



A Guide to IMS Hardware Data Compression

Rich Lewis
IMS Advanced Technical Support
IBM

April 2004

Table of Contents

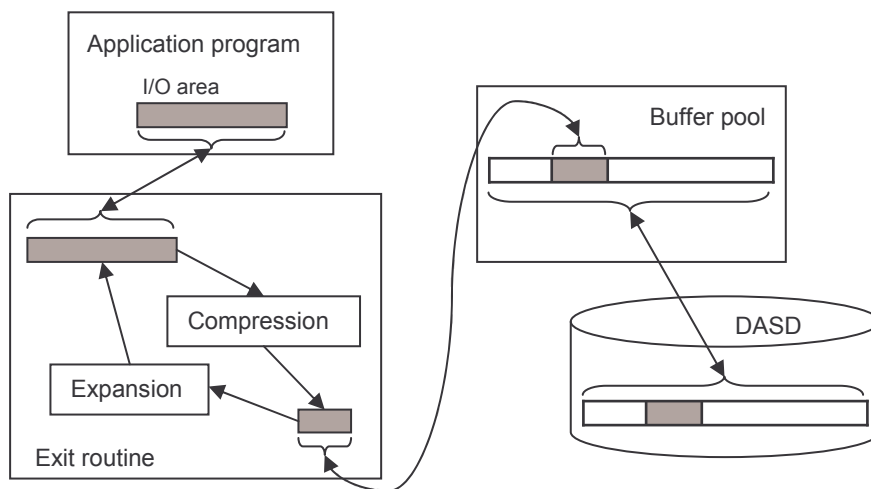
An Overview of IMS Compression	3
Exit Routine Options.....	3
Sample Exit DFSCMPX0	4
Hardware Data Compression	4
Selecting an Exit Routine.....	4
Building Dictionaries	5
Key Compression vs. Data Compression.....	5
Setting a Minimum Size for Segments	6
Adjusting HDAM, PHDAM, and DEDB Parameters.....	7
HDAM and PHDAM	8
DEDB.....	8
Defining Compression for a Database Segment	8
SEGM Statement	8
COMPRTN=	8
BYTES= parameter.....	10
Why Compression Routines Might Not Compress.....	11
Warning About Compression Routine Abends.....	11
Implementation Steps for Hardware Data Compression	12

An Overview of IMS Compression

This paper is an overview of the use of hardware data compression with IMS databases. It explains how you define compression exit routines and how IMS implements their use.

You may use compression with HDAM, HIDAM, HISAM, PHDAM, PHIDAM, and DEDB databases. HIDAM indexes, PHIDAM indexes, and secondary indexes cannot be compressed. For databases with logical relationships, logical children cannot be compressed.

IMS implements compression through Segment Edit/Compression Exit routines. These routines reduce the amount of space used by segments in VSAM CIs and OSAM blocks. This reduces the size requirements for the database data sets. The use of compression does not affect application programs. The segments in program I/O areas are not compressed. The following diagram illustrates the use of compression.



On an insert or replace call, the application program places the uncompressed segment in the program's I/O area. The exit routine is passed a copy of the segment. The exit routine compresses the segment. IMS takes the compressed segment and places it in a block or CI in the buffer pool. IMS writes this block or CI to the database data set on DASD.

On a get call, IMS retrieves the compressed segment from the buffer pool. It may have to read the block or CI from DASD. IMS passes the compressed segment to the exit routine. The exit routine expands the segment. IMS then places the expanded segment in the program I/O area.

A Segment Edit/Compression Exit routine is defined in the database DBD. The same routine may be used for multiple segments or different routines may be used for different segments.

Exit Routine Options

IMS gives you several alternatives for segment edit/compression routines. First, it supplies a sample routine, DFSCMPX0. Second, it provides a capability to generate exit routines which

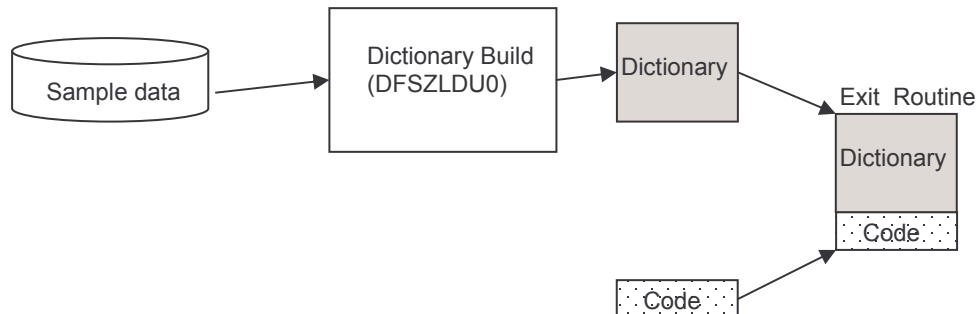
invoke hardware data compression. Third, you may write an exit routine. Finally, you may purchase one. Only the first two options will be discussed here.

Sample Exit DFSCMPX0

The sample exit routine, DFSCMPX0, uses run length encoding (RLE) to compress data. RLE compresses repeating bytes. If a string consists of a byte which is repeated more than three times, it is reduced to three bytes. For example, a string of 40 blanks is reduced to three bytes. This type of compression is very effective on some types of data. The most typical example is data with many large blank fields. Obviously, it is not effective on data without repeating bytes.

Hardware Data Compression

IMS supplies facilities to generate exit routines which use hardware data compression. This compression uses a Ziv-Lempel compression technique. This technique replaces a variable number of bytes with a fixed number of bits. A dictionary is built that maps these byte strings to their corresponding sets of 12 bits. The dictionary can be customized for a set of data. The dictionary is built so that frequently occurring byte strings get the best compression. For example, the dictionary might have an entry for "San". These three bytes would be compressed to 12 bits. It might also have an entry for "San Jose". These eight bytes would be compressed to 12 bits. IMS provides facilities that read sample data and build a dictionary which is optimized for the data. This dictionary is combined with code to form a segment edit/compression exit routine. The following diagram illustrates the building of an exit routine which uses hardware data compression.



The IMS Hardware Data Compression Extended product simplifies the dictionary build process and provides dictionary evaluation capabilities.

Selecting an Exit Routine

Selecting exit routines involves a trade-off between compression effectiveness and management of many routines. Typically, the best compression is produced with hardware data compression using customized dictionaries. A separate dictionary is developed for each segment type. On the other hand, maintaining separate dictionaries for hundreds of segment types can be inconvenient.

A reasonable compromise is to create custom dictionaries for a small number of segment types. These would be the segments which require the most space and, therefore, get the most benefit from compression. One or more generalized dictionaries could be shared by other segments.

The IMS Hardware Data Compression Extended product supplies an exit routine which uses a generic dictionary. It was developed from English text data. This exit routine may be sufficient for many segments. It typically provides 40 to 60 percent compression. You could also create enterprise dictionaries. These are dictionaries built from enterprise wide samples of data. For example, segments which primarily contain text could use a dictionary built from a sample of your text data. Segments which primarily contain numeric data could use a dictionary built from sample numeric data. Finally, segments with mixtures of text and numeric data could use a dictionary built from a mixed sample. The three enterprise exit routines built from these dictionaries could be used for most of the segment types.

Building Dictionaries

IMS supplies a utility, DFSZLDU0, which builds dictionaries from sample data. The sample data must be in a sequential data set with variable length records. You may use an output file from the HD Database Unload utility (DFSURGU0), however, all segments in the unload file will be used to build the dictionary. Alternatively, you could produce a file with sample data. The IMS Hardware Data Compression Extended product provides the capability to build dictionaries from Image Copy and Unload files. It includes an option to select only instances of specified segments.

There is no need to read megabytes or gigabytes of data when building a dictionary. Dictionaries have a fixed number of entries. After these entries are filled by the build process, subsequent input data is ignored. Typically, a dictionary can be built from 200K to 400K bytes of data.

Key Compression vs. Data Compression

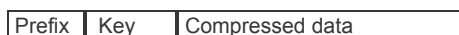
When you implement compression you have the option of specifying whether you want key compression or only data compression. With key compression all the segment past the prefix is compressed. With data compression only the data following the key is compressed. Key compression cannot be used with DEDBs or HISAM root segments. They must use data compression.

The following diagrams show the differences between data compression and key compression:

Typically the key of a segment is in the first field. The following diagrams show compression for these segments.



Segment after compression with data compression:

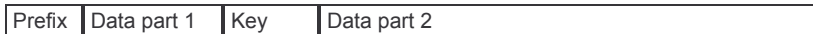


Segment after compression with key compression:



If the key of a segment is not in the first field, the data portion of the segment is split into two parts. One precedes the key. One follows it. The following diagrams show compression for these segments.

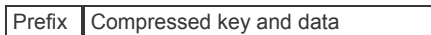
Segment before compression:



Segment after compression with data compression:



Segment after compression with key compression:



Obviously, key compression produces greater space savings. But there is a cost. When a database call needs to examine the key of a segment, the segment must be expanded when the key is compressed. This is not required when only the data is compressed. An example is a get call which is qualified on the key where the segment is in a long twin chain. IMS may have to look at many segments before it finds one that satisfies the call. With key compression this would require the expansion of many segments. With data compression only the segment which satisfies the call would be expanded.

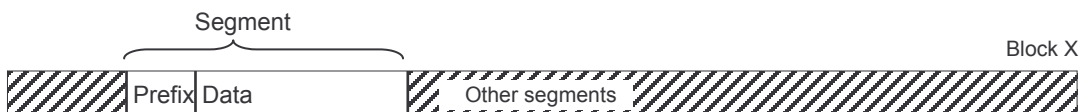
Which compression option you choose should depend on the amount of potential savings from key compression versus the extra processing it requires. This depends on the size of the key, the location of the key in the segment, and the type of processing done against the segment.

Setting a Minimum Size for Segments

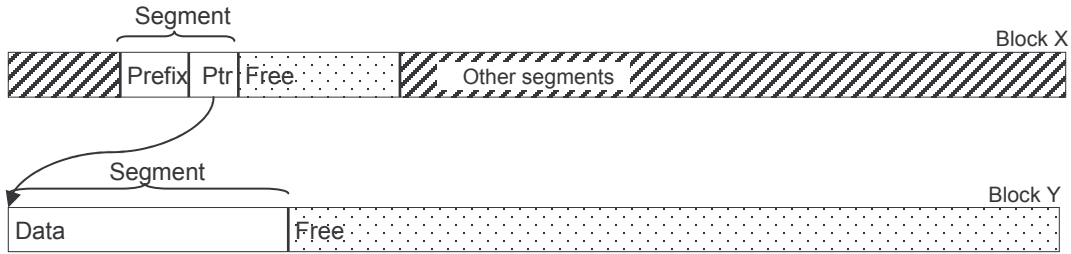
For performance reasons you may want to set the minimum size to which some full function segments will be compressed. Replace calls for full function databases can split the data and prefix parts of a segment. This occurs when the segment grows and there is not sufficient space to contain the larger segment. IMS leaves the prefix in its current location and places the data part elsewhere. A pointer to the data part is added to the prefix. The data part could be placed in another block or CI. Subsequent retrievals of the segment might require an additional read. This does not occur with Fast Path DEDBs. Replace calls for DEDBs never split a segment. If the segment will not fit at its current location, the entire segment is moved to another location.

The following diagrams illustrate an example of a replace call for a full function database.

This diagram shows the segment before a replace call. The space following the segment in this block is occupied by another segment.



The replace call increases the size of the segment. It will not fit in the available space. The data is moved to block Y. A pointer is added after the prefix to point to the data part of the segment. This diagram shows the segment after it is split and placed in two blocks.



Splitting of segments may be reduced or eliminated by setting a minimum size for the segment. If compression reduces it below the minimum, IMS pads the compressed segment with extra bytes. This allows subsequent replaces which do not expand the segment over the minimum size to store the segment without splitting the prefix and data parts.

The following diagrams illustrate the use of a minimum size for a segment.

This diagram shows the segment before a replace call. The segment is padded to the minimum size.



The replace call increases the size of the segment as seen by the application program. It does not increase it beyond the minimum size. The segment can still be stored in the same place. It does not have to be split. This diagram shows the segment after this replace call.



The minimum size for full function segments is specified differently for fixed length and variable length segments. For fixed length segments, the minimum size is specified on the COMPRTN= parameter. The fourth subparameter is the minimum size when PAD is specified for the fifth subparameter. For variable length segments, the minimum size is specified by the second subparameter of the BYTES= parameter. These definitions are explained below.

Since replace calls for DEDBs never split the segment, it is not necessary to set a minimum compressed size for DEDB segments.

Adjusting HDAM, PHDAM, and DEDB Parameters

Compression will reduce the space required for database records. Since less space is required, you may want to adjust the other space parameters for your databases. This is especially applicable to HDAM, PHDAM, and DEDB databases.

HDAM and PHDAM

For HDAM and PHDAM you should examine the parameters used for the size of the root addressable area (RAA), the number of root anchor points (RAP) per block, and the maximum number of bytes inserted for a database record. These are specified in the RMNAME parameter of the DBD statement. For PHDAM they may also be specified in the corresponding parameters for the HALDB Partition Definition utility and the DBRC INIT.PART and CHANGE.PART commands.

You may want to use a smaller root addressable area. If you do not, sequential scans of the database will read the same number of blocks in the RAA.

If you decrease the size of the RAA, you may want to increase the number of RAPs per block. This would allow you to have the same number of roots per RAP and keep the size of synonym chains on a RAP the same.

If you have specified the 'bytes' parameter, you should understand why the previous value was chosen. It may need to be adjusted. It was probably specified to preserve space in the RAA for other database records. It might not be needed with compression. More likely, you will want to keep it but will need to adjust it for the changing space requirements of database records and the space available for them in the RAA.

DEDB

For DEDBs you should examine the parameters used for the size of the root addressable parts and allocation of Units of Work (UOWs). These are specified in the ROOT and UOW parameters of the AREA statement. The compressed segments may reduce the requirement for CIs in the root addressable parts, dependent overflow, or independent overflow. The specification of the size of the root addressable parts is especially important. These parameters determine the minimum number of CIs read by applications which scan an area.

You may want to specify a smaller number of UOWs for the root addressable part, for independent overflow, or both. These are specified on the ROOT parameter of the AREA statements.

You may want smaller UOWs. In a UOW you may want a smaller number of CIs used for dependent overflow. These are specified on the UOW parameter of the AREA statement.

Defining Compression for a Database Segment

SEGM Statement

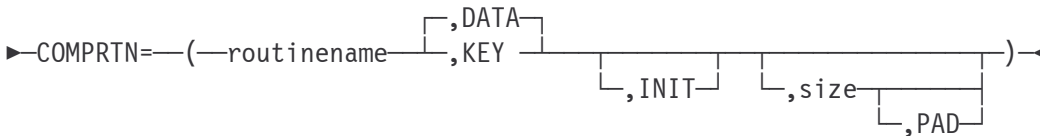
When adding compression for a segment, you must specify the COMPRTN= parameter on the SEGM statement in the DBD. You may want to modify the BYTES= parameter also.

COMPRTN=

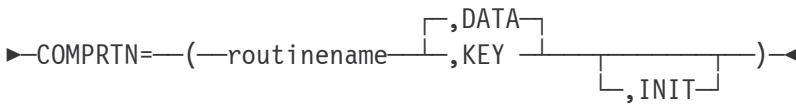
Segment edit/compression routines are defined for a segment on the SEGM statement in the DBD. The COMPRTN= parameter identifies the name of the compression routine and other parameters.

The syntax of the COMPRTN= parameter is the following.

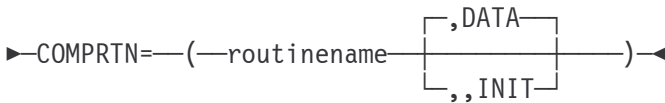
Full function fixed length segments



Full function variable length segments



Fast Path DEDB segments



The COMPRTN= parameter is used to specify a segment edit/compression routine. There are five positional subparameters.

routineName

The first subparameter, routineName, identifies the name of the routine.

DATA and KEY

The second subparameter indicates if all of the segment, including the key, will be compressed or if only the data past the key will be compressed. DATA is the default. It indicates that only the data past the key will be compressed. KEY indicates that the key and all of the data will be compressed. KEY cannot be specified for DEDB segments or HISAM root segments.

INIT

The third subparameter, INIT, is optional. It indicates that the routine will be driven at initialization and termination. This allows the routine to do some special processing, such as loading a table at initialization and deleting it at termination. Hardware data compression implementations require the specification of INIT.

size and PAD

The fourth and fifth subparameters are valid only for fixed length full function databases. The fourth subparameter has two possible meanings. It is either an “increment” size or a “PAD” size. The fifth subparameter determines the meaning of the fourth subparameter. If the fifth subparameter is omitted, the fourth subparameter is an increment size. If PAD is specified for the fifth subparameter, the fourth subparameter is a PAD size.

An increment size should not be specified. The following is an explanation of its meaning and why it should not be specified. An increment size is the number of bytes that the routine may add to a fixed length segment’s size. This capability is provided because a compression routine algorithm could increase the size of a segment instead of decreasing it. This is explained below under “Why Compression Routines Might Not Compress”. Increment values of 1 to 32,767 may be specified. If an increment size less than 10 is specified or if no value is specified, an increment size of 10 is used. When PAD is specified, an increment size of 10 is used. The increment size is not required for any compression routines supplied by IMS or IBM products, such as IMS Hardware Data Compression Extended. They never add more than one byte to a segment. Specifying increment sizes is rarely required for any routines since most compression routines never increase a segment’s size by more than 10 bytes.

A PAD size is used to specify a minimum size that the segment will be when written to DASD. When compression reduces the segment’s length to less than the PAD size, IMS pads the segment to the PAD size before writing it to DASD. PAD sizes are never required, but may be recommended for performance reasons. This is explained above under “Setting a Minimum Size for Segments”.

BYTES= parameter

Fixed Length Segments

►BYTES=bytes◄

Variable Length Segments

►BYTES=(maxbytes,minbytes)◄

The size of segments is specified on the BYTES= parameter of the SEGM statements.

Full function fixed length segments

When adding a segment edit/compression exit routine for a full function fixed length segment, it is never necessary to change the BYTES= parameter on the SEGM statement for the segment. Some editions of IMS manuals have implied that a change is required. This is not true.

Full function variable length segments

When adding compression for a full function variable length segment, it is rarely necessary to increase the maximum number of bytes (maxbytes) specified in the BYTES= parameter on the

SEGM statement for the segment. This depends on the existing BYTES= specifications. In highly unusual circumstances, a compression routine might increase the size of a segment. The IMS supplied sample, DFSCMPX0, and IMS created hardware data compression routines add, at most, one byte to the size of the segment created by the application program. The maximum size specification should be increased only if the application actually creates segments of the maximum size. Even in this case, it is rarely necessary to increase the maximum segment size. See the section entitled “Why Compression Routines Might Not Compress” for an explanation of why all compression routines might increase the size of a segment.

When adding compression for a full function variable length segment, it may be desirable to decrease the minimum number of bytes (minbytes) specified in the BYTES= parameter. If the exit routine compresses the segment to a size smaller than the minimum, IMS pads it to the minimum size. You may not want to decrease the minbytes specification to the smallest size that the compression routine may create. This is explained above under “Setting a Minimum Size for Segments”.

Fast Path DEDB variable and fixed length segments

When adding a segment edit/compression exit routine for a Fast Path DEDB variable length segment or fixed length segment, it is never necessary to change the BYTES= parameter on the SEGM statement for the segment. Fast Path allows the segment to be reduced to any size as long as the key is not compressed or modified. The segment is not padded to a minimum size. As explained below under “Why Compression Routines Might Not Compress”, an exit routine might increase the size of the segment by one byte in unusual circumstances. Fast Path allows this without requiring a change in the maximum size specification.

Why Compression Routines Might Not Compress

Normally, compression routines make segments smaller. That’s why we use them. Nevertheless, compression algorithms could expand some segments. These would typically be segments with highly unusual content. IMS’s hardware data compression routines replace a variable number of bytes with a combination of 12 bits. Frequently used byte strings get very good compression. For example, “San Jose”, which is eight bytes, could be compressed to 12 bits. On the other hand, an infrequently used byte might be expanded to 12 bits. An example might be x’EB’. So, the algorithm might expand a segment which has a lot of atypical data.

Routines supplied by IMS, such as those generated by IMS’s hardware data compression capability, place a flag byte at the beginning of the compressed segment. If the algorithm cannot shorten the segment, the original segment contents are placed after the flag byte and the flag byte is set to a “not compressed” value. This adds one byte to the segment length. If the algorithm can shorten the segment, the compressed data is placed after the flag byte which is set to a “compressed” value. Thus, these routines will never add more than one byte to the length of a segment. Adding the byte should be highly unusual. In many installations, it will never occur.

You should be aware that all compression routines might expand some segments. It is impossible to write a compression routine that can compress some data and never expand any data. Good compression routines minimize the possibility of expanding a segment. They cannot eliminate it.

Warning About Compression Routine Abends

You should be aware that an abend of a compression routine in an online system typically causes an abend of the control region. Compression routines are executed as part of DL/I calls. If they abend, the application program abends during the DL/I call. This causes a U0113 abend of the control region.

Normally, well tested exit routines do not abend. So, this should not be a problem. On the other hand, misuse of an exit routine could cause this problem. An example is attempting to use an exit routine to expand data which has not been compressed. This can occur when the load step for a database uses a DBD which does not specify compression. The data is not compressed. Later, the database is accessed from the online system using a DBD with compression specified. Obviously, this is an error. When the exit routine attempts to expand the uncompressed data, it may encounter unexpected conditions forcing an abend. Change procedures at most installations would prevent this from occurring in a production system. It is more likely to occur with test systems where change procedures are not as restrictive.

Implementation Steps for Hardware Data Compression

To implement hardware data compression for a database, follow these steps:

1. Evaluate which segments you want to compress. Since adding or changing the compression routine for a segment requires an unload and reload of the database, it is best to evaluate all of the segments in a database together. Then, compression for multiple segments may be added or changed with one unload and reload.
2. For each segment, decide if it will use a generic dictionary, an enterprise dictionary, or a customized dictionary.
3. Create the dictionaries by using the Hardware Data Compression Dictionary utility (DFSZLDU0).
4. Link-edit each dictionary to the IMS-supplied base exit routine. This produces a segment edit/compression routine for each dictionary.
5. Unload the database using the old DBD.
6. Create the new DBD specifying the new exit routine. Specify the newly created segment edit/compression routines in the COMPRTN= parameter of the DBDGEN SEGM statements. Specify padding for full function database segments, if desired. Do this by specifying PAD and 'size' in the COMPRTN= parameter for fixed length segments or adjusting the 'minbytes' value for variable length segments.
7. Run ACBGEN for the new DBD.
8. Reload the database using the new DBD.
9. Take Image Copies of the database data sets.

The IMS Hardware Data Compression Extended product assists in these steps.

- It can evaluate the potential compression from building a customized dictionary versus using another existing dictionary or the generic dictionary which the product supplies.
- It invokes the processes which build dictionaries and link-edits them into exit routines.

- It builds the jobs which unload and reload databases.

Application programs are not changed when you implement compression. The segments in program I/O areas remain the same. Compression changes the segment as it is stored in OSAM blocks or VSAM CIs. It does not change the view of the segment seen by application programs.