using a data dictionary to generate DBD source macros, follow this procedure to add the COMPRTN parameter to DBD.

Follow these steps:

1. Follow the procedure called for by the dictionary to modify a segment definition.
2. Add the ,COMPRTN= name (IMEXPRES) to the segment definition following the appropriate parameter.
   The data dictionary is updated, and the new DBD source macros for the desired segments are generated from the data dictionary.
3. Assemble the new DBD source macros and reorganize the database (REORG).
   During the REORG, use the current DBD macro to unload the database, and the new DBD macro to load the database.
Chapter 4: Reports

This section contains the following topics:

- Common Product Reports (see page 55)
- IMS Reports (see page 58)

Common Product Reports

The following reports are common to all CA Compress Data Compression for IMS products:

- FDT Index Listing Report (see page 55)
- Authorized Users Report (see page 57)

FDT Index Listing Report

This report lists all FDTs currently in the Interactive Dialog Control file.

<table>
<thead>
<tr>
<th>FDT ID</th>
<th>FDT NAME</th>
<th>FACE</th>
<th>B NAME</th>
<th>TITLE</th>
<th>CREATED BY</th>
<th>DATE TIME</th>
<th>REL</th>
</tr>
</thead>
<tbody>
<tr>
<td>FDT00054</td>
<td>SEGMENTA IMS</td>
<td>G SAMPLEDB</td>
<td>TEST</td>
<td>MMM 523</td>
<td>JOHN DOE</td>
<td>01/11/02 11:41</td>
<td>5.2.5</td>
</tr>
<tr>
<td>FDT00055</td>
<td>SEGMENTB IMS</td>
<td>G SAMPLEDB</td>
<td>TEST</td>
<td>MMM 523</td>
<td>JOHN DOE</td>
<td>01/11/02 11:41</td>
<td>5.2.5</td>
</tr>
<tr>
<td>FDT00056</td>
<td>SEGMENTC IMS</td>
<td>G SAMPLEDB</td>
<td>TEST</td>
<td>MMM 523</td>
<td>JOHN DOE</td>
<td>01/11/02 11:41</td>
<td>5.2.5</td>
</tr>
<tr>
<td>FDT00057</td>
<td>SEGMENTD IMS</td>
<td>G SAMPLEDB</td>
<td>TEST</td>
<td>MMM 523</td>
<td>JOHN DOE</td>
<td>01/11/02 11:41</td>
<td>5.2.5</td>
</tr>
<tr>
<td>FDT00058</td>
<td>SEGMENTE IMS</td>
<td>G SAMPLEDB</td>
<td>TEST</td>
<td>MMM 523</td>
<td>JOHN DOE</td>
<td>01/11/02 11:41</td>
<td>5.2.5</td>
</tr>
<tr>
<td>FDT00059</td>
<td>SEGMENTF IMS</td>
<td>G SAMPLEDB</td>
<td>TEST</td>
<td>MMM 523</td>
<td>JOHN DOE</td>
<td>01/11/02 11:41</td>
<td>5.2.5</td>
</tr>
</tbody>
</table>

The following fields appear on this report:

**company-name**

Identifies the name of the company as entered during the installation process.

**FDT ID**

Identifies the control ID for the FDT. The numeric portion of the ID is the FDT number, and is used in any error message.

**FDT NAME**

Identifies the load module name for the FDT.
**INTERFACE**
Identifies the interface for which the FDT was generated. The following values are available:
- IMS
- MVS

**RB**
Identifies the recorded by (RB) column. The following values are available:
- G
  Indicates that the FDT was generated by the Interactive Dialog.

**STRUCTURE**
Provides the following information about the file or database:

**NAME**
Identifies the data set name or the DBD and segment names.

**TITLE**
Identifies the title of the entry.

**FDT CREATION DATA**
Provides the following information:

**CREATED BY**
Identifies the name of the person who generated the FDT.

**DATE**
Identifies the date the FDT was generated.

**TIME**
Identifies the time the FDT was generated.

**REL**
Identifies the release of the Interactive Dialog that was used to generate the FDT.
Authorized User List Report

This report lists all persons authorized to use the Interactive Dialog Control File.

<table>
<thead>
<tr>
<th>TSO ID</th>
<th>USER'S FULL NAME</th>
<th>NICK NAME</th>
<th>IMS</th>
<th>MVS</th>
<th>IDMS</th>
<th>DB2</th>
<th>FDT</th>
<th>SEC</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISPAPT1</td>
<td>DOE, JOHN</td>
<td>JOHN</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>ISPDP1</td>
<td>DOE, JANE</td>
<td>JANE</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>ISPAM1</td>
<td>SMITH, JAMES</td>
<td>JAMES</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>ISPBA1</td>
<td>SMITH, JOAN</td>
<td>JOAN</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>ISPKTS1</td>
<td>JONES, DAVID</td>
<td>DAVID</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>ISPWAS1</td>
<td>TAYLOR, ROBERTA</td>
<td>ROBERTA</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

The following fields appear on this report:

**company-name**

Identifies the name of the company as entered during the installation process.

**TSO ID**

Identifies the TSO ID of the user.

**Note:** The Interactive Dialog is an ISPF dialog, and therefore it uses the TSO ID as the primary user identifier.

**USER'S FULL NAME**

Identifies the name of the user that is recorded on the Interactive Dialog Control File to identify the user.

**NICK NAME**

Identifies the user name used in the various HELP and TUTORIAL messages and panels.

**DIALOG USAGE**

Provides information about the various Dialog services that are available to the user:

- **IMS**—CA Compress Data Compression for IMS for IMS services
- **MVS**—CA Compress Data Compression for IMS for MVS services
- **IDMS**—CA Compress Data Compression for IMS for IDMS (not supported)
- **DB2**—CA Compress Data Compression for IMS for DB2 services (not supported)
- **FDT**—FDT regenerate and delete services
- **SEC**—Modify installation control parameters, perform user maintenance, and set default data set names
IMS Reports

The following are IMS-specific CA Compress Data Compression for IMS reports:

- **Index List Report** (see page 58)
- **Stored Data Report** (see page 59)
- **Extracted Data Sample Report** (see page 70)
- **Byte Distribution Analysis Report** (see page 73)
- **Test Compression Report** (see page 76)

**Index List Report**

This report lists all of the databases that are currently defined to the Interactive Dialog Control File.

<table>
<thead>
<tr>
<th>DBD NAME</th>
<th>DATABASE NAME</th>
<th>GEN DATE</th>
<th>TIME</th>
<th>REL</th>
<th>ANALYZED BY</th>
<th>DATE</th>
<th>TIME</th>
<th>REL</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAMPLEDB</td>
<td>MMM TEST 523</td>
<td>01/05/02</td>
<td>10.29</td>
<td>5.2.5</td>
<td>JOHN DOE</td>
<td>01/11/02</td>
<td>11:44</td>
<td>5.2.5</td>
</tr>
<tr>
<td>TRAINDB</td>
<td>TEST</td>
<td>01/22/02</td>
<td>15.01</td>
<td>5.2.5</td>
<td>JANE DOE</td>
<td>01/22/02</td>
<td>15:19</td>
<td>5.2.5</td>
</tr>
</tbody>
</table>

The following fields appear on this report:

- **company-name**
  
  Identifies the name of the company as entered during the installation process.

- **DBD NAME**
  
  Identifies the name of the DBD.

- **DATA BASE NAME**
  
  Identifies the name of the database.
IMS DATA

Provides information about the DBD load module and the IMS release used:

GEN DATE
Identifies the date the DBD load module was assembled.

TIME
Identifies the time the DBD load module was assembled.

REL
Identifies the release of the IMS used to assemble the DBD.

CA-COMPRESS for IMS DATA

Provides information about the database analysis:

ANALYZED BY
Identifies the name of the person that performed the database structure analysis.

DATE
Identifies the date the database structure was analyzed.

TIME
Identifies the time the database structure was analyzed.

REL
Identifies the release of the Interactive Dialog that was used to analyze the database.

Stored Data Report

The IMS Stored Data report for a database displays most of the information that has been recorded about an IMS database. The report consists of the following basic parts:

- [Page Heading and Database Information section](#) (see page 61)
- [Segment section](#) (see page 62)
- [Data Set Information section](#) (see page 66)
- [JCL Generation Information section](#) (see page 68)
**Stored Data Report 1**

The Stored Data Report includes the following information:

- **For the entire database:**
  - Organization type (HDAM, HIDAM, HISAM, DEDB)
  - Number of data set groups
  - Number of levels
  - Number of segments
  - Number of second indexes
  - Names of other associated DBDs

- **For each segment of the database:**
  - Segment name
  - Segment level
  - Segment hierarchical position
  - Minimum and maximum length
  - Prefix length
  - Key field
  - Key location
  - Maximum compressible length
  - Current RDL statement
  - FDT data
  - Data extract parameters
  - Byte distribution analysis
  - Best compression test results

- **Control information regarding the database:**
  - DDNAME
  - Device type
– DBD name
– Data set name

Basic testing information:
– Description of JCL generated for this database
– Date and time of byte distribution analysis
– Date and time of compression testing

Page Heading and Database Information Section

This section shows general information about the database.

<table>
<thead>
<tr>
<th>STORED DATA REPORT FOR IMS DBD -- HIDAMDB -- HIDAM DATABASE</th>
</tr>
</thead>
<tbody>
<tr>
<td>company-name: AS OF 01/25/02 AT 16:03 PAGE: 1</td>
</tr>
</tbody>
</table>

DATABASE INFORMATION:......................................................................
* ANALYZED ON 06/03/02 AT 13:58 BY JOHN DOE                        FROM SHRINK.IMSEXPDDBLIB *
* DBD WAS ASSEMBLED ON 06/03/02 AT 11:52 WITH RELEASE 5.2.5 OF IMS *
* ORGANIZATION IS HIDAM DATABASE CONTAINS:                          *
* 1 DATA SET GROUPS                                                *
* 3 LEVELS                                                        *
* 0 2ND INDEXES                                                   *
* 1 ASSOCIATED DBDS -- HIDAMNDX                                    *
* PRODUCTION DBD WAS GENERATED ON 06/03/02 AT 14:06 BY JOHN DOE     *
****************************************************************************************************************************

The following fields appear on this report:

class-name

Identifies the name of the company as entered during the installation process.

ANALYZED

Provides information about the database analysis:

- ON—The date database structure was analyzed.
- AT—The time the database structure was analyzed.
- BY—The name of the person that performed the analysis.
- FROM—Data set name of the DBD library from which analysis was performed.

DBD WAS ASSEMBLED

Provides information about the assembly of the DBD:

- ON—The date and time the database DBD was assembled.
- AT—The date and time the database DBD was assembled.
- WITH RELEASE—The release of IMS used to perform the assembly.

ORGANIZATION IS

Identifies the database organization.
DATABASE CONTAINS

Provides information about the content of the database:

- DATA SET GROUPS—The number of Data Set Groups (DSG) in the database.
- LEVELS—The number of levels in the database.
- SEGMENTS—The number of segments in the database.
- 2ND INDEXES—The number of secondary indexes that point to the database.
- ASSOCIATED DBDS—The number of other DBDs that are associated with this database. This includes the primary and secondary indexes, any logically related databases and their indexes, and the DBD names of all the associated databases.

PRODUCTION DBD WAS GENERATED

Provides the CA Compress Data Compression for IMS DBD generation information, if the DBD has been processed through the IMS implementation process.

- ON—The date DBD was generated.
- AT—The time the DBD was generated.
- BY—The name of the person that generated the DBD.

Segment Section

The Segment section contains information about a specific segment that consists of a root segment and a dependent segment. The two segment reports are nearly identical, except that they apply to the parent and child respectively.

Root segment

```
SEGMENT 1 IS CTROOTSG ON LEVEL 1 AND IS A CHILD OF ************************************************************
* FORMAT(F)   MIN LENGTH(0)    SC;RMH(40)   (GC;RMH(10) SECONDARY INDEX SOURCE(NO )) *
* KEY FIELD (ROOTKEY) STARTS AT(1) LENGTH(7) OCCURRENCES(3) *
* MAXIMUM COMPR LNGTH(33) THE CURRENT RDL SOURCE STATEMENT IS:  *
* +-----------------------------------------------*
* |N7,   | |I |CIF33.|
* +-----------------------------------------------*
* "FDT" NAME(CTROOTSG) SHARE(NO ) MINIMUM COMPRESSED SIZE(0) COMPRESS STATS(YES) PADDING -- LENGTH(0) TYPE(FIXED ) *
* "FDT" IS NOT SHARED *
* SEGMENT WAS SELECTED FOR EXPRESS COMPRESSION ON 06/03/02 AT 14:05 BY JOHN DOE *
* DATA EXTRACT PARAMETERS ARE -- Bypass(0) EXTRACT (ALL) SKIP(0) *
* DATA SAMPLE CONSISTS OF 3 SEGMENTS EXTRACTED ON 06/03/02 AT 14:04 *
* INPUT LENGTHS SMALLEST = 40 AVERAGE = 40 LARGEST = 40 *
* SAMPLE LENGTHS: SMALLEST = 40 AVERAGE = 40 LARGEST = 40 *
* BYTE DISTRIBUTION ANALYSIS WAS COMPLETED ON 06/03/02 AT 14:04 *
* THE LAST COMPRESSION TEST WAS RUN ON 06/03/02 AT 14:04 *
* THE BEST COMPRESSION WAS ACHIEVED BY TEST NBR 1 WHEN THE AVERAGE COMPRESSED SEGMENT SIZE WAS 46 BYTES *
* BEST TEST LENGTHS: SMALLEST = 35 AVERAGE = 36 LARGEST = 36 *
* LAST HUFF LENGTHS: SMALLEST = 35 AVERAGE = 36 LARGEST = 36 *
* LAST EXPR LENGTHS: SMALLEST = 35 AVERAGE = 36 LARGEST = 40 *
* ****************************************************************************************************************************
```
The following fields appear on this report:

### SEGMENT
Identifies the hierarchical position of the segment within the database.

### IS
Identifies the name of the segment.

### ON LEVEL
Identifies the level of the database that contains the segment.

### AND IS A CHILD OF
Identifies the parent segment. This field is blank for the root segment.

### FORMAT
Identifies the format of the segment:
- Fixed
- Variable

### MIN LENG
Identifies the minimum length of the segment. The minimum length is 0 for fixed-length segments.

### MAX LENG
Identifies the maximum length of the segment.

### PFX LENG
Identifies the length of the IMS prefix for the segment.
SECONDARY INDEX SOURCE
Indicates whether the segment is the source of a secondary index.

KEY FIELD
Identifies the name of the key field of the segment. This field is blank for nonkeyed segments.

STARTS AT
Identifies the starting position of the segment key. This field is 0 for nonkeyed segments.

LENGTH
Identifies the length of the segment key. This field is 0 for nonkeyed segments.

OCCURRENCES
Identifies the number of occurrences of the segment within the database. This number is based on the average number of this segment per root segment read during the extraction of the data sample.

Note: Supply the number of root segments in the database after the data sample has been extracted but before trying to test compress the data sample for the database.

MAXIMUM COMPRESSIBLE LENGTH
Identifies the number of compressible bytes of data in the segment.

THE CURRENT RDL SOURCE STATEMENT IS
Identifies the current RDL for the segment.

After the implementation process has been performed, a message appears indicating that the FDT has been generated if full service compression was selected or if Express compression was selected, the message will indicate the use of Express.

"FDT" NAME
Identifies the current FDT parameters for the segment.

SHARE
Indicates whether the FDT is shared.

MINIMUM COMPRESSED SIZE
Identifies the minimum size of the compressed segment (MINSIZE value).

COMPRESS STATS
Indicates whether compression statistics are to be generated.

PADDING
Identifies the length and type of padding used.
LENGTH
Identifies the length of padding used.

TYPE
Identifies the type of padding used.

"FDT" IS NOT SHARED
Indicates whether the FDT is shared.

"FDT" GENERATED
Provides information about the generation of the FDT. For root segments, this line contains data extraction parameters; for dependent segments, this line is omitted.
- ON—The date the FDT was generated.
- AT—The time the FDT was generated.
- BY—The name of the person that performed the generation.

SEGMENT WAS SELECTED
Provides information about the selection of the segment.
- FOR—The type of compression used for the segment
- ON—The date the segment was selected
- AT—The time the segment was selected
- BY—The name of the person that performed the selection

DATA EXTRACTION PARAMETERS ARE
Provides the data extraction parameters used to create the sample. This line is omitted for dependent segments.
- BYPASS—The number of records to bypass before extracting the first record.
- EXTRACT—The number of records to extract.
- SKIP—The number of records to skip between each extracted record. The SKIP parameter spreads the extraction across the database.

DATA SAMPLE CONSISTS OF
Identifies the number of segments in the sample after the extraction.
- ON—The date the sample was extracted.
- AT—The time the sample was extracted.

INPUT LENGTHS
Identifies the lengths of the smallest, the average, and the largest segments read.
SAMPLE LENGTHS

Identifies the lengths of the smallest, the average, and the largest segments extracted.

BYTE DISTRIBUTION ANALYSIS WAS COMPLETED

Identifies the date and time that the byte distribution was run. This field is blank if the Byte Distribution Analysis has not yet been run.

LAST COMPRESSION TEST WAS RUN

Identifies the date and time of the most recent compression test.

BEST COMPRESSION WAS ACHIEVED BY TEST NBR

Identifies the number of the test that obtained the best compression for the segment. This field is blank if the Byte Distribution Analysis has not yet been run.

WHEN THE AVERAGE COMPRESSED SEGMENT SIZE WAS

Identifies the average size in bytes of the segment that achieved the best compression.

BEST TEST LENGTHS

Identifies the smallest, average, and largest lengths that obtained the best test compression for the segment.

LAST HUFF LENGTHS

Identifies the smallest, average, and largest lengths of the most recent Huffman compression test for the segment.

LAST EXPR LENGTHS

Identifies the smallest, average, and largest lengths of the most recent Express compression test for the segment.

Data Set Information Section

The Data Set Information section provides information about the data sets that are required for IMS to process the database.
The following fields appear on this report:

**DDNAME**
Identifies the ddname for the data set.

**DEV**
Identifies the type of the device on which the data set is stored.

**DBD**
Identifies the name of the DBD in which the ddname was found.

**DSN**
Identifies the name of the data set.

**TRACKS**
Identifies the number of tracks allocated to the data set.

**SEGS**
Identifies the number of segments found in the data sets that correspond to the DSGs in the database.

**’DATA SET NAME’ ENTERED**
Provides event information about the data set name:
- **ON**—The date the data set name was entered.
- **AT**—The time the data set name was entered.
- **BY**—The name of the person that entered the data set name.

**DSG LENGTHS**
Identifies the length of the smallest, average, and largest DSG segments.

**SAMPLE LENGTHS**
Identifies the length of the smallest, average, and largest sample segments.

**HUFFMAN LENGTHS**
Identifies the length of the smallest, average, and largest segments using Huffman.

**EXPRESS LENGTHS**
Identifies the length of the smallest, average, and largest segments using Express.

**DDNAME**
Identifies the DDNAME for the index.

**DEV**
Identifies the type of device on which the index is stored.
DBD
Identifies the name of the DBD in which the ddname for the index was found.

DSN
Identifies the name of the index.

TRACKS
Identifies the number of tracks allocated to the index.

SEGS
Identifies the number of segments found in the index that correspond to the DSGs in the database.

‘DATA SET NAME’ ENTERED
Provides the event information about the index name:
- ON—The date the index name was entered.
- AT—The time the index name was entered.
- BY—The name of the person that entered the index name.

JCL Generation Information Section

The JCL Generation Information section provides the following information about what has transpired during the compression testing for the database:

SAMPLE DATA EXTRACTION JCL WAS GENERATED
Provides information about JCL extraction:
- ON—The date the extract JCL was generated.
- AT—The time the extract JCL was generated.
- BY—The name of the person who generated the extract JCL.

PSBLIB
Identifies the PSBLIB data set name used during the data sample extraction.

PSB NAME
Identifies the name of the PSB.

PCB NUMBER
Identifies the number of the PCB.

ACBLIB
Identifies the ACBLIB data set name used during the data sample extraction.

PSB NAME
Identifies the PSB name for the ACB.
PCB NUMBER
Identifies the PCB number for the ACB.

RESLIB
Identifies the RESLIB data set name used during the data sample extraction.

DBDLIB
Identifies the DBDLIB data set name used during the data sample extraction.

EXTRACT
Identifies the name of the data set to which the extracted data was written.

THE SAMPLE DATA FOR THE DATA BASE WAS EXTRACTED
Provides information about the extraction of the sample data:
- **ON**—The date the sample data was extracted.
- **AT**—The time the sample data was extracted.

BYTE DISTRIBUTION ANALYSIS WAS GENERATED
Provides information about the byte distribution analysis JCL generation:
- **ON**—The date the byte distribution JCL was generated.
- **AT**—The time the byte distribution JCL was generated.
- **BY**—The name of the person who generated the byte distribution JCL.

BYTE DISTRIBUTION ANALYSIS WAS LAST RUN
Provides information about the most recent byte distribution analysis JCL generation:
- **ON**—The date of the most recent byte distribution run.
- **AT**—The time of the most recent byte distribution run.

COMPRESSION TESTING JCL WAS GENERATED
Provides information about the compression testing JCL generation:
- **ON**—The date the compression testing JCL was generated.
- **AT**—The time the compression testing JCL was generated.
- **BY**—The name of the person who generated the compression testing JCL.
COMPRESSION TESTING FOR THE DATA BASE WAS LAST RUN

Provides information about the most recent test compression run:

- **ON**—The date of the most recent test compression run.
- **AT**—The time of the most recent test compression run.

**FOR COMPRESSION TEST NUMBER**

Identifies the number of the last test compression run.

**Extracted Data Sample Report**

The Extracted Data Sample Report provides basic information about the data sample used to base the compression of the segment. The report consists of the following sections:

- **Heading section** (see page 70)
- **Extraction data section** (see page 71)

**Extracted Data Sample Report Heading Section**

The heading section identifies the database, the installation, and the date and the time the report was generated.

**********************************************************************************
* EXTRACTED DATA SAMPLE REPORT FOR IMS DATABASE -- SAMPLEDB -- MMM TEST 523 *
* company-name PAGE: 1 *
* EXTRACT WAS FROM A HDAM DATABASE USING PCB 1 IN PSB SAMPLPSB *
* EXTRACT DATA WRITTEN TO DATA SET SHRINK.EXTRACT.FILE *
**********************************************************************************

The following fields appear on this report:

**EXTRACTED SAMPLE REPORT FOR IMS DATA BASE**

Provides the title of the report, the name of the data set, and test number.

- **company-name**
  
  Identifies the name of the company as entered during the installation process.

**AS OF**

Identifies the date the report was generated.

**AT**

Identifies the time the report was generated.

**EXTRACTED FROM A**

Identifies the type of data set from which the data was extracted.
USING

Identifies the PCB used during the data extraction.

IN PSB

Identifies the PSB used during the data extraction.

EXTRACT DATA WRITTEN TO

Identifies the name of the data set that contains the extracted data sample.

Data Section

The data section of the extract report provides information about database records and segments for IMS databases.

<table>
<thead>
<tr>
<th>POS</th>
<th>NAME</th>
<th>FMT</th>
<th>LEN</th>
<th>MIN</th>
<th>MAX</th>
<th>X1000</th>
<th>X1000</th>
<th>X1</th>
<th>READ</th>
<th>BYPASS</th>
<th>EXTRACTED</th>
<th>EXTRACTED</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SEGMENTA</td>
<td>F</td>
<td>18</td>
<td></td>
<td></td>
<td>100</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>360</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>SEGMENTB</td>
<td>V</td>
<td>10</td>
<td>100</td>
<td></td>
<td>312</td>
<td>3</td>
<td>9</td>
<td>0</td>
<td>9</td>
<td>1,312</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>SEGMENTC</td>
<td>F</td>
<td>6</td>
<td>80</td>
<td></td>
<td></td>
<td></td>
<td>9</td>
<td>0</td>
<td>9</td>
<td>720</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>SEGMENTD</td>
<td>V</td>
<td>6</td>
<td>50</td>
<td>200</td>
<td></td>
<td></td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>576</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>SEGMENTE</td>
<td>V</td>
<td>10</td>
<td>100</td>
<td>212</td>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>192</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>SEGMENTF</td>
<td>F</td>
<td>10</td>
<td>80</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>80</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>SEGMENTG</td>
<td>F</td>
<td>6</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>260</td>
<td></td>
</tr>
</tbody>
</table>

EXTRACT RUN TOTALS: 28           0           28       3,380

THE AVERAGE LENGTH OF THE RECORDS READ WAS 1,127 BYTES.
THE AVERAGE LENGTH OF THE RECORDS EXTRACTED WAS 1,127 BYTES.
THIS IS A VARIATION IN AVERAGE RECORD LENGTHS OF 0.0000 PERCENT.
THE PERCENT OF VARIATION BETWEEN THE AVG DATABASE RECORD LENGTHS CAN CAUSE A LIKE ERROR IN THE DASD SPACE NEEDED FOR THE DATABASE

The following fields appear on this report:

SEGMENT DATA

Provides the characteristics of the uncompressed data:

HIER POS

Identifies the hierarchical position of the segment within an IMS database.

SEGMENT NAME

Identifies the name of the segment in an IMS database.

SEG FMT

Identifies the format of the segment (Fixed or Variable).

PFX LEN

Identifies the length of the IMS segment prefix in bytes.
SEGMENT LENGTH MIN
Identifies the minimum segment length. This field applies only to variable-length IMS database segments.

SEGMENT LENGTH MAX
The maximum length of the segment.

EXTRACT CNTLS
Provides the criteria for extracting the data sample in terms of database records or root segments.
- BYPASS—The number of records bypassed before extracting the first record.
- COUNT—The number of records extracted.
- SKIP—The number of records skipped between each extracted record.

DATA SAMPLE EXTRACT INFORMATION
Provides information about the number of segments processed during the data extraction.

Note: The reading of the database terminates as soon as the desired number of records have been extracted.

SEGMENTS READ
Identifies the total number of segments read during the data extraction.

SEGMENTS BYPASSED
Identifies the number of segments that were bypassed or skipped during the data extraction.

SEGMENTS EXTRACTED
Identifies the number of segments that were written to the extracted data sample file.

BYTES EXTRACTED
Identifies the number of bytes of data contained in the extracted data sample. If the average length of the segments that is read are not within 0.01 percent of the average length of the segments extracted, a second line shows the average lengths and the percent of variation.

EXTRACT RUN TOTALS
Identifies the totals for all segments for the preceding four extraction items.

THE AVERAGE LENGTH OF THE RECORDS READ WAS
Identifies the average length of all the records read.
THE AVERAGE LENGTH OF THE RECORDS EXTRACTED WAS
Identifies the average length of only the records extracted.

THIS IS A VARIATION IN AVERAGE RECORD LENGTHS OF
Identifies the variance between the average length of records read and records extracted.

All of the DASD space and cost calculations make the assumption that the data sample is representative of the entire database. Average record length information helps you determine whether the extracted sample is representative of the database as a whole. If the lengths of the database records read and of the database records extracted are not close to each other, modify the data extraction parameters and extract another data sample.

Byte Distribution Analysis Report

The Byte Distribution Analysis Report is produced when the Byte Distribution Analysis program (SFEPBDA) is executed. Execute this program for every extracted data sample before the Test Compression program. If you have modified any RDL for a segment, execute the Byte Distribution Analysis program again.

Byte Distribution Analysis Report Heading Section

The heading section identifies the database, the installation, and the date and time the report was generated.

The following fields appear on this report:

BYTE DISTRIBUTION ANALYSIS REPORT FOR IMS DBD
Provides the title of the report, and the name of the data set.

company-name
Identifies the name of the company as entered during the installation process.

AS OF
Identifies the date the report was generated.
AT
Identifies the time the report was generated.

DATA SAMPLE EXTRACT INFORMATION
Provides information about the data set from which the data was extracted:

ORGANIZATION IS
Identifies the organization type of the data set

DBD
Identifies the name of the DBD.

DSNAME OF EXTRACT DATA SET
Identifies the name of the data set that contains the extracted data.

SOURCE OF DATA SAMPLE
Identifies the name of the data set that was the source of the extracted data.

DATE
Identifies the date that the data was extracted.

TIME
Identifies the time that the data was extracted.

Detail Section
The detail section of the report appears once for each segment in an IMS database.
The following fields appear on this report:

**EXTRACTED COMPONENT INFORMATION FOR SEGMENT**

Identifies the number and name of the segment.

**SAMPLE SIZE**

Provides information about the size of the sample.

**NUM OF SEGS**

Identifies the number of segments in the extracted data sample.

**NUM OF BYTES**

Identifies the number of bytes of data in the extracted sample.

**HUFFMAN FIELDS**

Identifies the number of C1 (# C1 FLDS), C2 (# C2 FLDS), and C3 (# C3 FLDS) fields in the RDL for the segment.

**DIST CALCULATED**

Provides information about byte distribution analysis. This information is calculated only when the program finds that there are no BDA records in the Interactive Dialog Control File for a segment. The BDA records are erased any time you execute the Data Sample Extract program or modify the RDL for a segment.

- **DATE**—The date the byte distribution analysis was performed.
- **TIME**—The time the byte distribution analysis was performed.
FDT DATA
Provides information about the FDT parameters for the specified segment area.

SOURCE OF RDL
Identifies the RDL source. When SHARE is NO, this field shows the name of the current database and segment. When SHARE is YES or SFS, this field shows the name of the specified database and segment.

SHARE
Identifies whether the segment shares an FDT:
- NO—The segment uses its own RDL for data compression.
- SFS—The segment uses the FDT of another segment that has the same field structure (SFS). The base segment for the FDT has a SHARE value of NO.
- YES—Two or more segments with the same segment format (fixed or variable) and the same key length use the same FDT.

Note: When SHARE YES is used, the only RDL permitted for the compressible data is C1VER. When SOURCE OF RDL field refers to the current segment, then this is the base segment used to generate the FDT. When the SOURCE OF RDL field refers to another segment, then the current segment cannot be used to generate an FDT but uses the FDT generated for the specified segment.

PAD
Identifies the amount of free space, if any, to be left after the segment has been compressed.

MIN SIZE
Identifies the minimum size of the compressed segment.

EXTRACT RDL
Identifies the RDL used to calculate the byte distribution.

Test Compression Report
The Test Compression Report provides the information needed to determine for each segment in the database whether that segment should be compressed, and, if so, what type of compression ratio can be expected from each type of compression. This report consists of the following sections:
- Heading (see page 77)
- Compression Test Information (see page 78)
- DASD Utilization (see page 81)
Heading Section

The heading section identifies the database, the installation, and the date and time the report was generated.

The following fields appear on this report:

**COMPRESSION TEST REPORT FOR IMS DBD**

Provides the title of the report and the name of the data set.

**TEST NUMBER**

Identifies the number of the test.

**company-name**

Identifies the name of the company as entered during the installation process.

**AS OF**

Identifies the date the report was generated.

**AT**

Identifies the time the report was generated.
DATA SAMPLE EXTRACT INFORMATION

Provides information about the data set from which the data was extracted:

ORGANIZATION IS

Identifies the organization type of the data set.

DBD

Identifies the name of the DBD.

DSNAME OF EXTRACT DATA SET

Identifies the name of the data set that contains the extracted data.

SOURCE OF DATA SAMPLE

Identifies the name of the data set that was the source of the extracted data.

DATE

Identifies the date that the data was extracted.

TIME

Identifies the date that the data was extracted.

SAMPLE SIZE

Identifies the number of segments in the sample.

Compression Test Information Section

The Compression Test Information section is reported for each segment type that was extracted into the data sample. It indicates how effective the specified RDL and FDT control parameters were in compressing the segment.

| * COMPRESSION TEST INFORMATION FOR SEGMENT A SEGMENTA *************************************************************** |
| * NUM OF SEGS IN SAMPLE PREFIX KEY IC DATA IN SAMPLE LENGTHS TOTAL AVERAGE LENGTHS IN SAMPLE BYTES SAVED UCMPRD CMPRSD W/PREFIX KEY+DATA DATA TEST * |
| * 3 18 10 3 90 300 100 |
| * HUFFMAN COMPRESSION -- 168 44 47.5 56.0 62.3 1 * |
| * EXPRESS COMPRESSION -- 121 60 33.9 40.0 44.5 * |
| * FDT DATA: SOURCE OF RDL(SAMPLEDB/SEGMENTA) PADDING=(N,000) MINSIZE=000 |
| * EXTRACT RDL | N10, C1F90. | |
| * ************************************************************************************************************************** |

The following fields appear on this report:

COMPRESSION TEST INFORMATION

Provides the title of the report.

FOR SEGMENT

Identifies the segment.
NUM OF SEGS IN SAMPLE
Identifies the number of segments that comprised the data sample.

SEGMENT LENGTHS
Provides information about the length of the segment:
PREFIX
Identifies the length of the IMS segment prefix.
KEY
Identifies the number of bytes in the segment key. For an unkeyed segment, this field is blank.
IC
A control field that only appears in compressed segments. This field is always 3.
DATA
Identifies the number of bytes of compressible data in the segment.

NUM OF BYTES IN SAMPLE
Identifies the size of the data sample expressed in bytes of data.

TOTAL BYTES SAVED
Identifies the number of bytes saved by the specified compression process.

AVERAGE LENGTHS
Provides information about the average lengths:
UNCPRSD
Identifies the average length of the uncompressed segment.
CMPRSD
Identifies the average length of the segment after being compressed by the specified process.
PERCENT OF COMPRESSION

Provides information about the percent of compression achieved for the following types:

**W/PREFIX**

Identifies the true percent of compression that can be expected for the segment. This value is based on the total IMS length of the segment and includes the IMS prefix length and the compressed segment length.

**KEY+DATA**

Identifies the percent of compression that can be expected for the segment. This value includes the noncompressible key, if any, and compressible data area.

**DATA**

Identifies the possible compression of the compressible portion of the segment. For Huffman compression, this value reflects how effective the RDL was in compressing the segment. Changing the RDL changes the Huffman compression percent, but not the Express compression.

**BEST TEST**

Identifies the test number which gave the smallest average compressed segment length for Huffman compression. The number is incremented each run where the average length is smaller than or equal to the previous smallest average segment length.

**HUFFMAN COMPRESSION**

Identifies the expected compression of the segment using the RDL and FDT shown.

**EXPRESS COMPRESSION**

Identifies the expected compression of the segment using Express.

**FDT DATA**

Provides information about FDT data. This field applies only to Huffman compression.

**SOURCE OF RDL**

This field has meaning only if the segment uses RDL and byte distribution data of another segment for compression.

**Padding**

Identifies the type and amount of padding that has been applied to the compressed segment.
MINSIZE
Identifies the minimum size of the compressed segment that will be written.

EXTRACT RDL
(Huffman compression) The RDL information only applies to Huffman compression.

DASD Utilization Section

The DASD Utilization section describes the current characteristics of each data set in a database. ISAM/OSAM is not supported. This section varies depending on the organization of the database and whether the access method is VSAM or OSAM.

The following fields are common for all database types:

DATASET GROUP TOTALS
Provides total information about data sets:

THE LENGTH OF THE AVG UNCOMPRESSED DB RECORD IS
Identifies the average uncompressed record length for the database.

THE LENGTH OF THE AVG COMPRESSED DB RECORD IS
Identifies the average compressed record length for the database.

DATASET
Identifies the name of the data set.

PRIMARY DATASET
(VSAM-HISAM) Identifies the name of the primary data set.

OVERFLOW DATASET
(VSAM-HISAM) Identifies the name of the overflow data set.

DDNAME
Identifies the type and DDNAME of the data set.

RAPS PER BLK
Identifies the number of root anchor points (RAPs) in each block.

RAP BLOCKS
(OSAM) Identifies the number of RAP blocks.

RAP BYTES
(OSAM) Identifies the number of RAP bytes.

DEVICE
Identifies the type of device on which the data set is stored.
RECD LENGTH
(OSAM) Identifies the length of the physical database record.

RECDS/TRACK
(OSAM, VSAM-HISAM) Identifies the number of physical records per track.

FREE BLOCK PCT
(OSAM) Identifies the free block percentage factor for the data set.

FREE BLK FREQ
(OSAM) Identifies the free block frequency factor for the data set.

DCB IS
(OSAM) Provides information about the DCB of the data set:
RECFM
Identifies the record format of the data set.
LRECL
Identifies the logical record length of the data set.
BLKSIZE
Identifies the block size of the data set.

CI SIZE
(VSAM) Identifies the size of the control interval (CI) for the data set.

CIs/TRACK
(VSAM) Identifies the number of CIs per track for the data set.

MAX LRECL
(VSAM-HISAM) Identifies the maximum LRECL size of the specified data set.

RECDS/CI
(VSAM-HISAM) Identifies the number of records per CI for the specified data set.

CURRENT DASD UTILIZATION IS
(VSAM) Identifies the percent of DASD that is currently used by the specified data set.

CURRENT FREE SPACE PCT
(VSAM-HIDAM) Identifies the percent of free space currently specified.

CURRENT FREE CI FREQ
(VSAM-HIDAM) Identifies the frequency of free CIs currently specified.
TRACKS

Provides information about the current DASD allocation, current DASD utilization, DASD utilization and savings based on current data set characteristics and allocations. For databases with multiple DSGs, a summary information for the preceding data sets is provided for the total database at the bottom of the report.

CURR ALLOCATED

Identifies the number of tracks or blocks currently allocated for the data set or database.

SEGMENTS

Identifies the number of segments.

CURR USED

Identifies the number of tracks or blocks currently used for the data set or database.

UNCOMPRESSED DATA BYTES

Identifies the number of uncompressed data bytes for the data set or database.

COMPRESS USES

Identifies the number of tracks or blocks used by CA Compress Data Compression for IMS for the data set or database.

COMPRESSED DATA BYTES

Identifies the number of compressed data bytes for the data set or database.

COMPRESS SAVES

Identifies the number of tracks or blocks saved by CA Compress Data Compression for IMS for the data set or database.

COMPRESS -- BYTES SAVED

Identifies the number of bytes saved by CA Compress Data Compression for IMS for the data set or database.
Multiple DSG Database—OSAM

DATASET GROUP TOTALS:

THE LENGTH OF THE AVG UNCOMPRESSED DB RECORD IS 8,819 BYTES INCLUDING SEGMENT PREFIXES.
THE LENGTH OF THE AVG COMPRESSED DB RECORD IS 5,968 BYTES INCLUDING SEGMENT PREFIXES.

1. DATASET: INFVSAM1.SHRINK.TESTDB.INDD031

<table>
<thead>
<tr>
<th>DONAME: PRIMARY</th>
<th>INDD031</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAPS PER BLK</td>
<td>4</td>
</tr>
<tr>
<td>RAP BLOCKS</td>
<td>10</td>
</tr>
<tr>
<td>RAP BYTES</td>
<td>1,020</td>
</tr>
<tr>
<td>DEVICE</td>
<td>3380</td>
</tr>
<tr>
<td>RECD LENGTH</td>
<td>1,690</td>
</tr>
<tr>
<td>RECDS/TRACK</td>
<td>21</td>
</tr>
<tr>
<td>FREE BLOCK PCT</td>
<td>0</td>
</tr>
<tr>
<td>FREE BLK FREQ</td>
<td>0</td>
</tr>
<tr>
<td>DCB IS:</td>
<td>RECFM</td>
</tr>
<tr>
<td>RECFM:</td>
<td>FBS</td>
</tr>
<tr>
<td>LRECL:</td>
<td>1,690</td>
</tr>
<tr>
<td>BLKSIZE:</td>
<td>1,690</td>
</tr>
<tr>
<td></td>
<td>TRACKS</td>
</tr>
<tr>
<td>CURR ALLOCATED</td>
<td>--</td>
</tr>
<tr>
<td>SEGMENTS</td>
<td>30</td>
</tr>
<tr>
<td>UNCOMPRESSED DATA BYTES</td>
<td>-- 1,379</td>
</tr>
<tr>
<td>COMPRESS USES</td>
<td>--</td>
</tr>
<tr>
<td>COMPRESSED DATA BYTES</td>
<td>-- 28,211</td>
</tr>
<tr>
<td>COMPRESS SAVES</td>
<td>--</td>
</tr>
<tr>
<td>COMPRESS -- BYTES SAVED</td>
<td>-- 16,114</td>
</tr>
</tbody>
</table>

2. DATASET: INFVSAM1.SHRINK.TESTDB.INDD032

<table>
<thead>
<tr>
<th>DONAME: PRIMARY</th>
<th>INDD032</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAPS PER BLK</td>
<td>4</td>
</tr>
<tr>
<td>RAP BLOCKS</td>
<td>10</td>
</tr>
<tr>
<td>RAP BYTES</td>
<td>1,020</td>
</tr>
<tr>
<td>DEVICE</td>
<td>3380</td>
</tr>
<tr>
<td>RECD LENGTH</td>
<td>1,690</td>
</tr>
<tr>
<td>RECDS/TRACK</td>
<td>21</td>
</tr>
<tr>
<td>FREE BLOCK PCT</td>
<td>0</td>
</tr>
<tr>
<td>FREE BLK FREQ</td>
<td>0</td>
</tr>
<tr>
<td>DCB IS:</td>
<td>RECFM</td>
</tr>
<tr>
<td>RECFM:</td>
<td>FBS</td>
</tr>
<tr>
<td>LRECL:</td>
<td>1,690</td>
</tr>
<tr>
<td>BLKSIZE:</td>
<td>1,690</td>
</tr>
<tr>
<td></td>
<td>TRACKS</td>
</tr>
<tr>
<td>CURR ALLOCATED</td>
<td>--</td>
</tr>
<tr>
<td>SEGMENTS</td>
<td>30</td>
</tr>
<tr>
<td>UNCOMPRESSED DATA BYTES</td>
<td>-- 8,668</td>
</tr>
<tr>
<td>COMPRESS USES</td>
<td>--</td>
</tr>
<tr>
<td>COMPRESSED DATA BYTES</td>
<td>-- 294,712</td>
</tr>
<tr>
<td>COMPRESS SAVES</td>
<td>--</td>
</tr>
<tr>
<td>COMPRESS -- BYTES SAVED</td>
<td>-- 134,551</td>
</tr>
</tbody>
</table>

3. DATASET: INFVSAM1.SHRINK.TESTDB.INDD033

<table>
<thead>
<tr>
<th>DONAME: PRIMARY</th>
<th>INDD033</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAPS PER BLK</td>
<td>4</td>
</tr>
<tr>
<td>RAP BLOCKS</td>
<td>10</td>
</tr>
<tr>
<td>RAP BYTES</td>
<td>1,020</td>
</tr>
<tr>
<td>DEVICE</td>
<td>3380</td>
</tr>
<tr>
<td>RECD LENGTH</td>
<td>1,690</td>
</tr>
<tr>
<td>RECDS/TRACK</td>
<td>21</td>
</tr>
<tr>
<td>FREE BLOCK PCT</td>
<td>0</td>
</tr>
<tr>
<td>FREE BLK FREQ</td>
<td>0</td>
</tr>
<tr>
<td>DCB IS:</td>
<td>RECFM</td>
</tr>
<tr>
<td>RECFM:</td>
<td>FBS</td>
</tr>
<tr>
<td>LRECL:</td>
<td>1,690</td>
</tr>
<tr>
<td>BLKSIZE:</td>
<td>1,690</td>
</tr>
<tr>
<td></td>
<td>TRACKS</td>
</tr>
<tr>
<td>CURR ALLOCATED</td>
<td>--</td>
</tr>
<tr>
<td>SEGMENTS</td>
<td>60</td>
</tr>
<tr>
<td>UNCOMPRESSED DATA BYTES</td>
<td>-- 35,657</td>
</tr>
<tr>
<td>COMPRESS USES</td>
<td>--</td>
</tr>
<tr>
<td>COMPRESSED DATA BYTES</td>
<td>-- 1,105,958</td>
</tr>
<tr>
<td>COMPRESS SAVES</td>
<td>--</td>
</tr>
<tr>
<td>COMPRESS -- BYTES SAVED</td>
<td>-- 411,139</td>
</tr>
</tbody>
</table>

DATABASE TOTALS:

<table>
<thead>
<tr>
<th>TRACKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>CURR ALLOCATED</td>
</tr>
<tr>
<td>SEGMENTS</td>
</tr>
<tr>
<td>UNCOMPRESSED DATA BYTES</td>
</tr>
<tr>
<td>COMPRESS USES</td>
</tr>
<tr>
<td>COMPRESSED DATA BYTES</td>
</tr>
<tr>
<td>COMPRESS SAVES</td>
</tr>
<tr>
<td>COMPRESS -- BYTES SAVED</td>
</tr>
</tbody>
</table>
Single DSG HIDAM Database—VSAM

**DATASET GROUP TOTALS:**

THE LENGTH OF THE AVG UNCOMPRESSED DB RECORD IS 8,820 BYTES INCLUDING SEGMENT PREFIXES.
THE LENGTH OF THE AVG COMPRESSED DB RECORD IS 8,966 BYTES INCLUDING SEGMENT PREFIXES.

1. DATASET: INFVSAM1.SHRINK.TESTDB.INDIV11
   - DNAME: PRIMARY --- INDIV11
   - DEVICE: 3380
   - CI SIZE: 4,096
   - CI/TRACK: 10
   - FREE SPACE PCT: 0
   - FREE CI FREQ: 0
   - DASD UTILIZATION IS 86 PERCENT.

   **CURRENT**
   - CURR ALLOCATED: 60
   - CURR USED: 44
   - COMPRESS USES: 29
   - COMPRESS SAVES: 15

   **TRACKS**
   - SEGMENTS: 45,705
   - UNCOMPRESSED DATA BYTES: 1,445,045
   - COMPRESSED DATA BYTES: 882,904
   - BYTES SAVED: 562,141

**HISAM Database—VSAM (DASD Utilization)**

**DATASET GROUP TOTALS:**

THE LENGTH OF THE AVG UNCOMPRESSED DB RECORD IS 5,043 BYTES INCLUDING SEGMENT PREFIXES.
THE LENGTH OF THE AVG COMPRESSED DB RECORD IS 3,217 BYTES INCLUDING SEGMENT PREFIXES.

1. PRIMARY DATASET: INFVSAM1.SHRINK.TESTDB.INSV1P
   - DNAME: --- INDIVIP
   - DEVICE: 3380
   - CI SIZE: 4,096
   - CI/TRACK: 10
   - MAX LRECL: 110
   - RECDS/CI: 37
   - RECDS/TRK: 370
   - DASD UTILIZATION IS 86 PERCENT.

   **OVERFLOW DATASET: INFVSAM1.SHRINK.TESTDB.INDSV1O**
   - DNAME: OVERFLOW --- INDSV1O
   - DEVICE: 3380
   - CI SIZE: 4,096
   - CI/TRACK: 10
   - MAX LRECL: 110
   - RECDS/CI: 37
   - RECDS/TRK: 370
   - DASD UTILIZATION IS 86 PERCENT.

   **CURRENT**
   - CURR ALLOCATED: 90
   - CURR ALLOCATED: 41
   - CURR USED: 21
   - COMPRESS USES: 28

   **TRACKS**
   - SEGMENTS: 29,551
   - UNCOMPRESSED DATA BYTES: 934,307
   - COMPRESSED DATA BYTES: 574,537
   - BYTES SAVED: 359,770

**DASD Allocation Recommendations Section**

The DASD Allocation Recommendations section reflects changes to the DASD allocations characteristics of the database. It also contains other recommendations for databases.
- HDAM randomization parameters.
- Free space percent and free record/CI frequency
- The amount of space to allocate in both tracks and cylinders.

**THE LENGTH OF THE AVG COMPRESSED DB RECORD IS**

Identifies the average compressed record length for the database.

**DATASET**

Identifies the name of the data set.
PRIMARY DATASET
(VSAM-HISAM) Identifies the name of the primary data set.

OVERFLOW DATASET
(VSAM-HISAM) Identifies the name of the overflow data set.

DDNAME
Identifies the type and DDNAME of the data set.

RAPS PER BLK
Identifies the number of root anchor points (RAPs) in each block.

RAP BLOCKS
(OSAM) Identifies the number of RAP blocks.

RAP BYTES
(OSAM) Identifies the number of RAP bytes.

DEVICE
Identifies the type of device on which the data set is stored.

RECD LENGTH
(OSAM) Identifies the length of the physical database record.

RECDs/TRACK
(OSAM, VSAM-HISAM) Identifies the number of physical records per track.

CI SIZE
(VSAM) Identifies the size of the control interval (CI) for the data set.

CIs/TRACK
(VSAM) Identifies the number of CIs per track for the data set.

MAX LRECL
(VSAM-HISAM) Identifies the maximum LRECL size of the specified data set.

RECDs/CI
(VSAM-HISAM) Identifies the number of records per CI for the specified data set.

FREE SPACE PCT
(VSAM-HIDAM) Identifies the percent of free space currently specified.

FREE CI FREQ
(VSAM-HIDAM) Identifies the frequency of free CIs currently specified.
RECOMMENDED
(OSAM) Provides information about the recommended allocation:

FREE BLOCK PCT
  Identifies the recommended free block percent.

FREE BLK FREQ
  Identifies the recommended free block frequency.

DCB IS
  Identifies the recommended DCB for the data set.

RECFM
  The recommended logical record length.

LRECL
  Identifies the recommended logical record length.

BLKSIZE
  The recommended block size.

DASD UTILIZATION IS
  Identifies the percent of DASD utilization.

RECOMMENDED DATA SET SIZE IS
  Identifies the recommended amount of DASD space for the data set in bytes.

ALLOCATION IN TRACKS
  Identifies the recommended allocation for the data set in tracks.

ALLOCATION IN CYLINDERS
  Identifies the recommended allocation for the data set in cylinders.

BLKFACT1
  (HISAM-VSAM) Identifies the recommended primary data set blocking factor.

BLKFACT2
  (HISAM-VSAM) Identifies the recommended overflow data set blocking factor.

RECLLEN1
  (HISAM-VSAM) Identifies the recommended logical record length of the primary data set.

RECLLEN2
  (HISAM-VSAM) Identifies the recommended logical record length of the overflow data set.
## Multiple DSG HDAM Database—OSAM

<table>
<thead>
<tr>
<th>Dataset recommendations:</th>
<th>The length of the avg uncompressed db record is 5,968 bytes including segment prefixes.</th>
</tr>
</thead>
<tbody>
<tr>
<td>These recommendations will maximize the use of dasd space.</td>
<td></td>
</tr>
<tr>
<td>These recommendations may be inappropriate for on-line applications.</td>
<td></td>
</tr>
<tr>
<td>These recommendations are based on the assumption that icf catalogs are being used. They are not valid for vsam user catalogs.</td>
<td></td>
</tr>
</tbody>
</table>

1. **DATASET: INFVSAM1.SHRINK.TESTDB.INDIV31**
   - **DDNAME:** PRIMARY --- INDIV31
   - **RAPS PER BLK:** 82 (RAP BLOCKS) 3 (RAP BYTES) 267
   - **DEVICE:** 3380
   - **RECOMMENDED:** FREE BLOCK PCT 0 (FREE BLK FREQ 0)
   - **DCB IS:** RECFM FBS LRECL 22,528 BLKSIZE 22,528
   - **DASD UTILIZATION IS 95 PERCENT.**
   - **RECOMMENDED DATA SET SIZE IS 22,528 ALLOCATION IN TRACKS 4 IN CYLINDERS --1**

2. **DATASET: INFVSAM1.SHRINK.TESTDB.INDIV32**
   - **DDNAME:** PRIMARY --- INDIV32
   - **RAPS PER BLK:** 20 (RAP BLOCKS) 12 (RAP BYTES) 1110
   - **DEVICE:** 3380
   - **RECOMMENDED:** FREE BLOCK PCT 0 (FREE BLK FREQ 10)
   - **DCB IS:** RECFM FBS LRECL 22,528 BLKSIZE 22,528
   - **DASD UTILIZATION IS 95 PERCENT.**
   - **RECOMMENDED DATA SET SIZE IS 22,528 ALLOCATION IN TRACKS 7 IN CYLINDERS --1**

3. **DATASET: INFVSAM1.SHRINK.TESTDB.INDIV33**
   - **DDNAME:** PRIMARY --- INDIV33
   - **RAPS PER BLK:** 4 (RAP BLOCKS) 60 (RAP BYTES) 5669
   - **DEVICE:** 3380
   - **RECOMMENDED:** FREE BLOCK PCT 0 (FREE BLK FREQ 10)
   - **DCB IS:** RECFM FBS LRECL 22,528 BLKSIZE 22,528
   - **DASD UTILIZATION IS 95 PERCENT.**
   - **RECOMMENDED DATA SET SIZE IS 22,528 ALLOCATION IN TRACKS 26 IN CYLINDERS --2**

## Single DSG HIDAM Database—VSAM

<table>
<thead>
<tr>
<th>Dataset recommendations:</th>
<th>The length of the avg uncompressed db record is 5,968 bytes including segment prefixes.</th>
</tr>
</thead>
<tbody>
<tr>
<td>These recommendations will maximize the use of dasd space.</td>
<td></td>
</tr>
<tr>
<td>These recommendations may be inappropriate for on-line applications.</td>
<td></td>
</tr>
<tr>
<td>These recommendations are based on the assumption that icf catalogs are being used. They are not valid for vsam user catalogs.</td>
<td></td>
</tr>
</tbody>
</table>

1. **DATASET: INFVSAM1.SHRINK.TESTDB.INDIV11**
   - **DDNAME:** PRIMARY --- INDIV11
   - **DEVICE:** 3380
   - **CI SIZE:** 22,528 (CI$s/track 2)
   - **CI SIZE:** 22,528 (FREE SPACE PCT 1
   - **FREE CI FREQ 10)
   - **DASD UTILIZATION IS 95 PERCENT.**
   - **RECOMMENDED DATA SET SIZE IS 22,528 ALLOCATION IN TRACKS 34 IN CYLINDERS --3**
HISAM Database—VSAM

**Note:** The HISAM database using VSAM can only have a single DSG.

DATASET GROUP RECOMMENDATIONS:
THE LENGTH OF THE AVG UNCOMPRESSED DB RECORD IS 3,217 BYTES INCLUDING SEGMENT PREFIXES.
THESE RECOMMENDATIONS WILL MAXIMIZE THE USE OF DASD SPACE.
THESE RECOMMENDATIONS MAY BE INAPPROPRIATE FOR ON-LINE APPLICATIONS.
THESE RECOMMENDATIONS ARE BASED ON THE ASSUMPTION THAT ICF CATALOGS ARE BEING USED. THEY ARE NOT VALID FOR VSAM USER CATALOGS.

1. PRIMARY DATASET: INFVSAM1.SHRINK.TESTDB.INDSV10
   DNAME: PRIMARY --- INDIV1P
   DEVICE ....... 3,380 CI SIZE ...... 22,520 CI s/TRACK ...... 2
   MAX LRECL .... 3,752 RECDS/Ci ...... 6 RECDS/TRK ...... 12
   DASD UTILIZATION IS 95 PERCENT.
   ALLOCATION IN TRACKS -- 19 IN CYLINDERS -- 2

OVERFLOW DATASET:
   DNAME: OVERFLOW --- INDSV10
   DEVICE ....... 3,380 CI SIZE ...... 22,582 CI s/TRACK ...... 2
   MAX LRECL .... 3,752 RECDS/Ci ...... 6 RECDS/TRK ...... 12
   DASD UTILIZATION IS 95 PERCENT.
   ALLOCATION IN TRACKS -- 3 IN CYLINDERS -- 1
   MODIFY THE 'DATASET' MACRO PARMS FOR THE DATA SET GROUP AS SHOWN:
   BLOCK PARAMETER -- BLKFACT1 = 6 BLKFACT2 = 6
   RECORD PARAMETER -- RECLEN1 = 3,752 RECLEN2 = 3,752

Management Summary Section

The Management Summary section consists of the following subsections:
- **DASD Usage and Savings** (see page 90)
- **CPU Performance** (see page 92)
- **Installation Computation Factors** (see page 94)
DASD Usage and Savings Subsection

This subsection compares two DASD usages and the unused DASD or DASD savings. Unused DASD indicates the amount of space available for growth. The allocations, usage and unused or savings are shown in tracks of DASD, percent, megabytes of DASD, cost of procuring that amount of DASD space and the amount charged for that amount of DASD each month. These values are reflected in the following horizontal headings: DASD TRACKS, %, DASD MEGABYTES, and COST IN DOLLARS - PURCHASE MNTHLY OPER.

<table>
<thead>
<tr>
<th>DASD</th>
<th>DASD MEGABYTES</th>
<th>COSTS IN DOLLARS</th>
<th>PURCHASE</th>
<th>MONTHLY OPER</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORIGINAL ALLOCATION:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CURRENT ALLOCATION:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CURRENT USED:</td>
<td>60 100 3 72 4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AVAIL FOR GROWTH:</td>
<td>50 84 3 72 4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>USED SPACE SAVINGS - FULL COMPRESSION BASED ON CURRENT ALLOCATION</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CURRENT USED:</td>
<td>10 100 0 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ESTIMATED USED:</td>
<td>10 100 0 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ESTIMATED SAVINGS:</td>
<td>0 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUGGESTED ALLOCATION BASED UPON FULL COMPRESSION</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUGG'D ALLOCATION:</td>
<td>2 100 0 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COMP'D DATA NEEDS:</td>
<td>1 50 0 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AVAIL FOR GROWTH:</td>
<td>1 50 0 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOTAL SAVINGS:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CURRENT ALLOCATION:</td>
<td>60 100 3 72 4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUGG'D ALLOCATION:</td>
<td>2 3 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EST'D DASD SAVINGS:</td>
<td>58 97 3 72 4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The following fields appear on this report:

**ORIGINAL ALLOCATION**

Provides information about current DASD allocation compared to current DASD needs for the uncompressed data giving the amount of DASD available for growth.

**CURRENT ALLOCATION**

Identifies the amount of DASD currently allocated.

**CURRENT USED**

Identifies the amount of DASD currently used.

**AVAIL FOR GROWTH**

Identifies the amount of DASD available for growth.

**USED SPACE SAVINGS - FULL COMPRESSION BASED ON CURRENT ALLOCATION**

Provides information about current DASD needs for the uncompressed data compared to the DASD needs for the compressed data giving the amount of space that CA Compress Data Compression for IMS can save.

**CURRENT USED**

Identifies the amount of DASD currently used. This value is the same as CURRENT USED under ORIGINAL ALLOCATION.

**ESTIMATED USED**

Identifies the amount of DASD used when full compression is implemented.

**ESTIMATED SAVINGS**

Identifies the amount of DASD that can be saved.
SUGGESTED ALLOCATION BASED ON FULL COMPRESSION

Provides information about suggested DASD allocations compared to the DASD needed by CA Compress Data Compression for IMS compressed data, when using the CA Compress Data Compression for IMS recommendations, and giving the amount of DASD available for growth.

SUGG’D ALLOCATION

Identifies the suggested allocation based on full compression.

COMP’D DATA NEEDS

Identifies the amount of DASD needed by the data at full compression.

AVAIL FOR GROWTH

Identifies the amount of DASD available for growth at full compression.

TOTAL SAVINGS

Provides information about current DASD allocation compared to the recommended DASD allocation and giving the total DASD savings that may be realized.

CURRENT ALLOCATION

Identifies the amount of DASD currently allocated.

SUGG’D ALLOCATION

Identifies the suggested allocation based on full compression.

EST’D DASD SAVINGS

Identifies the estimated DASD savings, arrived at by subtracting SUGG’D ALLOCATION from CURRENT ALLOCATION.

### CPU Performance Subsection

<table>
<thead>
<tr>
<th>Compression Performance (Costs are in Dollars):</th>
</tr>
</thead>
<tbody>
<tr>
<td>Megabytes Compressed Per CPU Second           = 3.3</td>
</tr>
<tr>
<td>CPU Seconds for Compressing Total Database    = 0</td>
</tr>
<tr>
<td>Cost for Compressing Total Database           = 0.00</td>
</tr>
<tr>
<td>CPU Seconds for Seqntl Procs of Database      = 0</td>
</tr>
<tr>
<td>Cost for Seqntl Procs of Database             = 0.00</td>
</tr>
<tr>
<td>Average Segment Length for Database           = 121</td>
</tr>
<tr>
<td>Segments Processed in One CPU Second          = 28,740</td>
</tr>
<tr>
<td>CPU Cost to Process 10,000 Segments -- Prime Time Off Shift</td>
</tr>
<tr>
<td>Batch:</td>
</tr>
<tr>
<td>On-Line:</td>
</tr>
</tbody>
</table>
The following fields appear on this report:

**COSTS ARE IN**

Identifies the monetary unit used by the country where the installation is located.

**MEGABYTES COMPRESSED PER CPU SECOND**

Identifies the number of megabytes of data that can be compressed or expanded per CPU second based on the processor speed provided by the installation.

**CPU SECONDS FOR COMPRESSING TOTAL DATABASE**

Identifies the amount of additional time in CPU seconds required to initially compress the data.

**COST FOR COMPRESSING TOTAL DATABASE**

Identifies the cost in CPU charges that incurred to initially compress the data. This cost is based on doing the conversion at night in the batch environment.

**CPU SECONDS FOR SEQNTL PROCS OF DATABASE**

Identifies the amount of additional CPU time that a run will increase if the data is processed in its totality in a sequential manner. This means reading and replacing every segment.

**COST FOR SEQNTL PROCS OF DATABASE**

Identifies the additional amount of CPU charges that a run will incur when the data is processed sequentially in its totality.

**AVERAGE SEGMENT LENGTH FOR DATABASE**

Identifies the result of dividing the total number of bytes of data, including keys, in the database by the total number of segments in the database.

**SEGMENTS PROCESSED IN ONE CPU SECOND**

Identifies the result of by dividing the number of bytes of data that can be processed in one CPU second by the length of the average segment.

**CPU COST TO PROCESS 10,000 SEGMENTS**

Identifies the amount that you can expect processing cost to increase due to data compression. This reflects the cost of the expansion/compression of 10,000 of the average segments. This increased cost is provided for each of the following four environments.

**BATCH/PRIME TIME**

A batch job executed during primary operating hours.

**BATCH/OFF SHIFT**

A batch job executed during nonprimary operating hours.
ON-LINE/PRIME TIME
An online transaction executed during primary operating hours.

ON-LINE/OFF SHIFT
An online transaction executed during nonprimary operating hours.

Installation Computation Factors Subsection
This subsection shows the CPU speed and monetary factors that you supplied and that were used during the DASD and cost calculations.

<table>
<thead>
<tr>
<th>INSTALLATION COMPUTATION FACTORS: (COSTS/CHARGES ARE IN DOLLARS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DFP (DATA FACILITY PRODUCT) LEVEL = 3.3</td>
</tr>
<tr>
<td>PROCESSOR ‘MIPS’ RATING = 16</td>
</tr>
<tr>
<td>DASD PURCHASE COST/MEGABYTE = 24.00</td>
</tr>
<tr>
<td>DASD USAGE CHARGE/MEGABYTE/MO = 220.00</td>
</tr>
<tr>
<td>ONE CPU HOUR - PRIME - BATCH = 890.00</td>
</tr>
<tr>
<td>ONE CPU HOUR - PRIME - ON-LINE = 2,200.00</td>
</tr>
<tr>
<td>ONE CPU HOUR - NIGHT - BATCH = 89.00</td>
</tr>
<tr>
<td>ONE CPU HOUR - NIGHT - ON-LINE = 220.00</td>
</tr>
<tr>
<td>CHARGE PER 1,000 EXCPS PRIME = 0.50</td>
</tr>
<tr>
<td>CHARGE PER 1,000 EXCPS NIGHT = 0.05</td>
</tr>
</tbody>
</table>

The following fields appear on this report:

COSTS/CHARGES ARE IN
Identifies the monetary unit used in the country in which the installation is located.

DFP (DATA FACILITY PRODUCT) LEVEL
Identifies the Data Facility Product (DFP) level provided by the installation. The rules for this DFP level were used in the DASD space calculations.

Note: DFP is responsible for storing and retrieving of the data. Different DFP levels have different capabilities.

PROCESSOR ‘MIPS’ RATING
Identifies the speed, in millions of instructions per second (MIPS), of the average processor for the installation. The MIPS rate directly affects the cost of expanding and compressing the data. The average MIPS speed should be rounded to the nearest MIPS.

DASD PURCHASE COST/MEGABYTE
Identifies the amount of money the installation spends when it acquires new DASD. This cost is expressed in terms of the money spent per megabyte of DASD storage acquired.

DASD USAGE CHARGE/MEGABYTE/MO
Identifies the amount of money the installation charges per MEGABYTE of DASD per month.
ONE CPU HOUR - PRIME – BATCH

Identifies the amount of money the installation charges for the use of one hour of CPU time during the prime operating hours to run jobs in the batch environment.

ONE CPU HOUR - PRIME - ON-LINE

Identifies the amount of money the installation charges for the use of one hour of CPU time during the prime operating hours to run transactions/jobs in the online environment.

ONE CPU HOUR - NIGHT – BATCH

Identifies the amount of money the installation charges for the use of one hour of CPU time during the nonprime operating hours to run jobs in the batch environment.

ONE CPU HOUR - NIGHT - ON-LINE

Identifies the amount of money the installation charges for the use of one hour of CPU time during the nonprime operating hours to run transactions/jobs in the online environment.

CHARGE PER 1,000 EXCPS PRIME

Identifies the amount of money the installation charges for 1,000 I/O operations during the prime operating hours.

CHARGE PER 1,000 EXCPS NIGHT

Identifies the amount of money the installation charges for 1,000 I/O operations during the nonprime operating hours.
Appendix A: Record Definition Language

This section contains the following topics:

- **RDL General Overview** (see page 97)
- **Performance Considerations When Using RDL** (see page 98)
- **How the RDL Operates** (see page 99)
- **RDL Terminology** (see page 99)
- **RDL Syntax Rules** (see page 101)
- **RDL Specification Field Type Codes** (see page 101)
- **RDL Specification Field Length Descriptors** (see page 114)
- **RDL Repetition Groups** (see page 116)
- **RDL Condition Groups** (see page 117)
- **RDL Position Function** (see page 121)
- **General Restrictions on RDL Use** (see page 123)
- **RDL Defaults** (see page 124)

**RDL General Overview**

The *Record Definition Language (RDL)* provided by the CA Compress Data Compression for IMS system lets you describe characteristics of data comprising a file that the CA Compress Data Compression for IMS system compresses and expands. The more you know about the data, the more detailed and precise the RDL specifications can be.

CA Compress Data Compression for IMS assumes default record definitions if you decide not to code RDL specifications. The effectiveness in achieving impressive compression ratios using default record definitions (and corresponding low processing overhead) is considerable. In the absence of other considerations, such as 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 system be used in first attempts to compress any file. Then, if performance is satisfactory, no further coding is necessary.

However, if you are interested in achieving the best possible trade-off between compression ratio and processing overhead for a particular file, you can experiment with alternative RDL specifications. An indication of the potential impact of coding a particular RDL specification on this trade-off can be found under individual RDL specification descriptions.
There are two primary performance characteristics of the CA Compress Data Compression for IMS data compression system:

**Compression Ratio**

The amount of storage space saved by compressing a data set.

Maximizing the compression ratio represents a cost saving, 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, depending on the unique circumstances.

**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.

Many factors influence whether the increase in processing overhead results in a cost or a saving. Although it seems that increased processing overhead to compress and expand records always costs more, more records can be stored per block by compressing records, reducing the number of times that the IBM Operating System File Access Method must be entered per program execution. 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, directly related to the computing power of the installed hardware. If there are CPU cycles available during a typical job mix, increased processing overhead may result in no increased cost.

There is a trade-off relationship between compression ratio and processing overhead. As compression ratio approaches the theoretical maximum (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 on your requirements.
How the RDL Operates

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

This compression is accomplished by using RDL specifications that you code and supply as input to build the File Description Table (FDT). The more complete and accurate description of the records that are compressed through RDL specifications you provide, the closer the compression ratio approaches the theoretical maximum. However, each RDL specification you code has an associated cost in processing overhead.

Consider two fields, customer name and customer address. Both fields contain textual data. However, unlike the customer address field, the customer name field never contains numeric characters. You can use this characteristic to differentiate these fields. Using two different RDL specifications to define these fields helps 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 data type 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 Data Compression for IMS is as a small set of fixed expected values. A file may contain a warehouse name field, where there are only few warehouses (assume eight) represented on the file. You can code an RDL specification that provides these names as a set of expected values in a table.

RDL Terminology

*RDL specification* is a unit of RDL coding that defines one CA Compress Data Compression for IMS field within a record. Each RDL specification defines one data field.

Field

*A field* is a series of consecutive byte locations within an uncompressed record that are defined with a single RDL specification as having similar compression or expansion characteristics. The boundaries of fields defined to CA Compress Data Compression for IMS need not correspond with actual field boundaries of data elements. For example, the entire record can be defined to CA Compress Data Compression for IMS as a single CA Compress Data Compression for IMS field. Unless otherwise noted, future references to the term *field* in this chapter denote a field as defined to CA Compress Data Compression for IMS, not a record data element.
Field Type

Fields are differentiated by the type of data they contain, such as character data, packed decimal data, and so forth. Consequently, each RDL specification consists, in part, of a field type code.

A field type code is a portion of an RDL specification that indicates the algorithm that the system uses to compress and expand the field currently defined.

Field Length

In addition to the type of data contained in a field, CA Compress Data Compression for IMS must know the boundaries of the field, where it begins and ends. Thus, each RDL specification 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 Data Compression for IMS 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 being defined. CA Compress Data Compression for IMS 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 (see page 121).

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 typically 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 Data Compression for IMS lets you perform arithmetic within a field definition to calculate a variable field length, using a special Variable Symbol (VS) (see page 111) to represent the calculated value. VS can be coded in certain field definitions in lieu of an integer length.

You may need to repeat an RDL specification two or more times in succession. For example, the record might contain 20 successive packed decimal numbers of identical length. To reduce user-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 many times, 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 contained in the record. This process is described later in this chapter.
Files may contain multiple record formats, where the format of a particular record can be determined from the contents of one (or more than one) individual field. CA Compress Data Compression for IMS allows alternative record definitions, which are effective for particular records based on the contents of a field. Such an RDL coding structure is referred to as a condition group, and is described later in this chapter.

RDL Syntax Rules

Syntax rules for the Record Definition Language are:

- Definitions appear within columns 1-72 of each card.
- Definitions can be continued onto any number of cards.
- Each field definition consists of a one- or two-character type code followed by a field length descriptor.
- Definitions are separated by commas or blanks or both.
- Groups of definitions may be 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, such as field lengths or arithmetic constants, must be from 1 through 32,767 (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.

Record definitions are checked for syntactical validity, but not applicability to the data, and printed. If syntax errors appear, each error is underscored by an alphanumeric character identifier that corresponds to the initial character of an explanatory error message printed directly below the RDL statement in error.

RDL Specification Field Type Codes

Each RDL specification consists of a field type code and a field length descriptor. The processing efficiency is a relative measure with 1 being the most efficient, and 6 being the least efficient. The following field type codes are available:

- **C1, C2, C3** (see page 102)
- **GA** (see page 103)
- **L** (see page 104)
- **MA, MB** (see page 104)
- **N** (see page 106)
Field Types C1, C2, and C3—Character Data

Field types C1, C2, and C3 indicate character data using internal frequency table 1, 2, or 3, as specified. Use type C to define groups of fields whose byte values have similar frequency distributions. Up to three different frequency distributions can be accommodated, one each for type C1, C2, and C3. If no other type code is clearly preferable, choose type C.

Compression is variable, depending on the skew of the distribution. The greater the skew of the distribution, the greater the compression ratio. Processing overhead is minimal.

Processing efficiency: 3

Fields that contain character data 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 one of three character frequency tables. A separate character frequency table is associated with each of the character-type RDL field specifications C1, C2 and C3.

For example, you can define mostly alphabetic fields as type C1, numeric fields as C2, and fields with another type of distribution as C3. The compression ratio thus obtained will be 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 Data Compression for IMS.

Example: Field Type C1, C2, and C3

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 six positions. The code could be C1F113 but C1F40, C2F39, C3F28, C2F6 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 mostly alphabetics, numerics, and blank so that its compression can 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 ZRF6. The city and state field can 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 inconceivable 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 GA—Garbage Data (Permanently Unused Fields)

Field type GA indicates garbage, filler, padding, alignment bytes, and so forth. Use this type to eliminate unneeded fields from the compression record. Compression is 100 percent. Processing overhead is negligible.

Processing efficiency: 1

Field type GA is specified when the content of a field is no longer of value in the file, such as permanently unused fields, fillers, or alignment bytes. This content is deleted in the compressed record, but appear as binary zeros on re-expansion.

Note: If these fields were not originally zeros, the expanded record is not an exact replica of the original. Redundancy checking used by CA Compress Data Compression for IMS does not consider type GA fields.

Example: GA Field Type

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

All four bytes of the RDW are sometimes superfluous in defining variable-length records, because the length may be implicit in the record content. The RDW can then be defined as GAF4. The File 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

Field type L indicates insert binary length indication (for COBOL users). Use type L to insert a binary tally of the compressed actual number of bytes comprising the compressed record as the first two bytes of the record following the RDW. This field type actually increases the compressed record length by two bytes. Processing overhead is negligible.

Processing efficiency: 1

The L field type must be the first specification of the record definition statements. In the compressed record it appears as a two-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 which follow this field. It is coded in the record definitions as the letter L, with no length descriptor.

Example: Field Type L

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

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

Field Types MA and MB—Pattern Matching

Field types MA and MB indicate pattern matching. Use type M when data in a field repeats partially or wholly from record to record, such as in a file containing multiple consecutive records with the same name and address. The M field type 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 set-up timing overhead per field and should 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. For fields smaller than 10 bytes in length, type C is preferable.

Compression is variable, depending on the degree of data repetition. Processing overhead is variable, decreasing as field length increases.

Processing efficiency: 3-5
Fields defined by this field type are compared with a pattern as follows:

- MA fields use the data in the first record as a pattern.
- MB fields use the data in the previous record as a pattern.

Matching characters and character repetitions are compressed. When your program calls SHRINK or EXPAND, these patterns are set during the initial CALL. Type MB patterns are reset during each subsequent CALL.

The following restrictions apply to the M type fields:

- The M type fields must be of fixed length.
- The M type fields cannot be specified within condition groups.
- When the M type fields appear within a repetition group or a nest of repetition groups, none of the enclosing repetition factors may be the variable symbol.
- The pattern used to compress and expand the field must be the same.

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, such as 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, the SCB can refer to the same FDT.

**Note:** The use of MA and MB pattern matching causes an 0C4 ABEND. (MA and MB pattern matching can be used with the CA Compress/2 Data Compression System.) Because FDTs were converted from sequential to load modules and linked as re-entrant, the additional storage address cannot be stored within the re-entrant FDT. This problem occurs in CA Compress Data Compression for IMS for IMS. For this reason, we do not recommend that you use pattern matching.
Field Type N—Fields Exempted From Compression

The N field type indicates exempt from compression. Use type N to exempt a field from compression, for example retrieval, match and sort keys, and any frequently processed field.

This field type yields no compression. Processing overhead is negligible.

**Processing efficiency:** 1

Type N fields 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.

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

The following requirements must be met:

- Type N fields cannot appear within condition groups (see page 117) or repetition groups (see page 116).
- 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 occurs in the record after variable-length fields, variable repetition groups or condition groups, use a Position Function (see page 121) to define the type N field before these other fields.
- The total length of all type N fields must not exceed 4095 bytes.

When deciding whether to exempt a field from compression consider the following:

- Key fields used to retrieve records from the file must be exempted from compression to enable record retrieval.
- Sort key fields, on which the file is regularly sorted, should be exempted from compression.
  - When sort key fields are exempted from compression, you can invoke a sort utility program to sort the file in its compressed state, avoiding all expansion overhead.
- Match key fields, for record matching applications, should be exempted from compression.
  - 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.
Any field at a fixed offset from the record origin that appears in all records of the file can be considered for exemption from compression.

The field that is always or frequently operated on in application programs can be considered for exemption from compression. Consideration must be given to the trade-off between compression ratio and processing overhead. Where compression ratio is critical, the field should be compressed. Where minimum processing overhead is critical, the field should be exempted from compression. Benchmark testing both ways helps you determine the optimal trade-off.

Field Type PD—Packed Decimal Data

The PD field type indicates packed decimal data (USAGE COMPUTATIONAL-3 for COBOL users). 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.

Processing efficiency: 5-6

The following requirements must be met:

- The field length must be less than nine bytes.
- 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 may be used in these cases.)

CA Compress Data Compression for IMS treats invalid fields automatically as type UN. Instead of being compressed, a field defined as type PD which contains invalid data is enlarged by one bit. No data is lost, and on expansion, field content is the same as it was before compression. If, across the file, a field is thought to contain invalid data a substantial number of times, greater compression and reduced processing overhead results from defining the field as a type C instead of PD. A message is printed with the statistics produced by the File Compression Utility indicating the number of times invalid data was encountered in a PD field during compression.
CA Compress Data Compression for IMS converts packed decimal numbers 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 three-byte field), then defining the field as type PD performs poorly 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, specifying type PD yields improved performance over alternative specifications.

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

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

$20'PDF4'$.

If the average number of significant digits is five 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**

The S or X field types indicate a set of expected values. 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.

**Processing efficiency:** 3
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 File 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 in the method of specifying 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 in the table is coded as two hexadecimal digits.

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

```
tmnv
```

- **t**
  - Specifies the field type, S or X.

- **m**
  - Specifies the field length.
    - **Limits:** a two-digit number 01 through 99 (code the leading zero for values from 01 through 09)

- **n**
  - Specifies the number of entries in the table of expected values.
    - **Limits:** a two-digit number 01 through 16 (code the leading zero for values from 01 through 09)

- **v**
  - Specifies the table of expected values. Table entries of expected values 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 must be continued starting in column 1 of the next RDL statement. For type X, spaces may appear between pairs of hexadecimal digits for readability, but the table must contain exactly 2*m*n hexadecimal digits.

CA Compress Data Compression for IMS 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. The expected value most likely to occur should be coded first; the expected value least likely to occur should be coded last.
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 illustrate correctly coded type S and X field definitions, and to illustrate 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 four-byte field containing DOGb/, CATb/, FISH, BIRD, FROG, or "other", where "other" occurs infrequently. If this field is specified as:

S0405DOGbCATbFISHBIRDFROG

The 32-bit (four-byte) field compresses to four bits, one bit as an error flag and three bits to represent which of the five values occur. An "other" field cannot be compressed, but grows by one bit to 33 bits.

Field Type UN—Undefined Fields

The UN field type indicates undefined field. Use the UN type for fields that fall into none of the other categories and cannot readily be compressed, such as bit switches or floating-point numbers. This type is useful when all three type C specifications are already in use. Type UN fields are not compressed, but neither do they grow in length.

This field type yields no compression. Processing overhead is minimal.

Processing efficiency: 2

Example: Field Type UN

Consider defining a floating-point field when 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 on the compression ratio for all of the other fields defined by the type C specification. To avoid this effect, define the floating-point field as type UN.
Field Types V, VP, and VZ—Calculating a Value for the Variable Symbol

The V, VP, and VZ field types indicate variable-length fields. Use one of these field types to define a field that contains the actual length of a variable-length portion of the record or whose content is used to calculate the actual length of a variable-length portion of the record.

Field types V, VP and VZ are functionally equivalent. The only difference is in the format of the data contained in the type V, VP or VZ field:

- Use type V to define fields that contain binary integer data.
- Use type VP to define fields which contain packed decimal data.
- Use type VZ for fields which contain right-justified zoned decimal data.

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 present, is stored within the record or can be calculated from information stored within the record. This capability is implemented using the Variable Symbol (VS) that that is coded in RDL specifications.

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 where it appears for each record that the specification applies to. 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.

Type V is not compressed. Type VP is compressed as PD. Type VZ compressed as ZR. Processing overhead is minimal to moderate.

**Processing efficiency:** V=2, VP=4, VZ=6

**Note:** If invalid data is encountered in a VP or VZ field, CA Compress Data Compression for IMS abends with a user code of 20, and the following error message in the system output writer:

```
VP
INVALID TYPE VZ FIELD
```

To suppress the abend, code the RC parameter when calling the SHRINK or EXPAND subroutine, as shown with the messages:

```
REC DEFS IMPLY WRONG LENGTH
```

```
CHECK BYTE MISMATCH
```
The following special formats are available for RDL specifications for field types V, VP and VZ:

\[ \text{tn} \]
\[ \text{tno} \]
\[ \text{tno1i1o2i} \]
\[ t \]

Specifies the field type. The following values are available:
- V
- VP
- VZ

\[ n \]

Specifies the length of the V, VP, or VZ field, in bytes.

Limits:
- For type V: 1 through 4
- For types VP and VZ: 1 through 8

\[ o1, o2 \]

Specify the operator. The following values are available:
- + — Addition
- - — Subtraction
- * — Multiplication
- / — Division

\[ i1, i2 \]

Specify the values for calculating the result that is stored in VS.

Limits: integers 0 through 32767

Note: Any remainder resulting from a division operation is dropped. Arithmetic operations are evaluated left to right.

Examples: Special Formats
- The following example means that the currently defined four-byte field contains a binary integer whose value is to be stored in the VS. Use of the VS in a subsequent RDL specification refers to the value of the binary integer.
  \[ V4 \]
- The following example means that 500 is added to the binary integer, and the result is stored in the VS.
  \[ V4+500 \]
The following example means that the currently defined three-byte field contains a packed decimal number. Three is subtracted from the number, and the result is divided by 20, and the result is stored in the VS.

VP3-3/20

The following example shows the record definition for a variable-length record that is treated as a single type C1 field:

V2-4,GAF2,C1FVS

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

The following example shows a variable-length record made up of the four-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 four-byte packed decimal numbers. The record is defined as follows:

V2-84/40,GAF2,C1F80,VS’10’PDF4’’

Field Types ZL and ZR—Zoned Decimal Data

The ZL and ZR field types indicate zoned decimal, left- or right-justified fields. Use field types ZL and ZR to define fields that contain zoned decimal data:

- Type ZL defines a field containing left-justified zoned numeric data, possibly followed by one or more filler characters.
- Type ZR defines a field containing right-justified zoned numeric data, possibly preceded by one or more filler characters.

ZL and ZR specifications are most useful when a separate type C specification cannot be dedicated to zoned numeric fields, or when the record contains multiple noncontiguous zoned numeric fields. For zoned decimal fields of nine bytes or less, a definition using type ZR or ZL is preferable to using a type C specification. 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 2,147,483,648. 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 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 are converted to binary, bit aligned, variable-length floating-point. Blank or zero fields compress to five bits, while fields with value between 16n−1 compress to 5+4n bits, regardless of field length.

Compression is excellent when value is zero, variable, increasing as the magnitude of the value increases. Processing overhead is moderate.

**Processing efficiency**: 3-6

**Note**: 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 the same data as it did before compression. A message with the statistics is printed indicating the number of times invalid data was encountered in a ZL or ZR field during compression.

**Example: Field Types Zl and ZR**

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.

---

**RDL Specification Field Length Descriptors**

The following forms for coding field length descriptors are valid:

**Fn**

Indicates fixed-length field of length n, where 1 < n < 16384; n may have leading zeros but may not exceed eight decimal digits.

**FVS**

Indicates length determined by the previous type V field. This descriptor is valid only for types C1, C2, C3, UN, and GA.

**VER**

Indicates variable-length field extending to the end of the record. This descriptor is valid only for types C1, C2, C3, UN, and GA.

**Dc**

Indicates variable-length field delimited by an 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 field definition, if any, begins beyond the delimiter.
**Example: Field Definitions with Length Specifications**

This example shows how you can specify field length using the descriptors:

ZRF8, C1F100, PDF4, PDF4, C1VER

ZRF8

Specifies that the first eight-byte field is a right-justified, zoned decimal number.

C1F100

Specifies that the next 100-byte field is character data.

PDF4, PDF4

Specify that the next two fields are four-byte packed decimal numbers.

C1VER

Specifies that the remainder of the record is treated as character data.
RDL Repetition Groups

A repetition group represents one or more RDL specifications, enclosed by single quotes and preceded by a repetition factor. If a sequence of one or more field definitions is repeated, you can code it once with a repetition factor by enclosing the sequence within single quotes and preceding the initial single quote with a two-digit number as follows:

\[ n \text{'}field-sequence\text{'n} \]

- \( n \) Specifies how many times the field sequence repeats.
- Limits: 02 through 99 ( inclusively)

\( field-sequence \)

- Specifies the repeating field sequence.

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

\[ \ldots(\ldots03\ldots)\ldots\text{'} \]

and

\[ 03\ldots(\ldots\ldots) \]

are syntactical errors.

The symbol VS can be used instead of the two-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 three fields:

- Line item description, 20 bytes of character data
- Line item quantity, four bytes packed decimal data
- Line item amount, seven bytes packed decimal data

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

\[ \ldots,VZ2,\ldots,VS \text{'}C1F20,PDF4,PDF7\text{' and so forth} \]

Where:

- VZ2 is the field containing the actual number of occurrences.
- VS is 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.

**Example: Repetition Group**

This example shows how you can use repetition groups. Following is the original field sequence:

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

You can code this sequence with a repetition group as follows:

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

**RDL Condition Groups**

A **condition group** is a group of RDL specifications, enclosed in parentheses, that applies to a particular record if the value contained in a field of the record equals the value coded within the condition group. Condition group lets you define multiple record formats for a file. The format of an individual record can be determined from the contents of a field within the record. You can specify multiple condition group. Use comma to separator the last condition group in a series from any RDL specifications (for example, another condition group) that follow.

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 four bits, regardless of the length of the value. No separate RDL specification is used to compress the value. The lengths of the values may differ within a condition group series.

Condition group has the following format:

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

- **n**
  - Specifies the length of the field that is tested.
  - **Limits**: two-digit number

- **v**
  - Specifies a value for comparison, n bytes in length.
  - **Limits**: Any of the 256 possible byte values. Nongraphic values must be multipunched.

- **f**
  - Specifies any RDL specification or repetition group.
If the current $n$ bytes in the input record being processed are equal to the value $v$, the remaining definitions within the parentheses apply to the record; otherwise, they are skipped.

**Note:** To avoid excessive multipunching, code the value $v$ in hexadecimal format as follows:

(03XYZ,C1F80)  
(X03E7 E8 E9,C1F80)

- Code an X preceding the length specification $n$.
- Code the value $v$ as pairs of hexadecimal digits. For readability, you can leave the spaces between pairs of hexadecimal digits.

### Alternative Condition Groups

You can define alternative condition groups by coding a series of consecutive condition groups that are not separated by a comma, but by one or more blanks instead. The following rules apply:

- The first condition group in the series whose condition is met applies to the record being processed.
- The remaining condition groups in the series are skipped.

The maximum number of condition groups in a series is 16, including the final default condition group (see page 120), whether user-specified or automatically provided. The maximum number of condition group series which can be coded is limited only by the amount of space available in the FDT.

#### Example: Alternative Condition Groups

This example describes how to use alternative condition groups. Consider an elementary invoicing file. This file consists of sets of records, where each set of records represents one invoice. Each set of records consists of a header record, one or more detail records, and a trailer record.
The following example illustrates these records and the fields contained in them.

<table>
<thead>
<tr>
<th>INVOICE#</th>
<th>RECORD</th>
<th>INVOICE TYPE</th>
<th>DATE</th>
<th>filler</th>
<th>RECORD TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;</td>
<td>&quot;</td>
<td>ITEM DESCRIPTION</td>
<td>ITEM QTY</td>
<td>AMOUNT</td>
<td>TYPE=H</td>
</tr>
<tr>
<td>&quot;</td>
<td>&quot;</td>
<td>&quot;</td>
<td>&quot;</td>
<td>&quot;</td>
<td>TYPE=D</td>
</tr>
<tr>
<td>&quot;</td>
<td>&quot;</td>
<td>filler</td>
<td>TOTAL QTY</td>
<td>TOTAL AMOUNT</td>
<td>RECORD TYPE=T</td>
</tr>
</tbody>
</table>

Field | Length | DATA TYPE
-----|--------|-------------
INVOICE# | 8 | Zoned Decimal
RECORD TYPE | 1 | Character (H, D, or T)
INVOICE DATE | 8 | Alphanumeric
ITEM DESCRIPTION | 20 | Characters
ITEM QUANTITY | 4 | Packed Decimal
ITEM AMOUNT | 7 | Packed Decimal
TOTAL QUANTITY | 4 | Packed Decimal
TOTAL AMOUNT | 7 | Packed Decimal

The following RDL specifications define these records using condition groups:

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

**Note:** To illustrate the importance of using correct separator, consider the sample RDL specifications in the previous code example modified as follows:

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

The first and the second condition groups are separated by comma. When a record whose RECORD TYPE field contained an "H" is encountered, the first condition group applies. But after processing the C1F8,GAF23 RDL specifications, the following condition groups are not skipped. CA Compress Data Compression for IMS 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 can occur.

If the file contains a record whose record type is not H, D or T, CA Compress Data Compression for IMS abends with a user code of 15. This is because no record definitions are specified past the invoice number for any record whose record type is not H, D, or T. To avoid this situation, code a default condition group at the end of the condition group series.
Default Condition Group

Default condition group supplies a set of record definitions if none of the preceding condition group tests are met, and the actual content of the bytes in the record is not known. The RDL specifications in the default condition group apply regardless of the contents of the current byte in the record.

Default condition group (00) is automatically supplied if you omit to code your default condition group. 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.

Default condition group must be coded as the last in the series.

Default condition group has the following format:

\[(00, f, \ldots, f)\]

00

Indicates a default condition group.

f

Specifies any RDL specification or repetition group.

Example: Default Condition Group

The following definition describes the records shown in the RDL Condition Groups section:

\[ZRF8, (01H, C1F8, GAF23)b(01D, C2F20, PDF4, PDF7)b(01T, GAF20, PDF4, PDF7)b(00, C3F32)\]

Without default conditions group, the definition would look as follows:

\[ZRF8, (01H, C1F8, GAF23)b(01D, C2F20, PDF4, PDF7,)b(01T, GAF20, PDF4, PDF7), C3F32\]
RDL Position Function

RDL Position Function lets you alter the internal field pointer (IFP). By default, CA Compress Data Compression for IMS automatically adjusts an internal field pointer (IFP) to the current displacement within the record as fields are processed.

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

The Position Function has the following format:

- To set IFP to \( n \) and reposition IFP to the \( n+1 \)th byte in the record, specify the following:
  
  \[ P_n \]

- To add \( n \) to the IFP and reposition IFP forward \( n \) bytes, specify the following:
  
  \[ P+n \]

- To subtract \( n \) from the IFP and reposition IFP backward \( n \) bytes, specify the following:
  
  \[ P-n \]

- To reset IFP to its value immediately preceding the last \( P_n, P+n \) or \( P-n \), specify the following:
  
  \[ P \]

**Note:** If there were no previous Position Functions executed, \( P \) is ignored. \( P \) cannot be specified as the initial Position Function.

- \( P \)

  Defines the IFP and is coded as shown.

- \( n \)

  Specifies the displacement of the IFP. Displacements are computed relative to 0 that indicates the start (origin) of the record.

**Limits:** one-to five-digit integer or the variable symbol (VS)
The following rules apply to the use of Position Function:

- The IFP must remain within the bounds of the record. This error is usually, but not always, caught during the prepass or compression phase, because complete checking would involve substantial timing 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 can occur on re-expansion.

- Redefining through repositioning degrades performance, and we recommend that you specify it only when necessary.

Example: RDL Position Function

Consider a name and address file that contains three types of 80-byte records, shown in the following illustration:

<table>
<thead>
<tr>
<th>Field</th>
<th>Length</th>
<th>Type of Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>79</td>
<td>Character</td>
</tr>
<tr>
<td>RECORD TYPE</td>
<td>1</td>
<td>Character</td>
</tr>
<tr>
<td>STREET ADDRESS</td>
<td>79</td>
<td>Character</td>
</tr>
<tr>
<td>CITY,STATE</td>
<td>74</td>
<td>Character</td>
</tr>
<tr>
<td>ZIP CODE</td>
<td>5</td>
<td>Zoned Decimal</td>
</tr>
</tbody>
</table>

The following RDL specifications define this file:


- P79,—Sets the IFP at the RECORD TYPE field. The IFP is relative to 0, thus IFP of 0 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 of the record, and defines the STREET ADDRESS field.

- (01C,P,C3F74,ZRF5),—If the RECORD TYPE field contains C, resets the IFP to the beginning of the record, and defines the CITY, STATE and ZIP CODE fields.

- P+1—One of the condition groups has been applied to the record (if all records in the file contain A, B, or C in the RECORD TYPE field). The IFP is pointing, once again, at the RECORD TYPE field. Coding P+1 is necessary 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 following message:

    REC DEFS IMPLY WRONG LENGTH.
General Restrictions on RDL Use

The Record Definition Language is employed to construct the File Descriptor Table (FDT). The FDT has a maximum size and, as such, there is a corresponding upper limit on the maximum amount of RDL specifications permitted 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.

- **GA PD ZL ZR**: 6 bytes
- **C1 C2 C3**: 18 bytes
- **MA**: 92 bytes for the first occurrence.
- **MB**: 54 bytes for each subsequent occurrence.
- **UN**: 14 bytes
- **N**: 20 bytes
- **V**: 48 bytes
- **VP, VZ**: 40 bytes
- **S, X**: 26 bytes
- **First “(” in a condition group series**: 72 bytes
- **Each remaining “(” in a condition group series**: 54 bytes
- **Fn(Where n < 128)**: 4 bytes
- **Fn(Where n > 128)**
The space required must be summed according to RDL specifications as coded, and the total may not exceed 2800. For example, consider the following RDL specifications:

V2·4,GAF2,C1FVS.

Using the table, the space required for these RDL specifications is:

- 48 bytes for V2·4
- 6 bytes for GA
- 4 bytes for F2
- 18 bytes for C1
- 46 bytes for FVS

The total is 122 bytes, which is well within the 2800-byte limit.

RDL Defaults

If the RECDEF DD statement is not present in the execution JCL for the File Prepass Utility or by the Interactive Dialog, or if it specifies a null data set (with no records), default RDL specifications are generated based on characteristics of the data set defined by the INFILE DD statement. The defaults are also generated if the only RDL specification that you supply is a single type L field.
In the default RDL specification formulas shown in the previous section, the following variables are substituted with appropriate values, obtained from the data set label or JCL specifications:

- $x$ — 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 Interactive Dialog.

### RDL Default Specifications for Sequential Files

<table>
<thead>
<tr>
<th>Files</th>
<th>Formulas for Default RDL Specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed-length</td>
<td>C1Fy</td>
</tr>
<tr>
<td>Variable-length</td>
<td>V2-4</td>
</tr>
<tr>
<td></td>
<td>GAF2</td>
</tr>
<tr>
<td></td>
<td>C1FVS</td>
</tr>
</tbody>
</table>

### RDL Default Specifications for ISAM Files

<table>
<thead>
<tr>
<th>File</th>
<th>Formulas for Default RDL Specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed-length, key at beginning of record</td>
<td>Nk,C1Fy</td>
</tr>
<tr>
<td>Fixed-length, RKP=1*</td>
<td>Nk’,C2Fy</td>
</tr>
<tr>
<td>Fixed-length, RKP&gt;1</td>
<td>N1,C1Fx’,Nk,C2Fy</td>
</tr>
<tr>
<td>Fixed-length, key at end of record*</td>
<td>N1,C1Fx’,Nk</td>
</tr>
<tr>
<td>Variable-length, relative key position=4</td>
<td>V2-j,GAF2,Nk,C2FVS</td>
</tr>
</tbody>
</table>
### RDL Default Specifications for VSAM Files

<table>
<thead>
<tr>
<th>Files</th>
<th>Formulas for Default RDL Specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed-length, key at beginning of record</td>
<td>Nk,C1Fy</td>
</tr>
<tr>
<td>Fixed-length, key within the record</td>
<td>N1,C1Fx’,Nk,C2Fy</td>
</tr>
<tr>
<td>Fixed-length, key at end of record</td>
<td>N1,C1Fx’,Nk</td>
</tr>
<tr>
<td>Variable-length, key at beginning of record</td>
<td>Nk,C2VER</td>
</tr>
<tr>
<td>Variable-length, RKP=1*</td>
<td>Nk’,C2VER</td>
</tr>
<tr>
<td>Variable-length, RKP&gt;1</td>
<td>N1,C1Fx’,Nk,C2VER</td>
</tr>
</tbody>
</table>

**Note:**
* Default definition permits record deletion when the DCB parameter OPTCD=L is specified. If "L." is specified, all defaults begin with "L,...."
byte value
The value of the binary number contained in one eight-bit byte. For example, the byte value for the character A is 11000001, the byte value for COBOL LOW-VALUES is 00000000, and the byte value for the character 9 is 11111001.

compression ratio
*Compression ratio* identifies the measure of the amount of storage space saved by compressing a data set. The ratio is expressed by the following percentage:

\[
100 \times \frac{x-y}{x}
\]

x
Identifies average record length in bytes before compression.

y
Identifies average record length in bytes after compression

condition group
A *condition group* is a group of RDL specifications, enclosed in parentheses, that applies to a particular record if the value contained in a field of the record equals the value coded within the condition group.

condition group series
A *condition group series* represents two or more adjacently coded, mutually exclusive condition groups separated by one or more spaces (blanks). The first condition group for which an equality is found between the field currently tested and the coded condition group test value applies to the record. The remaining condition groups in the series are skipped for the current record.

control block (SCB)
A *control block* is a user-supplied formatted main storage area used by the system when processing FDTs in load module format.

expansion overhead
The number of CPU cycles necessary to execute the code required for expansion of a compressed record.

field
A *field* is a series of consecutive byte locations within an uncompressed record that are defined with a single RDL specification as having similar compression or expansion characteristics.

field length descriptor
A *field length descriptor* is the portion of an RDL specification using custom compression which indicates the length, in bytes, of the field currently defined.
field type code

A field type code is a portion of an RDL specification that indicates the algorithm that the system uses to compress and expand the field currently defined.

File Description Table

A File Description Table (FDT) contains complete information that is needed to change the form of the file from uncompressed to compressed and conversely. FDT is a short 1-through 8-KB sequential data set.

file number

A file number is the value from 0 through 31, inclusive, passed as a parameter to subroutines, identifying the FDT associated with the subroutine call. The file number corresponds to the xx of a TABLxx DD statement defining the FDT in the JCL invoking the job.

Implement

Implementing is a multi-step process that places a file or database under the control of CA Compress Data Compression for IMS. The process begins after the CA Compress Data Compression for IMS product and its interfaces have been installed, and ends with the successful completion of the initial compression.

internal field pointer (IFP)

An internal field pointer (IFP) is a pointer, maintained by the CA Compress Data Compression for IMS system, that points to the current displacement within the record that is processed. For each record, the value in the internal field pointer is initially zero, corresponding to the first byte of the record. As each field is processed, the internal field pointer is increased by the length of the field, to the position of the next field that is processed. You can explicitly control the value of the internal field pointer by specifying the Position Function.

length descriptor

See the entry for "field length descriptor."

post-compression

A variable-length bit string to which a byte value from the uncompressed data set is converted as a result of compression by a type C RDL specification. The most frequently occurring byte value maps to the shortest (fewest number of bits) post-compression bit code. The least frequently occurring byte value maps to the longest post-compression bit code.

Prepass

Prepass is the process by which the CA Compress Data Compression for IMS system reads a file (or a portion of a file) before it is compressed, to collect statistical information, and creates the File Descriptor Table. The File Prepass Utility must be executed for a particular file before it is compressed for the first time.
processing overhead
The number of CPU cycles required to execute the code to perform a CA Compress Data Compression for IMS system function; for example, to compress or expand a field or a record.

RDL specification
*RDL specification* is a unit of RDL coding that defines one CA Compress Data Compression for IMS field within a record.

Record Definition Language (RDL)
The *Record Definition Language (RDL)* provided by the CA Compress Data Compression for IMS system lets you describe characteristics of data comprising a file that the CA Compress Data Compression for IMS system compresses and expands.

Record Descriptor Word (RDW)
The *Record Descriptor Word (RDW)*, defined by IBM Data Management as a four-byte field prefixing variable-length records. The two high-order bytes contain the actual length of the record as a binary integer.

redundancy check byte
A byte appended by the system to each compressed record, representing the logical sum of the bits in the uncompressed record. This value is calculated again upon expansion of the record and compared with the redundancy check byte to verify data integrity.

redundancy checking
The process of recalculating the logical sum of the bits in an expanded record and comparing the result with the redundancy check byte appended to the compressed record. An equality ensures data integrity.

repetition factor
A value, preceding one or more RDL specifications enclosed by single quotes, indicating the number of times to repeat that sequence of RDL specifications.

repetition group
A *repetition group* represents one or more RDL specifications, enclosed by single quotes and preceded by a repetition factor.

repetition indicator
A post-compression bit code indicating that the preceding post-compression bit code occurs repeatedly in sequence within the compressed record.

variable symbol
An internal CA Compress Data Compression for IMS register that you can set using type V, VP or VZ RDL specifications. The register is used to contain the length of a variable-length portion of a record, when this value is determined from information contained in the record.
VS

The coded representation of the Variable Symbol in the field length indicator or repetition factor of an RDL specification.