

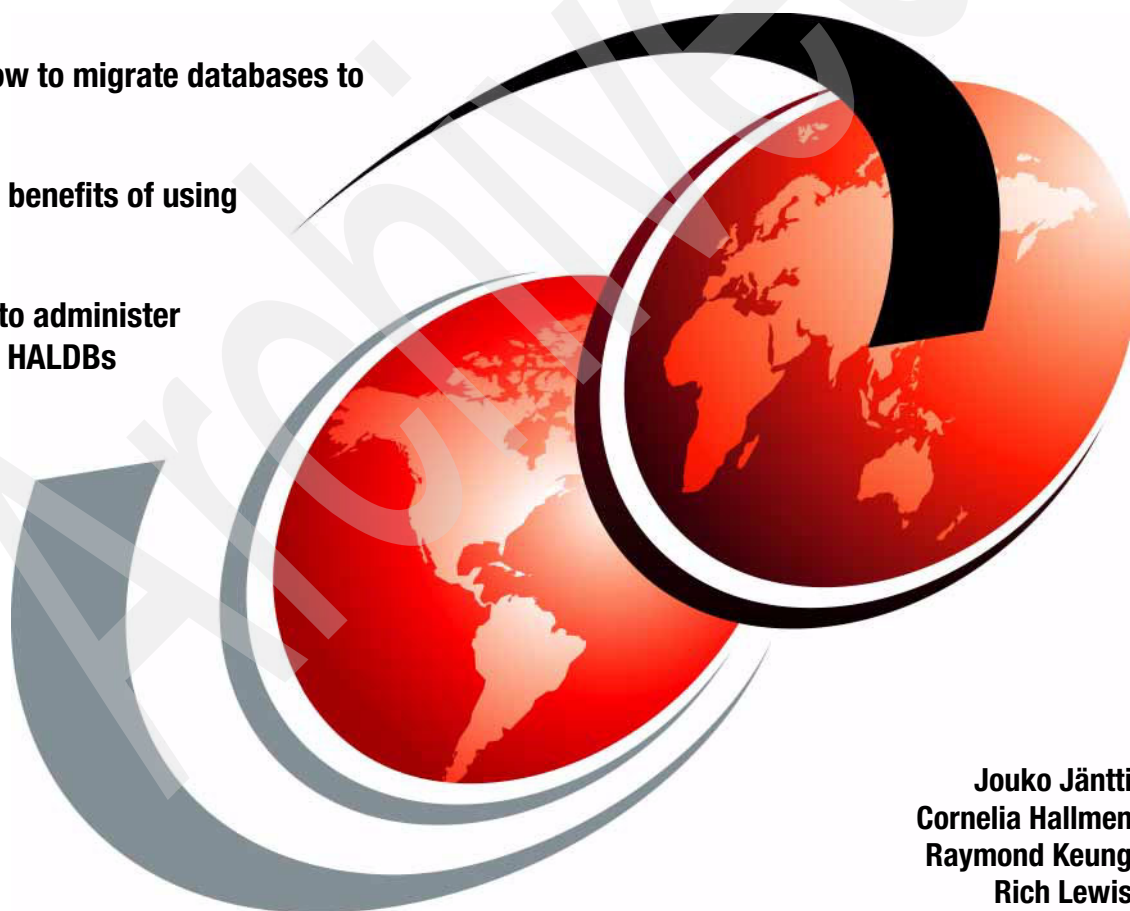
# The Complete IMS HALDB Guide

## All You Need to Know to Manage HALDBs

Examine how to migrate databases to  
HALDB

Explore the benefits of using  
HALDB

Learn how to administer  
and modify HALDBs



Jouko Jäntti  
Cornelia Hallmen  
Raymond Keung  
Rich Lewis





International Technical Support Organization

**The Complete IMS HALDB Guide  
All You Need to Know to Manage HALDBs**

June 2003

Archived

**Note:** Before using this information and the product it supports, read the information in “Notices” on page xi.

### **First Edition (June 2003)**

This edition applies to IMS Version 7 (program number 5655-B01) and IMS Version 8 (program number 5655-C56) or later for use with the OS/390 or z/OS operating system.

© Copyright International Business Machines Corporation 2003. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

# Contents

<b>Notices</b> .....	xi
Trademarks .....	xii
<b>Preface</b> .....	xiii
The team that wrote this redbook .....	xiii
Become a published author .....	xiv
Comments welcome .....	xv
<b>Part 1. HALDB overview</b> .....	1
<b>Chapter 1. HALDB introduction and structure</b> .....	3
1.1 An introduction to High Availability Large Databases .....	4
1.2 Features and benefits .....	5
1.3 Candidates for HALDB .....	6
1.4 HALDB definition process .....	7
1.4.1 Database Recovery Control (DBRC) .....	9
1.5 DL/I processing .....	9
1.6 Logical relationships with HALDB .....	10
1.7 Partition selection .....	10
1.7.1 Partition selection using key ranges .....	10
1.7.2 Partition selection using a partition selection exit routine .....	11
1.8 Database structure .....	11
1.8.1 Partition ID and reorganization number .....	11
1.8.2 Segment structure .....	12
1.8.3 Extended pointer set (EPS) .....	14
1.8.4 Indirect list data set (ILDS) .....	14
1.8.5 Number of data sets .....	15
1.8.6 Special considerations for secondary indexes .....	16
1.9 Naming conventions .....	17
1.9.1 Partition names .....	18
1.9.2 DD names .....	18
1.9.3 Data set names .....	18
<b>Chapter 2. Defining HALDB databases</b> .....	21
2.1 Overview of HALDB definition .....	22
2.1.1 Design the logical structure of the database .....	22
2.1.2 Implement the logical structure with the DBDGEN process .....	22
2.1.3 Determine the partitioning scheme .....	23
2.1.4 Create the partitioning scheme .....	23

2.1.5 Database exit routines . . . . .	24
2.1.6 System definition . . . . .	25
2.1.7 Buffer pools . . . . .	25
2.1.8 Dynamic allocation . . . . .	26
2.2 DBDGEN differences for HALDB . . . . .	26
<b>Chapter 3. HALDB and DBRC . . . . .</b>	<b>31</b>
3.1 RECON records for HALDB . . . . .	32
3.1.1 Master database record . . . . .	32
3.1.2 Partition database record . . . . .	33
3.1.3 Partition record . . . . .	33
3.1.4 Partition DBDS record . . . . .	34
3.2 Dynamic allocation . . . . .	35
3.3 Authorization processing . . . . .	35
3.4 Partition initialization . . . . .	35
3.5 DBRC commands . . . . .	35
3.5.1 INIT commands . . . . .	36
3.5.2 CHANGE commands . . . . .	36
3.5.3 DELETE commands . . . . .	38
3.5.4 LIST commands . . . . .	38
3.5.5 GENJCL commands . . . . .	40
3.6 DBRC groups for HALDB . . . . .	42
3.6.1 Change accumulation groups . . . . .	42
3.6.2 Database data set groups . . . . .	42
3.6.3 Database groups . . . . .	42
3.6.4 Recovery groups . . . . .	42
3.7 Use of database names in DBRC commands . . . . .	43
3.7.1 Commands that require a master database name . . . . .	43
3.7.2 Commands that require a partition name . . . . .	43
3.7.3 Commands that allow a master database or a partition name . . . . .	44
3.7.4 DBRC commands that are not allowed with HALDB . . . . .	44
<b>Chapter 4. Partition Definition utility . . . . .</b>	<b>45</b>
4.1 Using the PDU . . . . .	46
4.1.1 Configuring the PDU . . . . .	47
4.1.2 Selecting a database . . . . .	49
4.1.3 Setting HALDB master DBD parameters . . . . .	51
4.1.4 Setting processing options and global partition information . . . . .	52
4.1.5 Creating your partitions manually . . . . .	54
4.1.6 Creating your partitions automatically . . . . .	57
4.1.7 Changing partition definitions . . . . .	59
4.1.8 Deleting definitions . . . . .	60
<b>Chapter 5. Batch definition of HALDB . . . . .</b>	<b>61</b>

5.1 Using the batch interface . . . . .	62
5.2 DBRC initialization commands for HALDB . . . . .	62
5.2.1 INIT.DB . . . . .	62
5.2.2 INIT.PART . . . . .	64
5.3 DBRC change commands for HALDB . . . . .	68
5.3.1 CHANGE.DB . . . . .	69
5.3.2 CHANGE.PART . . . . .	69
5.3.3 CHANGE.DBDS . . . . .	70
5.4 DBRC delete commands for HALDB . . . . .	71
5.4.1 DELETE.DB . . . . .	71
5.4.2 DELETE.PART . . . . .	71
<b>Chapter 6. Partition initialization . . . . .</b>	<b>73</b>
6.1 Partition initialization function . . . . .	74
6.2 DBRC flags used with partition initialization . . . . .	75
6.2.1 Partition initialization required (PINIT) flag . . . . .	75
6.2.2 Image copy needed flag . . . . .	76
6.3 Database Preorganization utility . . . . .	76
6.4 Database Partition Data Set Initialization utility . . . . .	76
6.4.1 Unconditional partition initialization . . . . .	78
<b>Chapter 7. Partition selection . . . . .</b>	<b>79</b>
7.1 Choosing the type of partition selection . . . . .	80
7.1.1 Key range partitioning . . . . .	80
7.1.2 Partitioning with an exit routine . . . . .	81
7.2 Writing partition selection exit routines . . . . .	82
7.2.1 Partition selection exit routine functions . . . . .	82
7.2.2 Information passed to and from the exit routine . . . . .	84
7.2.3 Sample exit routine (DFSPSE00) . . . . .	84
<b>Part 2. Migration . . . . .</b>	<b>87</b>
<b>Chapter 8. Migration from non-HALDB to HALDB . . . . .</b>	<b>89</b>
8.1 General migration considerations . . . . .	90
8.1.1 DBD changes . . . . .	90
8.2 HALDB Migration Aid utility (DFSMAID0) . . . . .	93
8.2.1 Reports . . . . .	93
8.2.2 Adjusting the sizes from the reports . . . . .	94
8.2.3 Control statements . . . . .	94
8.3 Migrating simple databases . . . . .	96
8.3.1 Unloading the existing database . . . . .	96
8.3.2 Saving existing database information . . . . .	97
8.3.3 Changing the DBD . . . . .	97
8.3.4 Deleting database information from the RECONS . . . . .	97

8.3.5	Defining the partitions . . . . .	97
8.3.6	Allocating database data sets . . . . .	98
8.3.7	Initializing the partitions . . . . .	99
8.3.8	Reloading the HALDB database . . . . .	99
8.3.9	Image copying the database data sets . . . . .	100
8.4	ILDS creation options . . . . .	100
8.4.1	HD Reload with no control statement . . . . .	100
8.4.2	HD Reload with an ILDSMULTI control statement . . . . .	100
8.4.3	HD Reload with a NOILDS control statement . . . . .	101
8.5	Migrating databases with secondary indexes . . . . .	101
8.5.1	Changing DBDs for secondary indexes . . . . .	102
8.5.2	Estimating the sizes of secondary index partitions . . . . .	107
8.5.3	Unloading databases with secondary indexes . . . . .	109
8.5.4	Allocating secondary index data sets . . . . .	111
8.5.5	Loading databases and their secondary indexes . . . . .	111
8.5.6	Secondary index pointers after the migration . . . . .	113
8.5.7	Using MIGRATE=YES with secondary indexes . . . . .	113
8.6	Migrating databases with logical relationships . . . . .	114
8.6.1	Changing DBDs with logical relationships . . . . .	114
8.6.2	Changing DBDs with logical relationships . . . . .	115
8.6.3	Unloading logically related databases for migration . . . . .	118
8.6.4	Loading logically related databases for migration . . . . .	119
8.6.5	Logical relationship pointers after the migration . . . . .	119
8.7	Migrating from PDB or PDF . . . . .	120
8.7.1	Partition selection . . . . .	120
8.7.2	DBD changes . . . . .	120
8.7.3	Using standard IMS utilities for the migration . . . . .	121
8.7.4	Using application programs for the migration . . . . .	121
8.7.5	Using PSU Unload and application programs for the migration . . . . .	122
8.8	Migrating from user partitioning . . . . .	123
8.8.1	Using application programs for the migration . . . . .	123
8.8.2	Using HD Unload and application programs for the migration . . . . .	125
8.8.3	Warning on the use of HD Reload . . . . .	126
8.9	Fallback to non-HALDB . . . . .	126
8.9.1	Fallback with secondary indexes . . . . .	127
8.9.2	Fallback with logical relationships . . . . .	127
<b>Chapter 9.</b>	<b>Migration examples . . . . .</b>	<b>129</b>
9.1	Migration of HIDAM database to PHIDAM . . . . .	130
9.1.1	Image copy of non-HALDB database data sets . . . . .	130
9.1.2	Running the Migration Aid utility . . . . .	130
9.1.3	Unloading the HIDAM database . . . . .	136
9.1.4	Saving database information . . . . .	137



9.1.5	Deleting the database from the RECONS	137
9.1.6	Changing the DBD and running the DBDGEN	138
9.1.7	Defining partitions with PDU	139
9.1.8	Allocate HALDB data sets	139
9.1.9	Initializing HALDB partitions	140
9.1.10	Loading HALDB database	141
9.1.11	Image copy HALDB data sets	142
9.1.12	ACBGEN	142
9.2	Migration of a database with a secondary index	142
9.3	Migrating a PDB database	147
<b>Chapter 10.</b>	<b>Using the HALDB Conversion and Maintenance Aid</b>	<b>151</b>
10.1	HALDB Conversion and Maintenance Aid product	152
10.1.1	The conversion process overview	153
10.2	Activating the tool	154
10.2.1	Modifying the startup CLIST	155
10.2.2	Allocating the environment description data set	155
10.3	Migration example with HALDB Conversion tool	155
10.3.1	Define an IMS environment	156
10.3.2	Define a project	164
10.3.3	Process a conversion project	168
10.4	Additional functions	176
<b>Part 3.</b>	<b>Application considerations</b>	<b>181</b>
<b>Chapter 11.</b>	<b>Application considerations</b>	<b>183</b>
11.1	Initial loads	184
11.1.1	PHDAM considerations	184
11.1.2	PHIDAM considerations	184
11.1.3	Loading partitions in parallel	185
11.1.4	Initial loads of databases with secondary indexes	186
11.1.5	Initial loads of databases with logical relationships	188
11.2	Processing partitions in parallel	189
11.2.1	DBRC authorization of partitions	189
11.2.2	Parallel processing within an online system	190
11.2.3	Parallel processing with block level data sharing	190
11.2.4	Parallel processing without block level data sharing	190
11.2.5	Program modifications for parallel processing	192
11.3	Restricting a PCB to one partition	193
11.3.1	Processing a partition sequentially	194
11.3.2	Requesting segments outside of the partition	194
11.3.3	Secondary index and logical relationship considerations	195
11.4	Handling unavailable partitions	197
11.4.1	Database PCB status code priming	198

11.5	Processing secondary indexes as databases	198
11.5.1	Secondary indexes with /SX fields	199
11.5.2	Secondary indexes with symbolic pointers	202
11.6	Handling test environments	205
11.6.1	DBRC registration	205
11.6.2	Testing with non-HALDB databases	206
11.7	Copying databases to different environments	207
11.7.1	The partition ID	207
11.7.2	Using image copies	207
11.7.3	Using unloads	208
11.7.4	Copying part of the data	208
11.7.5	Copying partition definitions	208
11.7.6	Timestamp recoveries to a copy	208
11.7.7	Planned enhancement for copying partition definitions	209
<b>Part 4.</b>	<b>Administration</b>	<b>211</b>
<b>Chapter 12.</b>	<b>HALDB online commands</b>	<b>213</b>
12.1	Online commands with HALDB	214
12.2	/DISPLAY command	214
12.2.1	Display of a HALDB master database	215
12.2.2	Display of a HALDB partition	215
12.3	/DBRECOVERY command	217
12.3.1	DBR of a HALDB master database	217
12.3.2	DBR of a HALDB partition	217
12.4	/DBDUMP command	218
12.4.1	DBD of a HALDB master database	218
12.4.2	DBD of a HALDB partition	219
12.5	/START command	219
12.5.1	Start of a HALDB master database	219
12.5.2	Start of a HALDB partition	219
12.6	/STOP command	220
12.6.1	Stop of a HALDB master database	220
12.6.2	Stop of a HALDB partition	220
12.7	/LOCK command	220
12.7.1	Lock of a HALDB master database	220
12.7.2	Lock of a HALDB partition	221
12.8	/UNLOCK command	221
12.8.1	Unlock of a HALDB master database	221
12.8.2	Unlock of a HALDB partition	221
12.9	Using the ALL keyword with commands	221
12.10	Commands while databases are in use	221
12.11	Command examples	222

<b>Chapter 13. Backup and recovery</b>	227
13.1 Backup and recovery overview	228
13.2 Image copies for HALDB data sets	228
13.2.1 Database Image Copy utility (DFSUDMP0) example	229
13.2.2 Database Image Copy 2 utility (DFSUDMT0) example	229
13.2.3 Using GENJCL to create image copy JCL	230
13.3 Recoveries of HALDB data sets	230
13.3.1 Change Accumulation utility with HALDB	231
13.3.2 Database Recovery utility and ORS with HALDB	231
13.4 Rebuilding ILDS and PHIDAM index data sets	236
13.4.1 Performance of the Index/ILDS Rebuild utility	237
13.4.2 Using GENJCL.USER to generate Index/ILDS Rebuild JCL	238
13.5 Recovering HALDB secondary index data sets	239
13.5.1 Using IMS Index Builder for recoveries	241
13.6 Batch Backout utility with HALDB	241
<b>Chapter 14. HALDB reorganization</b>	243
14.1 The reorganization process	244
14.1.1 Non-HALDB reorganization overview	244
14.1.2 HALDB reorganization overview	245
14.1.3 Options for reorganization	246
14.1.4 Unloading HALDB partitions and databases	246
14.1.5 Reallocating HALDB database data sets	247
14.1.6 Reloading HALDB partitions and databases	249
14.1.7 Reorganizing HALDB secondary indexes	251
14.2 The self-healing pointer process	251
14.2.1 The elements of the self-healing process	251
14.2.2 Finding target segments	253
14.2.3 Healing pointers	254
<b>Chapter 15. Changing existing HALDB databases</b>	257
15.1 Changing database definitions	258
15.1.1 Changes not requiring unload and reload of the database	258
15.1.2 Changes requiring unload and reload of the database	258
15.1.3 Using online change with HALDB	258
15.1.4 Adding a secondary index	259
15.2 Changing partition definitions	259
15.2.1 Changing high key definitions	260
15.2.2 Making changes with a partition selection exit routine	265
15.2.3 Changing partition names	267
15.2.4 Partition ID numbers	267
15.3 Changing secondary indexes	268
15.4 Changing logically related databases	269

<b>Chapter 16. Using IMS tools for HALDB</b> .....	271
16.1 IMS High Performance Unload .....	272
16.2 IMS High Performance Load .....	273
16.3 IMS Index Builder .....	275
16.4 IMS Image Copy Extensions .....	276
<b>Part 5. Appendices</b> .....	279
<b>Appendix A. Sample GENJCL.USER skeletons for HALDB allocation</b> ..	281
Overview .....	282
Creation of skeletal JCL .....	282
Index skeletal JCL member .....	283
ILDS skeletal JCL member .....	284
VSAM data skeletal JCL member .....	284
OSAM data skeletal JCL member .....	285
Defaults member .....	286
Definition of DBDS groups .....	286
Execution of GENJCL.USER commands .....	288
Execution of allocation jobs .....	289
<b>Appendix B. HALDB functions added by APARs</b> .....	291
HALDB functions delivered by APARs .....	292
<b>Appendix C. Summary of HALDB utilities</b> .....	293
IMS database utilities for HALDBs .....	294
<b>Abbreviations and acronyms</b> .....	297
<b>Related publications</b> .....	299
IBM Redbooks .....	299
Other publications .....	299
Online resources .....	300
How to get IBM Redbooks .....	300
<b>Index</b> .....	301

# Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

*IBM Director of Licensing, IBM Corporation, North Castle Drive Armonk, NY 10504-1785 U.S.A.*

*The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.*

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.


## COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

# Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

CICS®  
FICON™  
IBM®  
ibm.com®  
IMS/ESA®  
IMS™

Language Environment®  
MVS™  
OS/390®  
Redbooks (logo) ™ FICON™  
Redbooks™  
S/390®

Sequent®  
SP1®  
SP™  
server™  
WebSphere®  
z/OS™

The following terms are trademarks of other companies:

Partitioned Database Facility (PDF) is a trademark of Neon Enterprise Software, Inc. in the United States, other countries, or both.

ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

C-bus is a trademark of Corollary, Inc. in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

SET, SET Secure Electronic Transaction, and the SET Logo are trademarks owned by SET Secure Electronic Transaction LLC.

Other company, product, and service names may be trademarks or service marks of others.

# Preface

This IBM Redbook describes the High Availability Large Database (HALDB) capability available with IMS™. IMS HALDB was introduced with IMS Version 7. It allows IMS databases to grow to almost unlimited sizes while providing increased availability. This book updates *IMS Version 7 High Availability Large Database Guide*, SG24-5751, as well as adds topics that were not covered in the previous book.

This publication provides a broad explanation of HALDB and its uses. Specific areas covered include:

- ▶ HALDB overview, definition, and structure
- ▶ Migration from non-HALDB databases
- ▶ Application considerations
- ▶ HALDB database administration

This publication documents our hands-on experience in a test environment. It includes migration and administration examples. Some IBM Data Management Tools for IMS are also discussed in this book. Special emphasis is given to the IMS HALDB Conversion and Maintenance Aid product. Examples of the use of these tools are provided.

## The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization, San Jose Center.

**Jouko Jäntti** is a Project Leader specializing in IMS with the IBM International Technical Support Organization, San Jose Center. He holds a Bachelors degree in Business Information Technology from Helsinki Business Polytechnic, Finland. Before joining the ITSO in September 2001, Jouko worked as an Advisory IT Specialist at IBM Global Services, Finland. Jouko has been working on several e-business projects with customers as a specialist in IMS, WebSphere®, and UNIX on the OS/390® platform. He has also been responsible for the local IMS support for Finnish IMS customers. Prior to joining IBM in 1997, he worked as a Systems Programmer and Transaction Management Specialist in a large Finnish bank for 13 years, and was responsible for the bank's IMS systems.

**Cornelia Hallmen** is an IMS instructor in Munich, Germany. She has over 11 years of experience as an IMS Systems Programmer in a large German bank

and was responsible for its IMS systems, especially for high availability and for the implementation of new versions and functions. She joined IBM in 2000 as an instructor for IMS databases.

**Raymond Keung** is a Systems Analyst/IMS Database Administrator in Toronto, Canada. He has five years of experience in IMS. He has worked at IBM for 14 years. His areas of expertise include technical support for IMS upgrades and software installation for database conversions. He is also responsible for performing IMS database administrative tasks and maintaining data and structural integrity of production databases.

**Rich Lewis** is a Senior Consulting IT Specialist with IBM Advanced Technical Support, Americas, located in Dallas, Texas. He has provided IMS technical support and consulting services as a member of the Dallas Systems Center for over 20 years. His areas of expertise include IMS database design, performance, and availability. He has developed and taught IMS classes for IBM and is a frequent speaker at IMS technical conferences, user group meetings, road shows, and seminars.

Thanks to the following people for their contributions to this project:

**Rich Conway, Bob Haimowitz, Julie Czubik**  
International Technical Support Organization, Poughkeepsie Center

**Rose Levin, Becky Bard, Harley Beier, Claudia Ho, John Langley, Rick Long, Elmer Rohde, Pete Sadler, Pat Schroeck, Larry Sullivan, Vern Watts**  
IBM Silicon Valley Laboratory, San Jose, USA

**Christian Koeppen**  
Peck-Software, USA

**Rick Long, Ken Daugherty, Jim Kelly, Geoff Nicholls, Norbert Theurich**  
The authors of *IMS Version 7 High Availability Large Database Guide*,  
SG24-5751

## Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You'll team with IBM technical professionals, Business Partners and/or customers.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you'll develop a network of contacts in IBM development labs, and increase your productivity and marketability.



Find out more about the residency program, browse the residency index, and apply online at:

[ibm.com®/redbooks/residencies.html](http://ibm.com®/redbooks/residencies.html)

## Comments welcome

Your comments are important to us!

We want our Redbooks™ to be as helpful as possible. Send us your comments about this or other Redbooks in one of the following ways:

- ▶ Use the online **Contact us** review redbook form found at:

[ibm.com/redbooks](http://ibm.com/redbooks)

- ▶ Send your comments in an Internet note to:

[redbook@us.ibm.com](mailto:redbook@us.ibm.com)

- ▶ Mail your comments to:

IBM Corporation, International Technical Support Organization  
Dept. QXXE Building 80-E2  
650 Harry Road  
San Jose, California 95120-6099





# Part 1

# HALDB overview

In this part we discuss the benefits of using HALDB databases. We identify which databases are candidates for migration to HALDB. We describe the structure of HALDB databases and their partitions. We explain how you define HALDB databases and how to initialize their partitions. We describe the ways in which HALDB can assign database records to partitions.

Archived

# HALDB introduction and structure

High Availability Large Database (HALDB) was introduced with IMS Version 7 to extend the functions provided by IMS full-function databases. As the name suggests, HALDB is designed to increase the capacity and availability of full-function databases. HALDB has three database types:

<b>PHDAM</b>	Partitioned HDAM
<b>PHIDAM</b>	Partitioned HIDAM
<b>PSINDEX</b>	Partitioned secondary index

HALDB support includes the following:

- ▶ OSAM and VSAM (ESDS and KSDS) access methods
- ▶ Logical relationships and secondary indexes
- ▶ Data sharing
- ▶ Remote Site Recovery (RSR)
- ▶ Extended Recovery Facility (XRF)
- ▶ Online change
- ▶ OSAM sequential buffering
- ▶ IMS Monitor
- ▶ IMS Performance Analyzer product

This chapter provides an overview of HALDB and its structure.

# 1.1 An introduction to High Availability Large Databases

The capacity of a non-HALDB full-function database is limited by the number of data sets allowed in a database and their maximum sizes. The capacity of a data set is either 4 GB if it is VSAM or 8 GB for OSAM. Up to ten data sets are allowed. All segments of the same type must be in the same data set. So, when one data set is full, the database is full.

HALDB removes this limit by removing the restriction that all occurrences of the same segment type must be in the same data set. HALDB groups database records into sets of partitions. The hierarchic structure is maintained within a partition and a whole database record always resides in a single partition.

Partitions may be very large. Each partition may have up to ten data sets. A database may have up to 1001 partitions, so segments in a HALDB database may be stored in up to 10,010 data sets. Each HALDB data set may be up to 4 GB, so a HALDB database may contain over 40 terabytes of data.

HALDB also has availability and manageability benefits. Partitions may be managed, allocated, authorized, and allocated independently. Some partitions may be available to an online system while others are being processed by utilities. This increases the flexibility available in handling databases.

HALDB has several benefits:

<b>Capacity</b>	It allows a single database to be very large.
<b>Manageability</b>	It allows for the division of a database into small partitions that are easier to manage than the entire database.
<b>Availability</b>	It allows different subsystems or utilities access to manage partitions independently.
<b>Usability</b>	It provides for increased use of online processes for definitions.
<b>Compatibility</b>	It provides application programming compatibility with non-HALDB databases.

These benefits are delivered with application compatibility. Only in rare cases are application programming changes required when a database is migrated to HALDB.

## 1.2 Features and benefits

HALDB addresses a number of limitations associated with non-HALDB full-function databases. Non-HALDB database limitations include:

- ▶ Storage capacity for each database is limited by the maximum size of its data sets. These sizes are 4 GB for VSAM and 8 GB for OSAM.
- ▶ Limited options are available for increasing the storage capacity of a single database.
- ▶ Availability of the database is not granular. The entire database is either available or unavailable.
- ▶ Secondary index databases with direct pointers must be rebuilt when their indexed databases are reorganized.
- ▶ Logical relationships require updates to pointers in related databases when a database is reorganized. Utility executions are required to make these updates.
- ▶ Increasing the size of a database requires the reorganization of the entire database. This may require an extended outage.

Each of these limitations is specifically addressed by HALDB while maintaining application compatibility.

HALDB increases the storage capacity of full-function databases by providing the ability to partition each database. Each database can have up to 1001 partitions, and each partition can have up to 10 data set groups, each of which can be up to 4 GB. Application programs access the database without needing to understand the partitioning of the database.

HALDB improves the manageability of databases. Large databases can be split into several partitions with small data sets. These smaller data sets may be easier to handle. This is particularly true for databases that previously required multi-volume data sets. You can create multiple small partitions such that each data set fits on a single volume. This can simplify database data set management.

Availability of data can also be improved with HALDB. Allocation, authorization, reorganization, and recovery can be done independently by partition. In the event that a partition is unavailable, programs can continue to access the partitions that are available. A status code can be returned to a program to indicate that data are presently unavailable. In this way, much of the database can remain available even when parts of it are offline. Similarly, reorganization of the database can be run independently for each partition. Only those partitions of the database that are being reorganized are unavailable. Online systems and batch jobs may continue to access the rest of the partitions.

Secondary indexes can be managed independently from their indexed databases. Logically related databases can be managed independently from each other. The reorganization of a partition or database does not require changes to secondary indexes or logically related databases. HALDB's self-healing process for these pointers provides this independence.

## 1.3 Candidates for HALDB

HALDB can be used to replace any existing HDAM or HIDAM database and its secondary indexes. The migration process for these databases is similar to an ordinary reorganization process. The database is unloaded into a sequential data set, the database is redefined as a HALDB, and the database is reloaded as a HALDB database. Application programs can then access the new HALDB database. In almost all cases, no changes are required to existing application programs.

The following list categorizes databases that are candidates for conversion to HALDB:

- ▶ Very large databases

The most obvious candidates for HALDB are the databases that are approaching the 4 GB (VSAM) or 8 GB (OSAM) data set limitation. Their conversion to HALDB provides for future growth and makes them more manageable.

- ▶ Previously partitioned databases

Some of these large databases already may have been partitioned with user partitioning or by products like IMS/ESA® Partition Support Product (PDB). These databases are also candidates for conversion to HALDB.

- ▶ Databases benefitting from improved reorganization processes

HALDB partitioning provides faster reorganizations. This means that the reorganizations may be done more frequently. The self-healing pointer process for secondary indexes and logical relationship eliminates many of the utilities associated with the reorganization of non-HALDB databases. This simplifies their reorganization.

- ▶ Databases benefitting from more free space

The virtually unlimited size of HALDB databases allows users to specify increased free space. This may improve database performance and reduce or eliminate the need to reorganize them.



- ▶ Databases benefitting from parallel processing

Parallel execution of application programs may access the same database. A conversion of the database to HALDB simplifies this processing.

There are advantages to using HALDB even with only one partition:

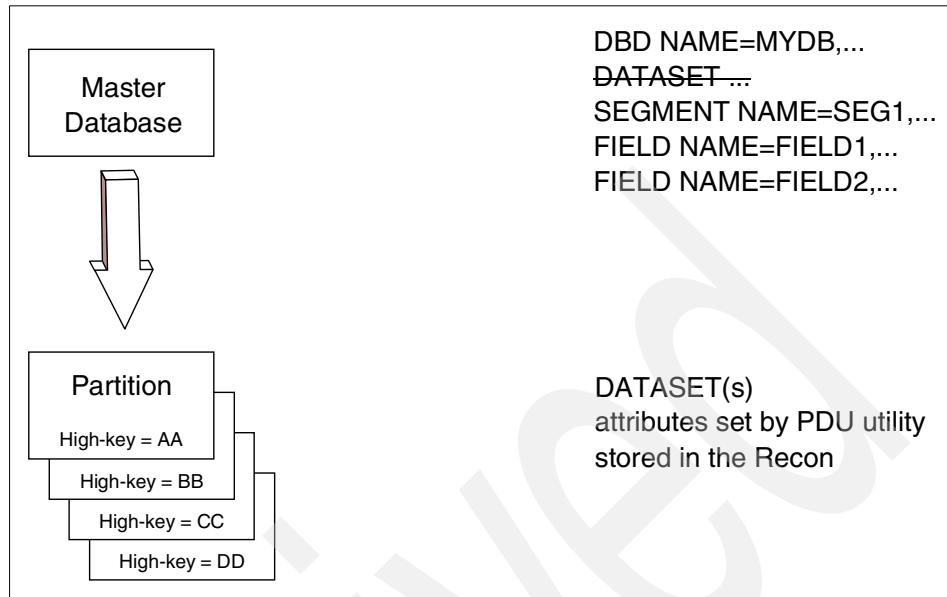
- ▶ The reorganization process for databases with secondary indexes and logical relationships is shortened and simplified.
- ▶ PHIDAM databases are more easily managed than HIDAM databases. The primary index of a PHIDAM database is no longer a separate database.
- ▶ If the database grows, space may be added by simply adding partitions.

### **Future IMS database enhancements**

HALDB support is the basis for future enhancements to IMS full-function databases. Conversions to HALDB may provide both immediate benefits and additional future benefits.

## **1.4 HALDB definition process**

A HALDB definition consists of two distinct entities, the master database definition and its associated partition definitions. The HALDB master database definition is created by the ordinary DBDGEN process. This definition is input to the partition definition process. This is shown in Figure 1-1 on page 8.



*Figure 1-1 Changes to the database definition with HALDB*

The definition of partitions is external to DBDGEN. DBDGEN is used to define the organization of the database including its hierarchic structure, data set group boundaries, pointer options, logical relationships, and secondary indexes. It does not generate the partition definitions or data set information. The HALDB Partition Definition utility (PDU) is an ISPF interface to DBRC. It provides the mechanism for the definition of partitions and data set characteristics. It stores these definitions in the DBRC RECONS. Alternatively, DBRC batch commands can be used to define the partitions and data sets.

Both databases and partitions may be controlled and displayed with online commands. Commands used with non-HALDB databases, such as /START, /STOP, and /DISPLAY, are also used with HALDB databases and partitions.

See Figure 1-2 on page 9 for an illustration of a database before and after conversion to HALDB.

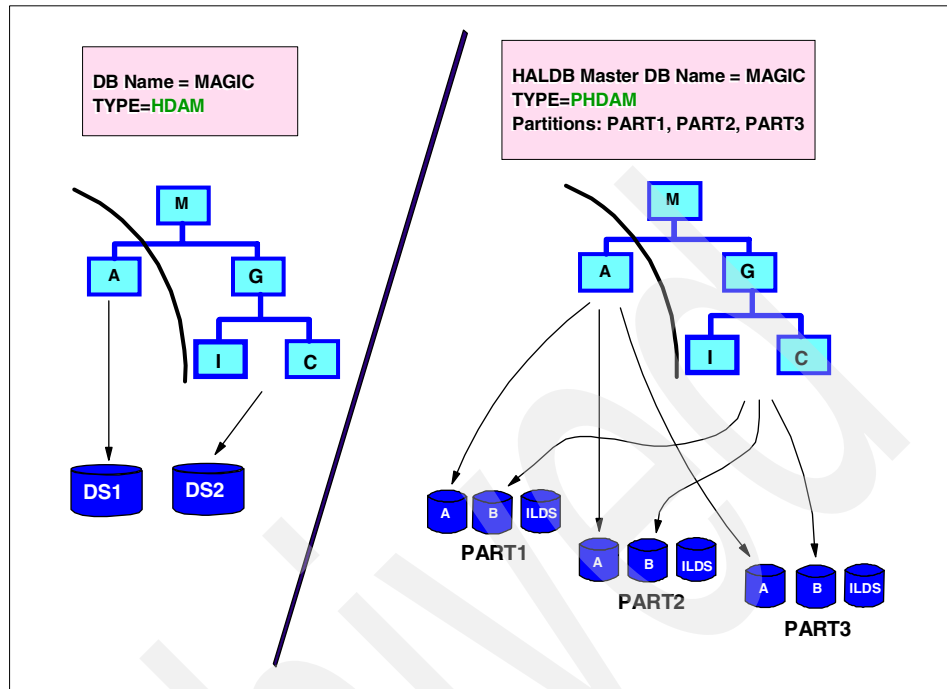


Figure 1-2 IMS database before and after conversion to HALDB

### 1.4.1 Database Recovery Control (DBRC)

Database Recovery Control (DBRC) is required for HALDB. HALDB databases must be registered in the DBRC recovery control data set (RECON) either by use of PDU or DBRC batch commands. DBRC information is also used for dynamic allocation. DFSMDA members are not used.

From DBRC's point of view, a HALDB is a master database consisting of partition databases.

## 1.5 DL/I processing

The application programming interface for HALDB is the same as that used with non-HALDB full-function databases. In general, programs that process a non-HALDB will operate the same when the database is converted to HALDB. There are three exceptions. First, if you make a partition unavailable, the program will receive an unavailable data condition. This condition can exist with non-HALDB databases, but for other reasons. Second, if you process a secondary index as a database, the I/O area may differ. This depends on the

definition of the secondary index. Processing of the indexed database does not change. Third, you cannot insert logical children with an application load program (PROCOPT=L). This would only affect you if you created a new logically related database. We explain how to handle these changes in Chapter 11, “Application considerations” on page 183.

No other changes to the external characteristics of the DL/I application programming interface were introduced for HALDB.

## 1.6 Logical relationships with HALDB

Logical relationships are supported with HALDB in the same way as they are with non-HALDB DL/I databases, with a single exception: Bidirectional logical relationships in a HALDB database must be implemented with physical pairing.

When loading a new partitioned database that contains logical relationships, the logical child segments may not be loaded as part of the load step. Logical children are added by normal update processing after the database has been loaded.

## 1.7 Partition selection

HALDB assigns database records to partitions based on the key of the root segment. When retrieving or inserting a database record, HALDB must select the correct partition. This process is called partition selection. Partition selection determines where root segments are placed and the order in which partitions are processed.

Partition selection is based either on the key ranges or a partition selection exit routine. For batch, BMP, and JBP programs, a PCB can be restricted to access a single partition.

### 1.7.1 Partition selection using key ranges

Partition selection can be done using key ranges for each partition. These ranges are created by assigning a high key to each partition. A root segment is assigned to the partition with the lowest high key, which is also no higher than the key of the root, for example, if there are three partitions with high keys of 1000, 2000, and 3000. Keys of 1001 through 1500 are assigned to the partition with a high key of 2000. Partitions are ordered in the order of their high keys.

## 1.7.2 Partition selection using a partition selection exit routine

Installations that prefer to select partitions by some criteria other than key ranges may use a partition selection exit routine. The routine assigns records to partitions based on the key of the root segment. It also determines the order in which sequential processing accesses the partitions. We explain the requirements and possibilities for these exit routines in 7.2, “Writing partition selection exit routines” on page 82.

## 1.8 Database structure

There are three HALDB database types:

- ▶ PHDAM: Partitioned HDAM
- ▶ PHIDAM: Partitioned HIDAM
- ▶ PSINDEX: Partitioned secondary index

The primary index of a PHIDAM database is not separately defined. Its definition is included with the definition of the database. Each partition of a PHIDAM database has its own index data set.

Each partition of a PHDAM or PHIDAM has an indirect list data set (ILDS). The ILDS is used for the self-healing pointer process. This process eliminates the need to rebuild secondary indexes or resolve logical relationships when databases are reorganized. We describe the process in 14.2, “The self-healing pointer process” on page 251. It is only used for logical relationships and secondary indexes.

HALDB uses direct pointers, such as physical child and physical twin pointers, within PHDAM and PHIDAM databases. These are the same pointers that are used with non-HALDB databases.

### 1.8.1 Partition ID and reorganization number

Each partition in a database is assigned a partition ID when it is defined. The partition ID is a 2-byte number that is incremented with the definition of each new partition in a database. Partition IDs are not reused.

Each partition in a PHDAM or PHIDAM has a reorganization number. This number is incremented when the partition is reorganized.

The partition ID and reorganization number are stored in the first data set of each partition of the PHDAM and PHIDAM databases. This is the data set in which root segments are stored. They are stored in the field of the first bit map block that would otherwise hold the free space element anchor point (FSEAP).

## 1.8.2 Segment structure

HALDB segments consist of two parts, prefix and data. This is the same as non-HALDB databases. However, HALDB prefixes have some additional fields.

### PHDAM and PHIDAM

The prefix for PHDAM and PHIDAM segments includes an indirect list key (ILK). The ILK is assigned to a segment when it is created. Its value never changes during the life of the segment. The ILK is 8 bytes and has the following data:

- ▶ The relative byte address (RBA) of the segment at its creation time (4 bytes)
- ▶ The partition ID at creation time (2 bytes)
- ▶ The partition reorganization number at creation time (2 bytes)

Figure 1-3 shows the format of a HDAM or HIDAM segment and that of a PHDAM or PHIDAM segment. The ILK is the last part of the prefix in the HALDB segments. These HALDB segments always include a physical parent pointer in the prefix of segments other than roots. Some non-HALDB segments also have physical parent pointers. They are required in non-HALDB segments only when the segment is the target of a logical relationship or secondary index or the parent of such a segment. They may also be included by specifying the pointer in the DBDGEN.

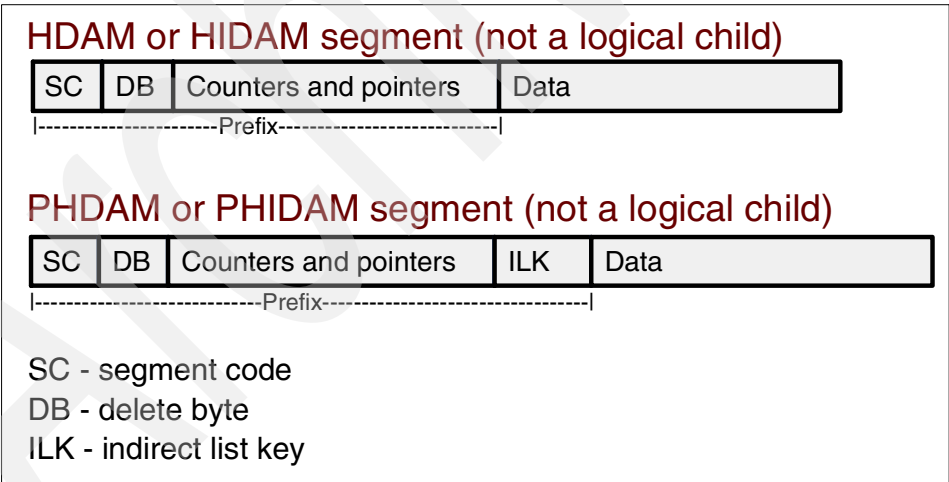


Figure 1-3 HDAM, HIDAM, PHDAM, and PHIDAM segments

Figure 1-4 on page 13 shows the format of HDAM, HIDAM, PHDAM, and PHIDAM logical child segments. HALDB logical child segments have an extended pointer set (EPS) in their prefix. This replaces the RBA pointer, which non-HALDB logical child segments have among their pointers. HALDB logical

child segments always have their logical parent's concatenated key stored in the data area. This is optional with non-HALDB logical child segments.

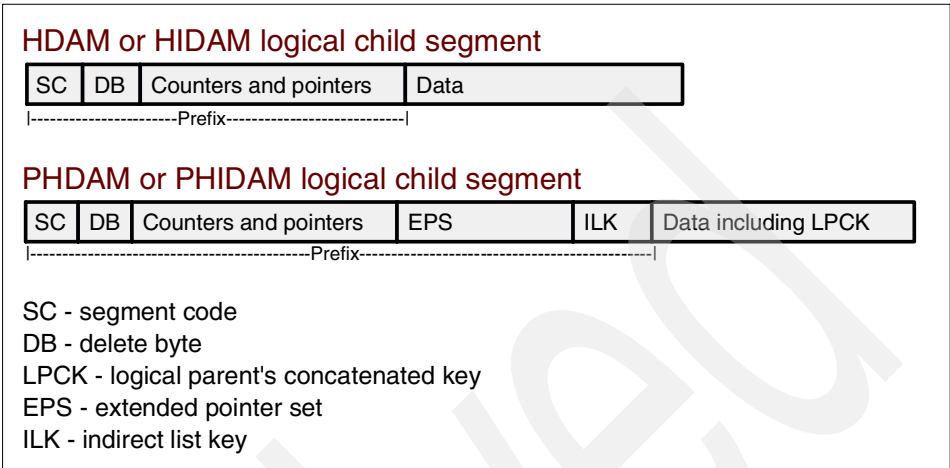


Figure 1-4 HDAM, HIDAM, PHDAM, and PHIDAM logical child segments

### Secondary indexes

The prefix of non-HALDB secondary index segments has a four-byte RBA when direct pointers are used. The prefix of HALDB secondary index segments includes an EPS and the key of the target's root segment. Figure 1-5 shows the format non-HALDB (INDEX) secondary index segments and HALDB (PSINDEX) secondary index segments.

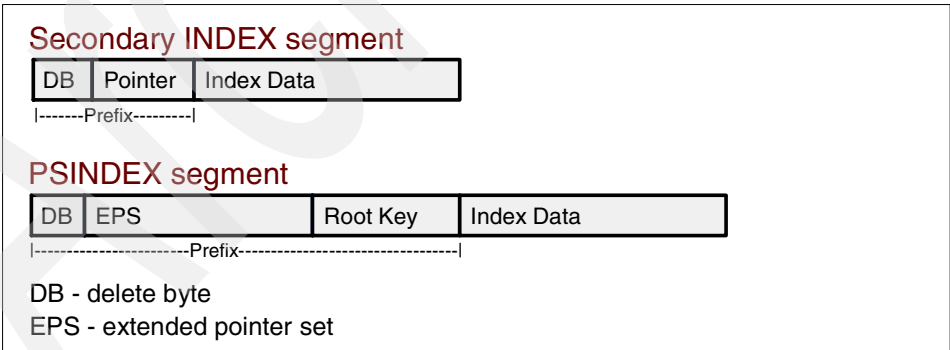


Figure 1-5 Secondary index segments

### 1.8.3 Extended pointer set (EPS)

Each logical child or secondary index segment contains an extended pointer set (EPS). The EPS contains the pointer for the logical relationship or secondary index entry. With bidirectional logical relationships, the EPS contains pointers to the logical parent and to the logical child pair. It replaces the direct pointers or symbolic pointers that are used with non-HALDB databases. They allow you to reorganize a database without updating logical relationships or rebuilding secondary indexes. We explain how EPSs are used in 14.2, “The self-healing pointer process” on page 251. The EPS is 28 bytes and contains the following fields:

- ▶ Reorganization number of target partition
- ▶ Partition ID of target partition
- ▶ RBA pointer to logical parent or secondary index target
- ▶ RBA pointer to paired logical child for bidirectional logical relationships
- ▶ Indirect list key (ILK) of target segment
- ▶ Database record lock ID of the target

### 1.8.4 Indirect list data set (ILDS)

The ILDS is a VSAM KSDS data set. It is the repository for indirect pointers. These pointers eliminate the need for updating logical relationship or secondary index pointers after a reorganization. An ILDS contains indirect list entries (ILEs), which are composed of keys and data. The data parts of ILEs contain direct pointers to the target segments.

Each PHDAM or PHIDAM partition has an ILDS. The ILDS is required even when there are no secondary indexes or logical relationships. Indirect list entries only exist for segments involved in inter-record pointing. These are target segments of secondary indexes or logical relationships. Secondary indexes do not have an ILDS because they never contain any target segments.

Indirect list entries are created or updated only by reorganization reload. They are not created or updated by non-reload processing. An initial load program (PROCOPT=L) does not create any ILEs. After an initial load, the ILDS is empty. If you insert a target segment during normal processing (PROCOPT=I or A), no ILE will be created either. ILEs are not needed in these cases because inserts of target segments create accurate pointers in the EPSs of the secondary index entries or logical child segments.

The key to the ILE is a 9-byte field consisting of the indirect list key and the segment code of the target segment. The ILK is also stored in the segment prefix of the target segment. The next field contains flags (1 byte). ILE data contains an old and a new copy (20 bytes each) of information needed for the self-healing pointer process.



These are:

- ▶ Partition ID
- ▶ Reorganization number
- ▶ RBA of logical parent or secondary index target segment
- ▶ Database record lock ID for target segment
- ▶ RBA of paired logical child when physical pairing is used

### 1.8.5 Number of data sets

The number of data sets per partition depends on the database organization.

- ▶ PSINDEX

A PSINDEX may contain one to 1001 partitions. Each partition contains only one data set, so a PSINDEX may have one to 1001 data sets.

- ▶ PHDAM

A PHDAM database may contain one to 1001 partitions. Each partition has an ILDS and one to 10 data sets for data set groups, so a partition has two to 11 data sets and a PHDAM database may have two to 11,011 data sets.

- ▶ PHIDAM

A PHIDAM database may contain one to 1001 partitions. Each partition has an ILDS, a primary index data set, and 1 to 10 data sets for data set groups, so a partition has three to 12 data sets and a PHIDAM database may have three to 12,012 data sets.

The data sets for data set groups are called data data sets. These data sets contain the segment data for PHDAM and PHIDAM databases.

Figure 1-6 on page 16 shows the data sets for the different types of HALDB databases.

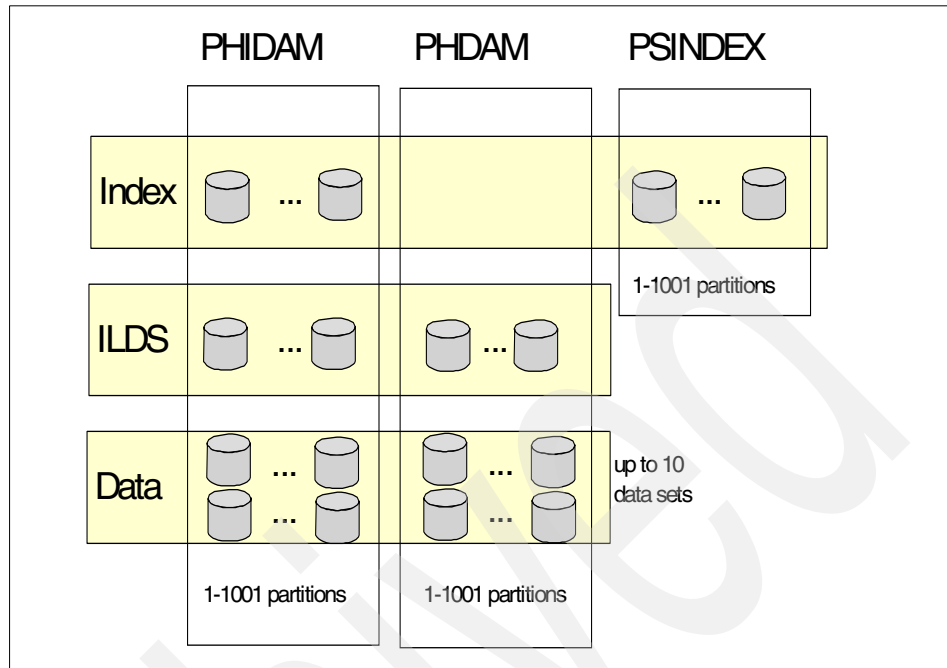


Figure 1-6 Data sets in HALDB

The maximum data set size is 4 GB. This applies to both OSAM and VSAM data sets.

### 1.8.6 Special considerations for secondary indexes

A secondary index database must be a PSINDEX if the target database is a HALDB. The secondary index itself may have one or more partitions.

The key ranges for the partitions of a secondary index are not likely to match the key ranges of its target database. Partition selection for the secondary index is based on its key. Partition selection for the target database is based on its key. Usually, these keys are unrelated. In Figure 1-7 on page 17 the target database is partitioned on last name and the secondary index is partitioned on employee number. You will notice that both partitions of the secondary index have target segments in each of the partitions of the target database.

This will be significant for timestamp recoveries of partitions. A timestamp recovery of a partition in the target database would require a timestamp recovery of all of the partitions in the secondary index to the same time. This would require a timestamp recovery of the other partitions in the target database to this same

time. That is, you will not be able to do a timestamp recovery of an individual partition for most databases that have secondary indexes.

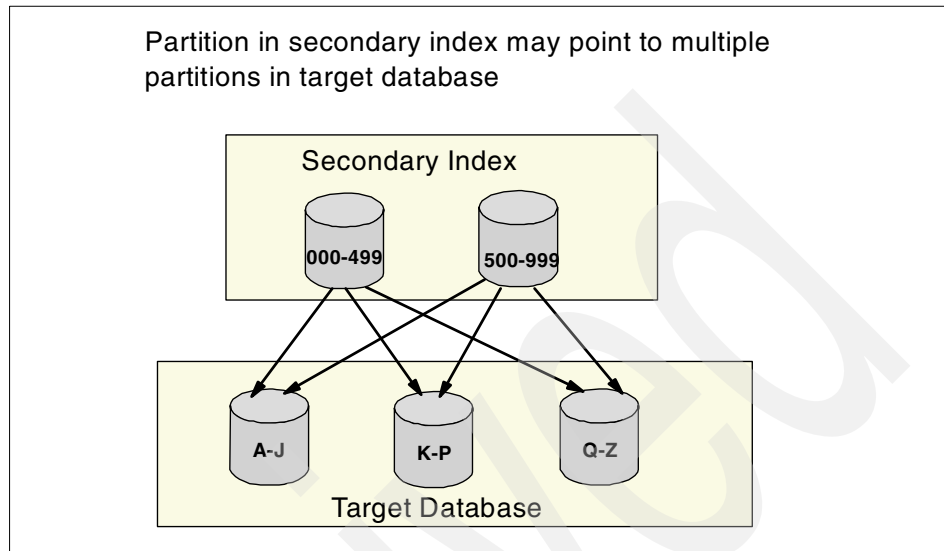


Figure 1-7 Partitioned secondary index

A partition selection exit could be used to partition the secondary index along the same boundaries of the main database. There has to be an appropriate key, because the selection of a partition always depends on the root key. The exit has the possibility to use just parts of the root key to define the correct partition.

### Miscellaneous requirements and considerations

There are some other requirements and considerations for secondary indexes:

- ▶ Symbolic pointing is not allowed.
- ▶ Secondary indexes must have unique keys. /SX and /CK fields may be used to provide uniqueness.
- ▶ Shared secondary indexes are not supported.
- ▶ The reorganization of a target database does not require the rebuilding of its secondary indexes.

## 1.9 Naming conventions

HALDB databases may have many partitions and even more data sets. Naming conventions for DD names and data set names are used to make their management simpler.

### 1.9.1 Partition names

Each partition is given a partition name that is one to seven bytes in length. These names must be unique among the database names, partition names, and Fast Path area names that are registered in a set of RECONS. A partition name may be used to describe the data in the partition, but the name should be chosen carefully. If you add or delete partitions or modify their boundaries, data may be moved from one partition to another. This may make the assignment of meaningful names difficult. There is no way to change the name of an existing partition.

### 1.9.2 DD names

DD names are constructed by appending a one-byte suffix to the partition name. The suffix indicates the function of the data set.

For PHIDAM or PHDAM databases, these are:

- ▶ Suffixes A through J are used for data data sets.  
This is the same value that is defined in the DSGROUP statement in the DBD. If there is no DSGROUP specified, the default is A.
- ▶ Suffix L is used for the indirect list data set (ILDS).
- ▶ Suffix X is used for the primary index data set.

For PSINDEX databases, there is only one data set in a partition: Suffix A is used for the secondary index data set.

Thus, a partition in a PHIDAM database with partition name PART1 would have DD names of PART1A for its first data data set, PART1B through PART1J for any successive data data sets, PART1L for its ILDS, and PART1X for its primary index data set.

A secondary index partition with partition name PARSI would have a DD name of PARSIA for its data set.

### 1.9.3 Data set names

Each partition is given a data set name prefix. It may be up to 37 characters. IMS appends seven characters to this prefix to form the data set name. The IMS-supplied portion contains a one-character alphabetic suffix to indicate the function of the data set and a five-digit partition ID number. The appendix of .A00001 indicates the first data data set in the partition with partition ID 1. The alphabetic character is the same one used as the suffix for the DD name of the data set.

There is no required correlation between the partition name and the names of its data sets. You may assign the same data set name prefix for every partition in a database. Since partition IDs are unique, data sets in different partitions will have different data set names.

Some examples of databases that we use later in this publication are the Customer database (CUSTDB) and the New Order database (NORDDDB). In our example the Customer database is PHDAM and has two partitions, and the New Order database is PHIDAM and has two partitions. Table 1-1 shows the corresponding DD and data set names. The example used may not be a practical one for a real environment, as the partition names and DD names must be unique within their RECON.

Table 1-1 Customer and order databases - DSN table

Master DBD name	Partition name	DD name	Partition ID	Data set name
CUSTDB	CUST1	CUST1A CUST1L	1	IMSPSA.IM0A.CUSTDB.A00001 IMSPSA.IM0A.CUSTDB.L00001
CUSTDB	CUST2	CUST2A CUST2L	2	IMSPSA.IM0A.CUSTDB.A00002 IMSPSA.IM0A.CUSTDB.L00002
NORDDDB	NORD1	NORD1A NORD1L NORD1X	1	IMSPSA.IM0A.NORDDDB.A00001 IMSPSA.IM0A.NORDDDB.L00001 IMSPSA.IM0A.NORDDDB.X00001
NORDDDB	NORD2	NORD2A NORD2L NORD2X	2-	IMSPSA.IM0A.NORDDDB.A00002 IMSPSA.IM0A.NORDDDB.L00002 IMSPSA.IM0A.NORDDDB.X00002
NORDDDB	NORD3	NORD3A NORD3L NORD3X	3	IMSPSA.IM0A.NORDDDB.A00003 IMSPSA.IM0A.NORDDDB.L00003 IMSPSA.IM0A.NORDDDB.X00003
NORDDDB	NORD4	NORD4A NORD4L NORD4X	4	IMSPSA.IM0A.NORDDDB.A00004 IMSPSA.IM0A.NORDDDB.L00004 IMSPSA.IM0A.NORDDDB.X00004



## Defining HALDB databases

This chapter gives an overview of the definition tasks for HALDB. The process you follow to implement a HALDB is similar to implementing a full-function non-HALDB database.

This chapter describes system definition, buffer pool assignment, definition of database exit routines, dynamic allocation, and DBDGEN differences for HALDB databases.

## 2.1 Overview of HALDB definition

This section briefly describes the steps you need to follow for defining a HALDB database. A HALDB structure consists of two distinct entities, the master database structure and its associated partition structures. The master structure is defined and generated with the DBDGEN process. It is basically unchanged from non-HALDB methods, so your first step is to design the logical structure of your database and implement it with the DBDGEN process. This defines the structure of the master database. A HALDB database contains one or more partitions. You must determine the number and characteristics of your partitions and implement them. The definition process for partitions is external to the DBDGEN process. It uses either the Partition Definition utility (PDU) or DBRC commands.

After ACBGEN and system definition for the master database, assignment of data sets to buffer pools also must be done.

Some of the differences between HALDB and non-HALDB definitions are covered in more detail Chapter 8, “Migration from non-HALDB to HALDB” on page 89.

### 2.1.1 Design the logical structure of the database

You need to determine your segment structures, the relationships between segments, segment layouts, and, if appropriate, logical relationships and secondary indexes. You follow the same process here as you would for a non-HALDB database.

### 2.1.2 Implement the logical structure with the DBDGEN process

The DBDGEN creates the HALDB master DBD.

The master DBD provides the structure for DL/I processing, as well as a model for creating the partitions.

The logical structure of the database includes the relationship between the segments, the pointers used, the database access method (PHDAM or PHIDAM), the segment layouts, the database physical data set organization (VSAM or OSAM), and, if appropriate, any secondary indexes or logical relationships. You should also decide whether you need data set groups. One reason for data set groups is now obsolete. You do not need them to resolve size problems, but they could still be helpful for resolving potential performance problems.



Other functions of DBDGEN for non-HALDB databases, such as defining the physical characteristics of the data sets and their DDNAMEs, are not performed for HALDB. You have to use other tasks to define them.

### 2.1.3 Determine the partitioning scheme

You must determine your partitioning scheme. The questions you need to answer here include:

- ▶ How many partitions do you need in your database?
- ▶ How much DASD storage space is required for each partition?
- ▶ How much free space do you need in your database? Do the free space parameters need to be different for each partition?
- ▶ What key is to be used to determine the split between partitions?
- ▶ Do you need a partition selection exit?
- ▶ For PHDAM, do you want all of your partitions to use the same randomizing parameters, or do you want to have randomizing parameters specific for each partition?
- ▶ Do you want to automatically or manually define your partitions? This decision influences the names of the partitions. You cannot change these names afterwards.

The HALDB Migration Aid utility (DFSMAID0) may help you in determining the answers to some of these questions.

You have to decide how many partitions you want. Usually this is determined by two considerations: The size of the database and the time required for the reorganization of the database. Partitions may be reorganized independently and in parallel. Since smaller partitions may be reorganized faster, defining many small partitions allows you to reorganize the database in a shorter time.

### 2.1.4 Create the partitioning scheme

HALDB partition definitions are stored in DBRC RECONs. When defining partitions, you must have update authority for the RECON data sets.

From DBRC's point of view, a HALDB is a master database consisting of partition databases.

You have two ways of defining partitions:

- ▶ Partition Definition utility (PDU)

This utility is invoked under ISPF. It reads the DBDLIB and registers the database. You can use it only after having performed a DBDGEN.

There are panels in which you can specify the physical characteristics, such as high keys, randomizer values, and block sizes, as well as some DBRC options, such as skeletal JCL members and maximum number of image copies. Parameters for which default values were defined in DBDGEN are included on some panels. The output of the definition is placed in DBRC's RECON data sets.

For more information about the PDU see Chapter 4, "Partition Definition utility" on page 45.

- ▶ DBRC batch commands

You may issue the INIT.DB and INIT.PART commands to register the master database and define its partitions.

For more information about DBRC batch commands see Chapter 5, "Batch definition of HALDB" on page 61.

When using key range partitioning we recommend that you specify a value of x'FF' for the last partition. This ensures that all keys may be assigned to a partition. If the last partition (the partition with the highest key specified) has a key value other than x'FF's, any attempt to access or insert a database record with a key higher than the high key specified results in an FM status code for the call. Application programs written for non-HALDB databases are unlikely to expect this status code.

## 2.1.5 Database exit routines

With the exception of PHDAM randomizing routines, database exit routines are defined for the whole database, not just for a partition. These exit routines are:

- ▶ Data capture exit routine
- ▶ Data conversion user exit routine
- ▶ secondary index database maintenance exit routine
- ▶ Segment edit/compression exit routine

These exit routines are not passed any information about partitions. Their actions cannot be tailored for individual partitions.

## 2.1.6 System definition

The DATABASE macro statement is used to define physical databases for an IMS online system. One DATABASE macro must be specified for each PHDAM, PHIDAM, and PSINDEX database. Unlike HIDAM primary indexes, PHIDAM primary indexes are not defined explicitly and DATABASE macros are not used in the system definition. Instead, they are implicitly defined by the DATABASE macro for the PHIDAM database.

## 2.1.7 Buffer pools

HALDB database data sets are assigned to buffer subpools with DFSVSMxx members for online systems and the DFSVSAMP DD data sets for batch jobs and utilities. This is the same method that is used for non-HALDB databases.

Buffer subpools may be used for both HALDB and non-HALDB data sets. These subpools are defined with IOBF statements for OSAM and VSRBF statements for VSAM. They are not changed for HALDB support.

DBD statements are used to assign data sets to specific subpools. Example 2-1 shows the DBD statement format. The dbdname value is used to specify a database or partition. For non-HALDB data sets, the data set identifier is a number. This is the data set number within the database specified by the dbdname value. For HALDB data sets, the data set identifier is a letter. This is the letter associated with the type of data set. It is the same letter that is used for the suffix in the data set's DDNAME. A through J are used for data set groups. X is used for PHIDAM primary indexes. L is used for ILDSs. A is used for secondary indexes. The id value identifies the database subpool to which the data set is assigned.

If a master database name is specified on a DBD statement, it applies to all partitions in the database. If a partition name is specified, it applies only to that partition. Statements specifying a partition name override any statements specifying the master database name of the partition.

*Example 2-1 DBD control statement for buffer pool assignment*

---

```
DBD=dbdname(data set identifier,id)
```

---

Buffer pools for ILDSs are always required when processing the PHDAM and PHIDAM databases. This is true even when the ILDSs have no entries in them.

## 2.1.8 Dynamic allocation

Dynamic allocation of HALDB data sets uses DBRC RECON information. Since all HALDB databases are registered in the RECONS, this information always exists. DFSMDA members are not used with HALDB data sets.

Dynamic allocation for HALDB data sets cannot be overridden. DD statements should never be used for HALDB data sets. If they are present and they do not match with the information in the RECON, the opening of the data sets fails.

The NODYNALLOC statement in DFSVSAMP specifications for batch jobs is ignored for HALDB data sets. For non-HALDB data sets it is used to disable dynamic allocation.

## 2.2 DBDGEN differences for HALDB

The database definition statements for HALDB databases are slightly different from non-HALDB databases. The four main things that are different in HALDB definitions are:

- ▶ ACCESS operands PHDAM, PHIDAM, and PSINDEX are used.
- ▶ DDNAME and data set information are not part of DBDGEN for HALDB. The database definition is purely for defining the hierarchical structure and relationships of the data. DATASET statements are not allowed for HALDB.
- ▶ The primary index of a PHIDAM database is automatically included in the generation of the PHIDAM database. DBDGEN cannot be performed for the primary index.
- ▶ Since DATASET statements are not allowed in HALDB database definitions, another method is used for defining multiple data set groups. They are defined with the DSGROUP parameter on SEGM statements. This parameter is not used with non-HALDB databases.

DBDGEN does not define individual partitions. This is done using the HALDB Partition Definition utility or DBRC batch commands. See Chapter 4, “Partition Definition utility” on page 45, and Chapter 5, “Batch definition of HALDB” on page 61.

Definitions specified in the DBDGEN source will generally apply to all partitions. The possible exception is the RMNAME parameter on the DBD statement. RMNAME must be specified in the DBD for PHDAM databases, but it may be overridden. Overrides may specify different values for different partitions.

The PSNAME parameter on the DBD statement specifies the name of a partition selection exit routine. It may be overridden by the PDU or DBRC commands. The

override may specify a different name or may specify that key range partitioning is to be used. Similarly, if the PSNAME parameter is not specified in the DBD, it may be specified by the PDU or DBRC commands.

**Note:** Partition selection is done prior to invoking the randomizing module. Randomizing modules only select locations within partitions.

Table 2-1 shows the parameters that are changed or added to DBDGEN statements.

Table 2-1 DBDGEN changes

Statement	Keyword	Description of change
DBD	NAME	No change.
	ACCESS	Add operands PHDAM, PHIDAM, and PSINDEX.
		INDEX operand is not used for primary index of PHIDAM.
		Other operands are unchanged.
	RMNAME	(mod,anch,rbn,bytes) - Default values for partitions. RBN is high RBN per partition.
	PSNAME (new)	Optional parameter. The partition selection exit module name is the operand. A user provided partition selection routine is not used with key range partitioning. PSNAME may be overridden by PDU or DBRC commands.
AREA/ DATASET		Not allowed for HALDB. Partitions are defined outside of DBDGEN.

Statement	Keyword	Description of change
SEGM	PARENT	VIRTUAL/PHYSICAL: Either may be specified, but only physical used. That is, either specification results in the logical parent's concatenated key being physically stored in the logical child segment.
	POINTER	HIER and HIERBWD are not allowed. TWIN, TWINBWD, or NOTWIN must be specified PHDAM and PHIDAM databases. For root segments in PHIDAM databases, only NOTWIN or TWINBWD are allowed.
		All dependent segments have a physical parent pointer.
		If the segment is a logical child, PAIRED and LPARNT are assumed. LTWIN and LTWINBWD are not allowed.
	SOURCE	Not allowed for PHDAM or PHIDAM. Only physical pairing is supported for bidirectional logical relationships.
	DSGROUP (new)	This keyword is used to specify multiple data set groups for partitioned databases. The format is DSGROUP= <i>c</i> , where <i>c</i> is the letters A through J. Up to 10 data set groups can be specified. The DSGROUP value corresponds to the values used in the DDNAME and data set naming conventions for HALDB.
		The default for every segment is A (first data set in each partition). If specified on the root segment, it must be DSGROUP=A.
		Gaps in the A–J sequence are not allowed. If DSGROUP=C is specified on a SEGM statement, there must also be at least one SEGM statement with DSGROUP=B and one with A.

Statement	Keyword	Description of change
LCHILD		LCHILD statements are not used for the primary index of a PHIDAM database. These indexes are automatically generated by PHIDAM definitions.
	POINTER	PTR=INDX is required for LCHILD statement following the target segment in indexed PHDAM or PHIDAM DBD. PTR=SNGL is required for the LCHILD statement in PSINDEX DBD. It is the default value for this LCHILD statement PTR=NONE is required for the LCHILD statement following a logical parent segment. It is the default for this LCHILD statement.
	PAIR	Unchanged from non-HALDB. HALDB does not support virtual pairing, so paired segment is physically paired.
	INDEX	Not needed for a primary index of PHIDAM, otherwise unchanged.
	RKSIZE (new)	For PSINDEX only, specifies the length of the root segment key of the target database.
	RULES	Not used: It does not apply to physical pairing.
FIELD		Unchanged. Note: /CK means the same as with non-HALDB. /SX creates an 8-byte ILK instead of 4-byte RBA.
XDFLD	CONST	For shared secondary indexes. Not allowed for HALDB.

We provide examples of the needed DBD changes later in this publication. Example 9-8 on page 138 shows the DBD source for a HIDAM database, and Example 9-9 on page 139 shows the DBD source after the database has converted to a PHIDAM.





## HALDB and DBRC

All HALDB databases are registered in the DBRC RECONS. Dynamic allocation always uses RECON information. In this chapter we explain the RECON records used for HALDB and the DBRC commands used to add, delete, modify, and list HALDB information. We provide sample listings of HALDB information from the RECONS.

## 3.1 RECON records for HALDB

HALDB databases are described in four types of RECON records. These records are:

- ▶ Master database record
- ▶ Partition database record
- ▶ Partition record
- ▶ Partition database data set record

Each HALDB database has one master database record. There is a partition database (DB) record and a partition (PART) record for each partition. There is a partition database data set (DBDS) record for each database data set. We describe these records and their uses in the following sections.

### 3.1.1 Master database record

Each HALDB database has a master database record. This is a new type of database record. There is a flag in the record to indicate that it is a HALDB master database record. This record is created either by the HALDB Partition Definition utility or an INIT.DB command that specifies TYPHALDB.

The record stores information that applies to the entire database. Among the data are:

- ▶ Partition selection exit routine name if one is used
- ▶ Number of partitions defined
- ▶ Change version number
- ▶ Highest allocated partition ID number
- ▶ Data set group count
- ▶ Global DMB number
- ▶ Share level
- ▶ RSR global service group and tracking level
- ▶ Recoverable or not recoverable

Authorization information is not stored in this record.

The record may be mapped with the DSPDBHRC macro in ADFSMAC and SDFSMAC.

The key of the record is composed of:

- ▶ HALDB database name (bytes 1–8)
- ▶ Zeros (bytes 9–16)
- ▶ x'18' (byte 17)
- ▶ Zeros (bytes 18–32)

Example 3-1 on page 39 shows a listing of a master database record.

### 3.1.2 Partition database record

Each HALDB partition has a partition database record. This is a new type of database record. There is a flag in the record to indicate that it is a HALDB partition database record. This record is created either by the HALDB Partition Definition utility when a partition is defined or by an INIT.PART command.

The record stores information that applies to the partition. Among the data are:

- ▶ Master database name
- ▶ Change version number for the partition
- ▶ *Partition init needed* flag
- ▶ Authorization flags including *prohibit further authorization* and *read only*
- ▶ Global DMB number, which is copied from the master database record
- ▶ Share level, which is copied from the master database record
- ▶ RSR global service group and tracking level, which are copied from the master database record
- ▶ Recoverable or not recoverable, which is copied from the master database record

Authorization information is stored in this record. The record may be mapped with the DSPDBHRC macro in ADFS MAC and SDFS MAC.

The key of the record is composed of:

- ▶ Partition name (bytes 1–8)
- ▶ Zeros (bytes 9–16)
- ▶ x'18' (byte 17)
- ▶ Zeros (bytes 18–32)

Example 3-2 on page 39 shows a listing for a partition.

### 3.1.3 Partition record

Each HALDB partition has a partition record. This is a new type of RECON record. This record is created either by the HALDB Partition Definition utility when a partition is defined, or by an INIT.PART command. It is created when the partition database record is created.

The record stores information that applies to the partition. Among the data are:

- ▶ Data set name prefix

- ▶ Partition ID
- ▶ Partition high key or partition string
- ▶ Length of partition high key or partition string
- ▶ Free space information
- ▶ Data set group count
- ▶ *Partition init needed* flag (PINIT flag)
- ▶ Previous and next partition names if using key range partitioning
- ▶ Randomization specifications for PHDAM
- ▶ Data set group block sizes for OSAM

The record may be mapped with the DSPPTNRC macro in ADFSMAC and SDFSMAC.

The key of the record is composed of:

- ▶ HALDB master database name (bytes 1–8)
- ▶ Partition name (bytes 9–16)
- ▶ x'19' (byte 17)
- ▶ Zeros (bytes 18–32)

### 3.1.4 Partition DBDS record

Each partition data set has a partition DBDS record. This is a new type of DBDS record. This record is created either by the HALDB Partition Definition utility when a partition is defined or by an INIT.PART command. It is created when the partition database data sets are defined. There are records for all partition data sets including ILDSs and PHIDAM primary index data sets.

The record stores information that applies to the partition data set. Among the data are:

- ▶ Data set name
- ▶ Skeletal JCL members
- ▶ Image copy information
- ▶ Change accumulation information
- ▶ Flags and counters for *image copy needed*, *recovery needed*, and so on

The record may be mapped with the DSPDSHRC macro in ADFSMAC and SDFSMAC.

The key of the record is composed of:

- ▶ Partition name (bytes 1–8)
- ▶ DDName (bytes 9–16)
- ▶ x'20' (byte 17)
- ▶ Zeros (bytes 18–32)

## 3.2 Dynamic allocation

The information in the RECONS is used for dynamic allocation of HALDB database data sets. The data set names and DD names from partition DBDS records are used to provide this information. DFSMDA members are never used with HALDB.

## 3.3 Authorization processing

Authorization for HALDB is done by partition, not database. The share level is set for an entire database, but individual partitions are authorized. Authorization occurs when a partition is first used. For example, a batch program that reads a database sequentially will authorize partitions one at a time. This occurs when each partition is first accessed. Authorization information is kept in the partition database records for each partition. The master database record contains the share level for the database but does not have information about any authorizations.

Since authorizations are by partition, not an entire database, different partitions may be authorized to different subsystems or utilities at the same time. This allows you to reorganize partitions in parallel as explained in Chapter 14, “HALDB reorganization” on page 243. It also allows you to execute non-data sharing batch programs against different partitions of the same database, as explained in 11.2, “Processing partitions in parallel” on page 189.

## 3.4 Partition initialization

Partitions must be initialized before they may be loaded. This is explained in Chapter 6, “Partition initialization” on page 73. The initialization required state for each partition is kept in the partition database record and the partition record for each partition. The state may be turned off or on with the CHANGE.DB command. This is explained in 6.2.1, “Partition initialization required (PINIT) flag” on page 75.

## 3.5 DBRC commands

You may use DBRC commands to define, delete, or modify information about HALDB databases and partitions. These commands are an alternative to the HALDB Partition Definition utility. You also may use DBRC commands to list RECON information about HALDB databases and partitions and to generate JCL for use with HALDB databases and partitions.

### 3.5.1 INIT commands

You may use INIT commands to define a HALDB database and its partitions.

#### INIT.DB

The INIT.DB command can be used to register the *master* database with DBRC. It defines the DBD name of the master database, its sharing level, the type of database, whether this database is partitioned by key ranges or a selection exit, and whether it is recoverable.

#### INIT.PART

The INIT.PART command can be used to register a partition after the master database is already registered with DBRC. It creates the HALDB partition structure, that is a partition record, a partition database record, and one or more partition DBDS records according to the DBD specification.

We explain the use of INIT.DB and INIT.PART commands in 5.2, “DBRC initialization commands for HALDB” on page 62. INIT.DBDS commands are not used for registering HALDB database data sets.

### 3.5.2 CHANGE commands

You may use the CHANGE.DB, CHANGE.PART, and CHANGE.DBDS commands to modify information about HALDB databases, partitions, and data sets.

#### CHANGE.DB

The CHANGE.DB command may specify either a HALDB master database or partition.

The master database name may be used to change the following:

- ▶ Share level
- ▶ Recoverable attribute
- ▶ Partition selection exit usage or routine name
- ▶ Image-copied required attribute for non-recoverable databases
- ▶ RSR specifications
- ▶ Partition initialization required status

When the master database name is used, the command applies to all partitions and data sets in the database.

A partition name may be used to change the following:

- ▶ The *prohibit further authorization* flag

- ▶ The *backout needed* flag
- ▶ The *read only* authorization flag
- ▶ The authorization of the partition to a subsystem
- ▶ The *partition init needed* (PINIT) flag

When the partition name is used, the command applies only to the partition.

## CHANGE.PART

You may use the CHANGE.PART command to change partition definitions. Both the master database name and the partition name are specified in the command. The master database name and partition name cannot be changed with this command. The command may be used to change the following attributes for a partition:

- ▶ The high key value when using key range partitioning
- ▶ The partition string when using a partition selection exit routine
- ▶ The data set name prefix
- ▶ The PHDAM randomization parameters
- ▶ The free space specifications
- ▶ The OSAM block sizes
- ▶ The GENMAX value
- ▶ The RECOVPD value
- ▶ The image copy REUSE attribute
- ▶ The skeletal JCL member names

## CHANGE.DBDS

You may use the CHANGE.DBDS command to change information about data sets in a HALDB partition. Specify the partition name, not the master database name, in the DBD parameter. The command may not be used to change data set names or DD names. These are determined by the HALDB naming conventions. The command may be used to change the following attributes in data sets other than ILDSs and PHIDAM primary indexes:

- ▶ The GENMAX value
- ▶ The RECOVPD value
- ▶ The image copy REUSE attribute
- ▶ The skeletal JCL member names
- ▶ The *image copy needed* flag
- ▶ The *recovery needed* flag
- ▶ Error queue elements (EQEs)

For ILDSs and PHIDAM primary indexes the only attributes that may be changed are:

- ▶ The *recovery needed* flag
- ▶ Error queue elements (EQEs)

We explain the use of CHANGE.DB, CHANGE.PART, and CHANGE.DBDS commands in 5.3, “DBRC change commands for HALDB” on page 68.

### 3.5.3 DELETE commands

You may use DELETE.DB and DELETE.PART commands to delete HALDB databases and partitions from the RECONS. DELETE.DBDS commands are not allowed for HALDB database data sets.

#### DELETE.DB

You may use the DELETE.DB command to delete a HALDB master database, all of its partitions, and associated allocation, image copy, recovery, and reorganization records from the RECONS. You cannot specify a partition name on the DELETE.DB command. Use the DELETE.PART command to delete a partition from the RECONS.

#### DELETE.PART

You may use the DELETE.PART command to delete a partition and associated allocation, image copy, recovery, reorganization, and DBDS records from the RECONS.

We explain the use of DELETE.DB and DELETE.PART commands in 5.4, “DBRC delete commands for HALDB” on page 71. DELETE.DBDS commands are not used to delete HALDB database data sets.

### 3.5.4 LIST commands

You may use LIST.DB, LIST.DBDS, and LIST.HISTORY commands to list information about HALDB databases, partitions, and data sets from the RECONS.

#### LIST.DB

You may use the LIST.DB command to list HALDB database, partition, and data set information. You may specify the TYPHALDB parameter to list information about all HALDB databases. Alternatively, you may specify either a HALDB master database name or a partition name in the DBD parameter of the command. If you use the master database name, information from all partitions is listed.

#### LIST.DBDS

You may use the LIST.DBDS command to list information about HALDB database data sets. You may specify either a master database name or a



## Sample listings

*Example 3-1 LIST.DB output for master database*

Example 3-2 shows a listing for a partition. The partition high key record is listed twice. The second listing is in hexadecimal. Even though the key is only 10 bytes, it is listed as if it were 32 bytes. The last 22 bytes are a conversion of blanks to hexadecimal. The hexadecimal listings of partition high keys or partition strings are always done so that remaining bytes on a line are listed as X'40'.

### Example 3-2 LIST.DB output for a partition

Chapter 3. HALDB and DBRC 39

READ ONLY	=OFF	IMAGE COPY NEEDED COUNT	=0
PROHIBIT AUTHORIZATION	=OFF	AUTHORIZED SUBSYSTEMS	=0
		HELD AUTHORIZATION STATE	=0
		EEQE COUNT	=0
TRACKING SUSPENDED	=NO	RECEIVE REQUIRED COUNT	=0
OFR REQUIRED	=NO		
PARTITION INIT NEEDED	=NO		
ONLINE REORG ACTIVE	=NO		
PARTITION DISABLED	=NO		

---

Example 3-3 shows a listing of a HALDB database data set. ALLOC and REORG information is not shown but would appear for most data sets. Even though the database is PHDAM, the DBORG value is listed as HDAM. The DBORG value for PHIDAM database data sets is listed as HIDAM. The DBORG value for ILDS, PHIDAM primary index, and secondary index data sets is always INDEX.

*Example 3-3 LIST.DB or LIST.DBDS output for a HALDB database data set*

---

```

DBDS
DSN=IMSPSA.IMOA.CUSTDB.A00001                                TYPE=PART
DBD=CUSTDB1 DDN=CUSTDB1A DSID=001 DBORG=HDAM DSORG=OSAM
CAGRP=**NULL** GENMAX=2 IC AVAIL=0 IC USED=1 DSSN=00000000
NOREUSE RECOVPD=0 OTHER DDN=**NULL**
DEFLTJCL=**NULL** ICJCL=ICJCL OICJCL=OICJCL RECOVJCL=RECOVJCL
RECVJCL=RECVJCL
FLAGS:                                COUNTERS:
  IC NEEDED      =OFF
  RECOV NEEDED   =OFF
  RECEIVE NEEDED =OFF                EEQE COUNT          =0
-----
IMAGE
RUN      = 2003.073 16:26:23.3 -05:00      * RECORD COUNT =15000
STOP     = 0000.000 00:00:00.0 +00:00      BATCH      USID=0000000001

IC1
DSN=IMSPSA.IC1.CUSTDB1.CUSTDB1A.D03073.T143654  FILE SEQ=0001
UNIT=3390                                         VOLS DEF=0001 VOLS USED=0001
                                                  VOLSER=TOTIMM

```

---

### 3.5.5 GENJCL commands

You may use GENJCL commands to generate JCL for HALDB databases, partitions, and data sets. The commands that are used with non-HALDB databases are used with HALDB. You may specify either a HALDB master database or a partition name in the DBD parameter. If you specify a DDN parameter, you must specify a partition name in the DBD parameter.

All GENJCL commands, other than GENJCL.USER, process data for DDNAMEs ending A through J. They do not process data for ILDSs (L) and PHIDAM primary indexes (X). ILDSs and PHIDAM primary indexes do not use standard recovery techniques. There are no image copies for them. They have no database change after image log records. They are not included in Change Accumulation processing. Thus, they are not applicable to GENJCL.IC, GENJCL.OIC, GENJCL.RECOV, GENJCL.CA, and GENJCL.RECEIVE processing.

## GENJCL.USER

You may use GENJCL.USER to process any HALDB database, partition, or data set. This includes ILDSs and PHIDAM primary indexes.

### ***Generating Index/ILDS Rebuild utility JCL***

IMS supplies a sample skeletal JCL member that you may use to generate JCL for the Index/ILDS Rebuild utility (DFSPREC0). The sample member name is DSPUPJCL in ADFSISRC and SDFSISRC. DBRC installation places skeletal JCL members into an IMS procedure library (PROCLIB). When using DSPUPJCL you specify the master database name, the partition name, and the type of recovery as user keys. The format of the command is shown in Figure 3-1.

```
GENJCL.USER MEMBER (DSPUPJCL)
  USERKEYS ((%MDBNAME, 'masterdbname'),
             (%DBNAME, 'partitionname'),
             (%RCVTYP, 'ILE | INDEX | BOTH'))
```

*Figure 3-1 GENJCL.USER command format for DFSPREC0*

Example 3-4 shows an example of a GENJCL.USER command. It will generate the INDEX/ILDS Rebuild utility JCL for the recovery of the PHIDAM index of partition NORDDDB2 in database NORDDDB.

*Example 3-4 GENJCL.USER command to rebuild the index of partition NORDDDB2*

---

```
GENJCL.USER MEMBER (DSPUPJCL) -
  USERKEYS ((%MDBNAME, 'NORDDDB'), (%DBNAME, 'NORDDDB2'), (%RCVTYP, 'INDEX'))
```

---

## Symbolic keywords

IMS supplies symbolic keywords and symbolic keyword values for use with HALDB. The %mdbname symbolic keyword is used for the master database name. The %dbname symbolic keyword is used for the partition name. The %dbtype symbolic keyword has the following values for HALDB database data sets:

<b>PDATA</b>	Data set types A through J (data DBDS and PSINDEX)
<b>PILDS</b>	Data set type L (ILDS)
<b>PINDEX</b>	Data set type X (PHIDAM primary index)

## 3.6 DBRC groups for HALDB

HALDB databases, partitions, and data sets may be members of DBRC groups.

### 3.6.1 Change accumulation groups

When defining a change accumulation group, you must specify the partition name and DDNAME for HALDB database data sets. ILDSs and PHIDAM primary indexes cannot be included in a change accumulation group.

### 3.6.2 Database data set groups

When defining a DBDS group, you must specify the partition name and DDNAME for HALDB database data sets. ILDSs and PHIDAM primary indexes may be included in a DBDS group.

### 3.6.3 Database groups

When defining a database group, you may specify master database names, partition names, or both.

### 3.6.4 Recovery groups

When defining a recovery group, you must specify a master database name. You cannot specify a partition name. When a database belongs to a recovery group, all of its partitions belong to the group.

## 3.7 Use of database names in DBRC commands

Many DBRC commands include the specification of a database name. Some of these commands require the use of a master database name. Some require the use of a partition name. Some allow either to be used.

### 3.7.1 Commands that require a master database name

The following commands require that the specified database name be a master database:

- ▶ INIT.DB
- ▶ DELETE.DB
- ▶ CHANGE.DB with the SHARELVL, NONRECOV, RECOVABL, PARTSEL, HIKEY, ICREQ, NOICREQ, GSGNAME, NOTCOVER, RCVTRACK, or DBTRACK parameters
- ▶ INIT.DBDSGRP with the RECOVGRP parameter
- ▶ CHANGE.DBDSGRP with the ADDRECOV or DELRECOV parameter

### 3.7.2 Commands that require a partition name

The following commands require that the specified database name be a partition name:

- ▶ CHANGE.DBDS
- ▶ CHANGE.DB with the AUTH, NOAUTH, BACKOUT, NOBACK, READON, READOFF, or UNAUTH parameters
- ▶ LIST.DBDS with the DDN parameter
- ▶ LIST.HISTORY with the DDN parameter
- ▶ INIT.CAGRP
- ▶ CHANGE.CAGRP
- ▶ INIT.DBDSGRP with the MEMBERS parameter
- ▶ CHANGE.DBDSGRP with the ADDMEM or DELMEM parameter
- ▶ INIT.IC
- ▶ CHANGE.IC
- ▶ CHANGE.UIC
- ▶ NOTIFY.IC
- ▶ NOTIFY.UIC

- DELETE.IC
- DELETE.UIC
- NOTIFY.REORG
- DELETE.REORG
- NOTIFY.RECOV
- DELETE.RECOV
- NOTIFY.BKOUT
- CHANGE.BKOUT
- NOTIFY.ALLOC
- DELETE.ALLOC

### **3.7.3 Commands that allow a master database or a partition name**

The following commands allow the specified database name to be either a master database name or a partition name:

- CHANGE.DB with PINIT or NOPINIT parameters
- LIST.DB
- LIST.DBDS without the DDN parameter
- LIST.HISTORY without the DDN parameter
- GENJCL.xxx
- INIT.DBDSGRP with the DBGRP parameter
- CHANGE.DBDSGRP with the ADDDB or DELDB parameter
- LIST.DBDSGRP ALL

### **3.7.4 DBRC commands that are not allowed with HALDB**

The following commands are not allowed with HALDB:

- INIT.DBDS
- DELETE.DBDS

## Partition Definition utility

This chapter describes how you can define, change, and delete your HALDB databases and partitions after you have performed DBDGEN by using the Partition Definition utility.

This chapter provides an overview of the utility. It includes security considerations. It also covers the configuration of the utility, the selection of databases, and the definition of their partitions. You may choose between two methods of defining partitions, automatic or manual. We also explain how to change and delete these definitions.

You can use both the HALDB Partition Definition utility and commands to manage your HALDBs. Chapter 5, “Batch definition of HALDB” on page 61, provides information about the alternative DBRC commands that may be used for definitions.

Refer to manuals *IMS Version 8: Administration Guide: Database Manager*, SC27-1283, and *IMS Version 8: Utilities Reference: Database and Transaction Manager*, SC27-1308, for more information on using the PDU.

## 4.1 Using the PDU

The HALDB Partition Definition utility (PDU) is an ISPF application that allows the database administrator to register the HALDB master database and to manage its partitions.

The utility is accessed by logging on to TSO and starting ISPF after the dialog data sets have been made available to the TSO user. You can add the data sets to a LOGON procedure or use TSO commands to allocate them. You can use the TSOLIB command to add data sets to the STEPLIB. Table 4-1 shows which file names and data sets need to be allocated. Be sure to use your own high level qualifiers.

*Table 4-1 File names and data sets to allocate*

File DD name	Sample data set name	Disposition
STEPLIB	IVPEXE81.SDFSRESL	N/A
SYSPROC	IVPEXE81.SDFSEXEC	SHR
ISPMLIB	IVPEXE81.SDFSMLIB	SHR
ISPPLIB	IVPEXE81.SDFSPLIB	SHR
ISPTLIB	IVPEXE81.SDFSTLIB	SHR
IMS	IVPEXE81.DBDLIB	SHR

If you use a logon procedure, you must log on again and specify logon with the new procedure. If you use allocation commands, they must be issued outside of ISPF. After you allocate the data sets and restart ISPF, restart the Install/IVP dialog, return to this task description, and continue with the remaining steps.

Start the HALDB Partition Definition utility from the ISPF command line by issuing the following command:

```
TSO %DFSHALDB
```

The utility consists of several panels and programs that perform various actions on the database and its partitions. The PDU updates the RECON data sets. You must be careful to properly back up your installation's RECONS so you do not lose the information about your HALDB databases. As all changes are made directly to the RECON and not held anywhere else it is important to understand that the changes take effect immediately. There is no record kept of any changes made. It is important that you protect critical information.



You specify the database and the type of action to perform on the first panel. Actions include define, modify, and view. The succeeding panels guide you through the processes.

**Note:** The PDU has minimal impact on online IMS subsystems with regard to RECON contention. The RECON is only reserved for the time it takes to process a DBRC request. It is not held for the duration of the utility execution.

You must have control authority on the RECON data sets to be able to use the PDU. IMS Version 8 introduces additional security support for DBRC commands when entered using the DBRC utility or using PDU. Commands may be secured at the command verb, resource type, and resource name level. Table 4-2 shows the equivalent DBRC commands that need to be protected. For example, the HALDB QUERY command is protected by the LIST.DB master database command.

Table 4-2 DBRC command authorization with IMS Version 8

HALDB request	Master or partition	Equivalent DBRC command
Query	Master	LIST.DB DBD(master db)
Set	Master	INIT.DB DBD(master db)
Set	Partition	INIT.PART DBD(master db)
Change	Master	CHANGE.DB DBD(master db)
Change	Partition	CHANGE.PART DBD(master db)
Delete	Master	DELETE.DB DBD(master db)
Delete	Partition	DELETE.PART DBD(master db)

### 4.1.1 Configuring the PDU

The first time you use the PDU, you must configure it for you. Configuration means the setting of DBDLIBs and RECONs you want to use. You may have several configurations in the PDU with different data sets. Figure 4-1 on page 48 shows the initial PDU panel.

Help

Partitioned Databases

Enter required field

Command ==>

Type a database name and choose an option. Then press Enter.

To select a database from a list, type a filter (\*) and press F4.

Configuration . . . : DEFAULT

Database name . . . \_\_\_\_\_ +

Option . . . . . \_

1. Open database partitions

2. Open database information

3. Delete database information

4. Export database information

5. Import database information

6. Show IMS concatenation

7. Select an IMS configuration

To exit the application, press F3.

Figure 4-1 PDU primary panel

Type a 7 in the Option field to indicate that you want to do configuration, and press Enter. Figure 4-2 shows the Configurations panel.

Configurations

Row 1 to 5 of 5

COMMAND ==>

Select a default by pressing a '/' on the desired line then press Enter.

You can use 'O' to open or 'D' to delete a configuration.

Act	Def	Name	Description
___			
___	*	<u>DEFAULT</u>	<u>Defaulting to pre-allocated datasets</u>

Figure 4-2 PDU configuration selection panel

You must type a slash (/) character in the Act field of the blank line to reach the Configuration Details panel, as shown in Figure 4-3 on page 49. On this panel, enter the configuration name with a brief description. We recommend that you specify a meaningful name, so you can easily associate the RECON data sets used in this configuration. Enter the appropriate RECON and DBDLIB data sets (up to 10) for this configuration and press Enter.

Configuration Details

COMMAND ==>

Type in values in the fields and press Enter to continue.

Configuration name . . . .

IMS0A

Description . . . . .

IMS System V8 - IM0A

RECON dataset names

RECON1 dataset . . . .

'IMSPSA.IM0A.RECON1'

RECON2 dataset . . . .

'IMSPSA.IM0A.RECON2'

RECON3 dataset . . . .

'IMSPSA.IM0A.RECON3'

DBDlib dataset names

DBDlib dataset 1 . . . .

'IMSPSA.IM0A.DBDLIB'

DBDlib dataset 2 . . . .

DBDlib dataset 3 . . . .

DBDlib dataset 4 . . . .

DBDlib dataset 5 . . . .

DBDlib dataset 6 . . . .

DBDlib dataset 7 . . . .

DBDlib dataset 8 . . . .

DBDlib dataset 9 . . . .

DBDlib dataset 10 . . . .

Figure 4-3 PDU Configuration Details panel

Upon returning to the PDU main panel, your new configuration becomes your default configuration. When you are maintaining multiple configurations in the PDU, you can change your active configuration at any time by entering the Configurations selection panel and typing a slash (/) in the Act field of your desired configuration.

If you choose Option 6 you may see the DBDLIB allocated, and if you enter ONLY RECON in the command line, the output shows you the allocated RECON data sets. Typing ONLY shows you all allocated data sets.

### 4.1.2 Selecting a database

Prior to defining your partitions, you must run the DBDGEN utility to define the master database. The PDU does not allow you to proceed until the DBDGEN has been run.

After you select an active configuration, you need to select a database. In the main panel type a 1 in the Option field.

Help

Partitioned Databases

Command ==>

Type a database name and choose an option. Then press Enter.  
To select a database from a list, type a filter (\*) and press F4.

Configuration . . : IMS0A

Database name . . . NORDDB +

Option . . . . . 1

1. Open database partitions

2. Open database information

3. Delete database information

4. Export database information

5. Import database information

6. Show IMS concatenation

7. Select an IMS configuration

To exit the application, press F3.

Figure 4-4 PDU database selection

You may either select a database directly from the main panel (as shown in Figure 4-4), or enter a filter in the Database name field. Filter characters are standard ISPF filter characters: Percent (%) for matching single characters and asterisk (\*) for matching multiple characters. A single asterisk presents the entire list of DBDLIB members (Figure 4-5).

If you use a filter in the Database name field, you will be presented with a standard ISPF library listing. You may select a HALDB database from the list by typing in a slash (/) in the selection field.

File Help							
DBD LIST		IMSPSA.IMS0A.DBDLIB				Row 00001 of 00003	
Command ==>						Scroll ==> PAGE	
Name	Prompt	Alias-of	Size	TTR	AC	AM	RM
. CUSTDB			000003C0	000103	00	24	24
. CUSTSI			00000138	00010B	00	24	24
. NORDDB			00000158	000113	00	24	24

Figure 4-5 Database selection list

You cannot select a non-HALDB database from this list. Attempting to do so takes you back to the main panel.

### 4.1.3 Setting HALDB master DBD parameters

After making a selection, you can start defining the information that pertains to the entire HALDB master database. This is performed in the Partitioned Database Information panel, as shown in Figure 4-6 on page 52.

We can see that this database has been defined as a PHIDAM database with one data set group and no partition selection exit. The defaults of no RSR parameters, share level 0, and recoverable are also shown. You may change most parameters. You cannot change the database organization or the number of data set groups. If you change a value that is also in the DBD, such as the partition selection exit routine name, it is not updated in the DBDLIB. If you do so, your DBD could have definitions for your database that differ from those actually being used. The values in PDU override those in DBD.

The message about database information for NORDDDB means that it is not yet registered in the RECONS. The information shown is taken from the DBD (and also contains some default values).

After you have pressed Enter, the PDU registers the HALDB master database in the RECONS, but no partition information is available yet.

Help

Partitioned Database Information

Command ==>

Type the field values. Then press Enter to continue.

Database Name . . . . . : NORDDB

Master Database values

Part. Selection Routine . . .  
RSR Global Service Group . . .  
RSR Tracking Type . . . . .  
Share level . . . . . 0  
Database Organization . . . : PHIDAM  
Recoverable? . . . . . YES  
Number of Data Set Groups . : 1

To exit the application, press F3.

Database information for 'NORDDB' was not found.

Figure 4-6 Defining HALDB master DBD information

After this panel is processed, the RECON listing shows the master database without partitions (see Example 5-4 on page 64).

4.1.4 Setting processing options and global partition information

Next, you need to decide whether you want the PDU to automatically name and create your partitions, or whether you wish to perform this task manually. You may also set some options that apply to all of your partitions. Refer to the Partition Default Information panel (Figure 4-7 on page 53) where these options are set.

Help

Partition Default Information

Command ==>

Type the field values. Then press Enter to continue.

Database Name . . . . . : NORDDDB

More: +

Processing options

Automatic definition . . . . . YES

Input dataset . . . . .

Use defaults for DS groups . . YES

Defaults for partitions

Partition Name . . . . .

Data Set Name Prefix . . . . .

Free Space

Free block freq. factor . . 0

Free space percentage . . . 0

Defaults for data set groups

Block Size . . . . . 4096

Figure 4-7 Setting partition defaults

On the right side of the panel you see More: +, which means that there are further default settings. Press F8 to scroll and you get the panel shown in Figure 4-8 on page 54. This shows you the defaults. Regardless of your choice for the type of definition, you can set the free space parameters for this database. The values will be used in all of the partitions when they are created. You may also change the DBRC options to be used. For a PHDAM database, you are provided with the opportunity to set global randomizing parameters for your partitions. The defaults for the randomizer are taken from the DBD.

Help	
Command ==>	Partition Default Information      End of data
Type the field values. Then press Enter to continue.	
Database Name . . . . .	NORDDB
Partition Name . . . . .	_____
Data Set Name Prefix . . . . .	_____
Free Space	
Free block freq. factor . .	0
Free space percentage . . .	0
Defaults for data set groups	
Block Size . . . . .	4096
DBRC options	
Max. Image Copies . . . . .	2
Recovery Period . . . . .	0
Recovery Utility JCL . . .	RECOVJCL
Default JCL . . . . .	_____
Image Copy JCL . . . . .	ICJCL
Online Image Copy JCL . . .	OICJCL
Receive JCL . . . . .	RECVJCL
Reusable? . . . . .	_____

Figure 4-8 Scrolling partition defaults

### 4.1.5 Creating your partitions manually

To create your partitions manually, enter N0 in the Automatic definition field. Then provide the partition name and data set name prefix. These are shown in Figure 4-9 on page 55. The DDNAMEs and data set names will be created according to the naming conventions.

If you want to use the defaults defined on the previous panel, that is all you have to do. Otherwise, specify individual free-space information, randomizing values for PHDAM partitions, and your selected DBRC options.



Help

Partition Default Information

Command ===>

Type the field values. Then press Enter to continue.

Database Name . . . . . : NORDDDB

More: +

Processing options

Automatic definition . . . . . NO

Input dataset . . . . . \_\_\_\_\_

Use defaults for DS groups . . YES

Defaults for partitions

Partition Name . . . . . NORDD1

Data Set Name Prefix . . . . . IMSPSA.IM0A.NORDDB

Free Space

Free block freq. factor . . . 0

Free space percentage . . . . 0

Figure 4-9 Manual definition of partitions

When you press Enter, you will be presented with the Change Partition panel, as shown in Figure 4-10 on page 56. On this panel you can name your individual partitions as well as define the high key values for them. As you can see, some information is propagated forward from previous panels. The partition name and the data set name prefix were copied forward. You can rename them at this time.

In Figure 4-10 on page 56 the key is numeric. You could also have a key such as cz. The partition high key field is *case sensitive*, so the value cz is different from the value CZ. The values entered in this field are not translated automatically from lower case to upper case.

High key and partition string values may be specified as either character or hexadecimal. The default is character. To specify a hexadecimal value you must specify it explicitly. For example, you could specify a value of x'0084C68E0000' for a six-byte key.

The partition ID is generated by the PDU and is used as part of the database data set name. You cannot change it. When you press Enter you receive a message informing you that the partition was added successfully. Then the Change Partition panel is shown with the partition ID. To create another partition, you may just change the value in the Partition name field and enter a new high key. Then the next partition is defined. To stop defining further partitions press F12.

On your last partition, specify a value in the partition high key field that is a higher value than any expected key value for your database. This prevents the return of an FM status code to a program that attempts to insert or retrieve a root segment with a key value higher than all of your defined partition high keys.

Help

Change Partition

Command ===>

Type the field values. Then press Enter to continue.

Database Name . . . . . : NORDDB

Partition Name . . . . . : NORDD1

Partition ID . . . . . : 1

Data Set Name Prefix . . . . . : IMSPSA.IM0A.NORDDB

More: +

Partition high key

00060300002443

Free Space

Free block freq. factor . . 0

Free space percentage . . . 0

Figure 4-10 Change Partition panel

As you complete your partition definitions, the Database Partitions panel, as shown in Figure 4-11 on page 57, lists what you have defined. You can update your partition definitions by entering a slash (/) in the appropriate Act field and pressing Enter.

At this time, the RECONs have also been updated with the partition information. Example 5-8 on page 67 shows the first part of the RECON listing.

```

Database Partitions                               Row 1 to 4 of 4
COMMAND ==> _____
Select an item by pressing a '/' on the desired line then press Enter.

Database Name . . . . . : NORDDB

Act   Name      Id      Data Set Name Prefix
____  _____  _____  _____
___   NORDD1    1      IMSPSA.IM0A.NORDDB
___   NORDD2    2      IMSPSA.IM0A.NORDDB
___   NORDD3    3      IMSPSA.IM0A.NORDDB
___   NORDD4    4      IMSPSA.IM0A.NORDDB
***** Bottom of data *****

```

Figure 4-11 Database partitions listing

## 4.1.6 Creating your partitions automatically

If the number of partitions you need to define is small, creating them manually may be sufficient. You define the partitions serially, perhaps setting different attributes for each individual partition. Partition names may be easily controlled and defined. However, if you have a larger number of partitions, this process is rather tedious and prone to errors. You have the option of automatically creating your partitions. Then all partitions have the same attributes in the beginning, but you could change them afterwards. You cannot change partition names or partition IDs.

Before you can start an automatic definition you need to create an input file containing the partition high key values for each partition. This input file must have the same number of records as partitions you wish to define, and the partition high key values must start in column 1 of each record. Figure 4-12 is an ISPF browse of an input key's data set, which is used to define four partitions corresponding to the four records containing the high key. The last line contains the value x'FF'. It creates a high key of all x'FF' bytes.

```

BROWSE      IMSPSA.IM0A.PHKEYS                      Line 00000000 Col 001 000
Command ==> _____ Scroll ==> PAGE
***** Top of Data *****
00060300002443
00110500002902
00160800002208
x'FF'
***** Bottom of Data *****

```

Figure 4-12 Sample input keys data set

The data set must be allocated with RECFM=VB and a maximum LRECL of 540. You may define it as a PDS and then specify the appropriate member.

To automatically define the same database defined in 4.1.5, “Creating your partitions manually” on page 54, your entries on the Partition Default Information panel are slightly different. Figure 4-13 shows the Partition Default Information panel for automatic definitions.

You now enter YES in the Automatic definition field. The data set containing the partition high key values is entered in the Input dataset field. Then you specify the pattern for the partition name. The resulting partition names differ from the previously entered partition names. In our example, we enter NORDD%. The value may be up to seven characters, including % signs. The % signs are replaced with alphanumeric values in collating sequence. For the first partition, each % sign is replaced with an A. Further partitions get B to Z followed by 0 to 9. If you have more than 36 partitions, you must use more than one % sign. In our example, the names of the partitions will be NORDDAA, NORDDAB, NORDDAC, and NORDDAD. As before, you must enter the data set name prefix you want to use.

Help

Partition Default Information

Command ==>

Type the field values. Then press Enter to continue.

Database Name . . . . . : NORDDB

More: +

Processing options

Automatic definition . . . . . YES

Input dataset . . . . . 'IMSPSA.IM0A.PHKEYS'

Use defaults for DS groups . . YES

Defaults for partitions

Partition Name . . . . . NORDD%

Data Set Name Prefix . . . . IMSPSA.IM0A.NORDDB

Free Space

Free block freq. factor . . 0

Free space percentage . . . 0

Figure 4-13 Partition defaults for automatic definition

Press Enter, and the PDU defines the partitions for you. There will be a pop-up window showing you the status of the definition process. This includes the number of partitions defined and the elapsed time. Figure 4-14 on page 59 shows the results with the message that four partitions have been added successfully. Note that the partition names generated are different than the

manually created names used before. The information in the RECONS is equivalent to what we generated in our manual definitions. Only the partition names are different.

```

File Edit View Help
-----
Database Partitions                               Row 1 to 4 of 4
COMMAND ==> _____

Select an item by pressing a '/' on the desired line then press Enter.

Database Name . . . . . : NORDDB

Act   Name      Id      Data Set Name Prefix
---   ---      --      ---
1    NORDDAA    1      IMSPSA.IM0A.NORDDB
2    NORDDAB    2      IMSPSA.IM0A.NORDDB
3    NORDDAC    3      IMSPSA.IM0A.NORDDB
4    NORDDAD    4      IMSPSA.IM0A.NORDDB
***** Bottom of data *****

```

4 partitions were added successfully.

Figure 4-14 Automatic partition definition results

### 4.1.7 Changing partition definitions

After initially defining your database, you may discover that your partition definitions were either incorrect or incomplete. If you have not already loaded your database, changing the definitions is very easy. From the Database Partitions panel, which shows you the list of partitions defined, you can select a partition to change by placing a slash (/) in the appropriate Act field and pressing Enter. Alternatively, you could select a partition with a slash, move the cursor under Edit or File, and then press Enter. This opens a window (see Figure 4-15 on page 60 for the Edit pull-down window). You may then select the option you wish. By using this window you may also change the attributes of all partitions simultaneously. In this case do not select a partition by entering a slash before you open the window.

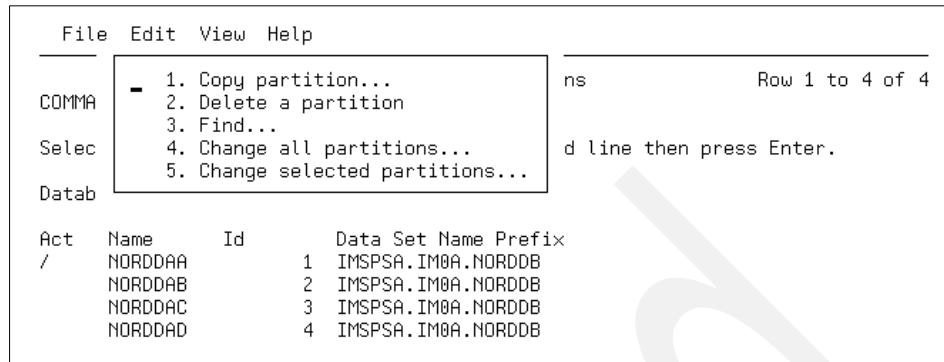


Figure 4-15 Using pull-down window

You can also manage individual partitions with line commands:

- ▶ To open a partition place an O in the Act field of the selected partition.
- ▶ To open its data set groups use a G.
- ▶ To copy the partition use a C.
- ▶ To delete the partition use a D.

If you have to change your partitions after you have already loaded your database, see “Changing existing HALDB databases” on page 257 for further information.

### 4.1.8 Deleting definitions

You may delete partition definitions by using line commands or the pull-down window. A Confirm Deletion panel is presented to you with two options. The difference is only applicable to requests for the deletion of more than one partition. Option 1 shows you the confirm panel for every partition you want to delete. Option 2 shows the panel only once for all partitions. Deleting a partition removes all information from the RECON.

**Note:** If you want to add the partition again, the PDU will not reuse the partition ID of the previously deleted partition. The PDU will assign the next sequential partition ID and your data set names will include the new partition ID.

To delete the partitioned database use Option 3 of the main menu. You get the Delete Database Information panel. There you have to type a slash (/) and press Enter. Then all information about the HALDB database and all its partitions is deleted from the RECONS. Therefore, handle this command with care.

## Batch definition of HALDB

This chapter describes how you can define, change, and delete your HALDB databases and partitions after you have performed DBDGEN by using DBRC batch commands. The commands are INIT, CHANGE, and DELETE. We describe these commands and give examples of RECON listings.

## 5.1 Using the batch interface

Instead of using the Partition Definition utility you may use DBRC batch commands to register HALDB databases and define their partitions. This capability was added by APARs. For the INIT commands, IMS Version 7 must include PQ35893. The INIT commands are included in the base for IMS Version 8. For the DELETE and CHANGE commands, Version 7 requires PQ52858, and Version 8 requires PQ59888.

These batch commands have the same functions as the PDU. With DBRC, however, you have to identify partitions one at a time. There is no automatic definition. You can use any combination of PDU and DBRC commands. When using DBRC batch INIT commands, you must include an IMS DD statement in your job stream. This statement identifies the DBDLIB. Otherwise, you get an error message as shown in Example 5-1.

*Example 5-1 Error message for missing DBDLIB*

---

```
DSP0011I  IMS      DD STATEMENT INVALID OR MISSING
DSP0209I  PROCESSING TERMINATED WITH CONDITION CODE = 12
```

---

Most, but not all, DBRC commands that are used with non-HALDB databases are also used with HALDB.

## 5.2 DBRC initialization commands for HALDB

You can use INIT.DB and INIT.PART DBRC commands for registering HALDB databases with DBRC.

### 5.2.1 INIT.DB

Prior to registering your database, you must run DBDGEN to define the master database. In Example 5-2 a DBDGEN for TESTDB has not been done. The results of an INIT.DB command for TESTDB are shown.

*Example 5-2 Error message if no DBDGEN has been run*

---

```
DSP1050I  DATA BASE NOT DEFINED IN DBD LIBRARY
DSP1050I  DBD=TESTDB
DSP0209I  PROCESSING TERMINATED WITH CONDITION CODE = 12
```

---

You may use the INIT.DB command to register a database with DBRC. A database must be registered with DBRC before you can define its partitions.



The INIT.DB command registers the *master* database with DBRC. It defines the DBD name of the master database, its sharing level, the type of database, whether this database is partitioned by key ranges or a selection exit, and whether it is recoverable. For further keywords see Table 5-1.

Table 5-1 Keywords with INIT.DB

HALDB keywords	Other valid keywords
TYPHALDB HIKEY   PARTSEL	DBD DBTRACK   RCVTRACK GSGNAME ICREQ   NOICREQ NONRECOV   RECOVABL SHARELVL(0   1   2   3)

**TYPHALDB**

TYPHALDB is a keyword that you must use to describe the HALDB databases. It is mutually exclusive with the keywords TYPEIMS for non-HALDB full-function databases and TYPEFP for Fast Path databases.

**HIKEY/PARTSEL**

These mutually exclusive optional keywords identify whether the database uses a partition selection exit or high key values. This setting determines whether the KEYSTRNG parameter on the INIT.PART command defines a partition selection exit string or a high key value. These parameters are only valid with TYPHALDB.

HIKEY specifies that this HALDB uses high key values (see Example 5-3).

PARTSEL(pgmname) specifies the name of the partition selection exit routine.

*Example 5-3 Registering a HALDB database using key ranges*

```
INIT.DB DBD(NORDDB) TYPHALDB HIKEY SHARELVL(3) +  
RECOVABL
```

If you omit both parameters, the default is taken from the DBD. If a partition selection exit routine is specified in the DBD, it is used. If it is not specified in the DBD, key range partitioning (HIKEY) is used.

Parameters specified with the INIT.DB command override the specifications in the DBD.

If you list the RECON after issuing the INIT.DB command, you get the output for your database. Example 5-4 on page 64 shows the results of the command from Example 5-3. You see the master database defined, but no partitions yet. To specify the partitions you have to use another command.

Example 5-4 LIST.RECON

DBD=NORDDB	DMB#=16	CHANGE#=0	TYPE=HALDB
SHARE LEVEL=3	GSGNAME=**NULL**		
PSNAME=**NULL**	DBORG=PHIDAM	DSORG=OSAM	CURRENT PARTITION ID=00000
FLAGS:	COUNTERS:		
RECOVERABLE	=YES	PARTITIONS	=0
		DATA SET GROUP MEMBERS	=1

---

DSP0060I NO PARTITIONS REGISTERED FOR THE DATABASE  
DSP0060I DBDNAME=NORDDB

---

5.2.2 INIT.PART

You can use the INIT.PART command only if the database is already registered with DBRC. The INIT.PART command registers a partition. It creates the HALDB partition structure (a partition record, a partition database record, and one or more DBDS records according to the DBD specification). The INIT.PART fails if the database is being used by the PDU (Example 5-5).

Example 5-5 Error message when HALDB is being used by PDU

DSP1052I	DATABASE NORDDB	IS IN USE BY HALDB PARTITION DEFINITION UTILITY
DSP0209I	PROCESSING TERMINATED WITH CONDITION CODE = 12	

---

Most parameters of the INIT.PART command apply to all the partition DBDSs created as a result of this command. An example is the ICJCL skeleton. This differs from the PDU, where these parameters may be specified separately for each partition DBDS being created. These parameters could be changed later by using the CHANGE.DBDS command (or PDU).

**Restriction:** You can not use an INIT.DBDS command for HALDB data sets. Data sets are always defined by the INIT.PART.

Table 5-2 on page 65 shows valid keywords for the INIT.PART command.

Table 5-2 Keywords for INIT.PART

HALDB keywords	Keywords with the same meaning as in INIT.DBDS for non-HALDB
DBD PART KEYSTRNG DSNPREFX RANDOMZR, ANCHOR, HIBLOCK and BYTES, if PHDAM FBFF and FSPF BLOCKSIZE, if OSAM	GENMAX DEFLTJCL ICJCL, OICJCL NOREUSE / REUSE RECOVJCL RECOVPD RECCVJCL

**DBD**

DBD(name) is a required parameter. It identifies the master database.

**PART**

PART(name) is a required parameter. It identifies the HALDB partition name. The name must be alphanumeric, up to seven characters long, with the first character being alphabetic.

**KEYSTRNG**

KEYSTRNG is an optional parameter you use to specify a partition high key value or a partition selection string for use by a partition selection exit. It may be a character value up to 256 characters long or a hexadecimal value up to 512 characters long.

Character values must be alphanumeric with no embedded blanks or commas unless the string is enclosed by single quotes. They will be folded to upper case unless enclosed in single quotes. Hexadecimal values must be enclosed by single quotes and preceded by the letter X (see Example 5-6).

If no partition selection exit is specified in the HALDB master definition, KEYSTRNG defines the partition high key and is required. The length of the high key must be equal to or less than the root key length. If you define a smaller length, the high key value is padded with hexadecimal 'FF's up to the defined root key length.

If a partition selection exit is defined, KEYSTRNG defines a partition selection string, which is passed to the partition selection routine. Your exit may or may not require a partition selection string. If required, you determine its contents. It may be to 256 bytes.

*Example 5-6 Definition of KEYSTRNG*

KEYSTRNG(00060300002443)

or in hex values:  
KEYSTRNG(X'F0F0F0F6F0F3F0F0F0F0F2F4F4F34')

---

### ***DSNPREFX***

This parameter is required. It specifies the data set name prefix of the partition's data sets. It may be up to 37 characters.

### ***PHDAM***

If your database is PHDAM, you may specify the randomizer module name with RANDOMZR and the values for ANCHOR, HIBLOCK, and BYTES, which correspond to the specifications in a DBD RMNAME parameter where RMNAME=(module name, number of root anchor points, number of control intervals in root addressable area, number of bytes).

If you omit these specifications here, they are obtained from the DBD. If the values in DBD and INIT.PART are different, the values specified by INIT.PART are used.

### ***Free space***

You may enter your specifications for space that is to be left as free space during load or reorganization by using the keywords FBFF (every nth control interval or block is left as free space) and FSPF (free space percentage value per interval or block). The specification applies to all data set groups in the partition.

Example 5-7 defines two partitions named NORD1 and NORD2 in master database NORDDDB. We specify their high key values, data set name prefixes, block sizes, and free space values.

#### ***Example 5-7 INIT.PART***

---

```
INIT.PART DBD(NORDDDB) PART(NORD1) KEYSTRNG(00060300002443) +  
DSNPREFX(IMSPSA.IMOA.NORDDDB) BLOCKSIZE(4096) FBFF(5) FSPF(25)  
INIT.PART DBD(NORDDDB) PART(NORD2) KEYSTRNG(00110500002902) +  
DSNPREFX(IMSPSA.IMOA.NORDDDB) BLOCKSIZE(4096) FBFF(5) FSPF(25)
```

---

After these commands, a LIST.RECON shows that the master database now has two partitions. It lists information about the partitions. Example 5-8 on page 67 shows only the first partition. The master database listing now indicates that the number of partitions is 2.

A database record for the NORD1 partition has been added with TYPE=PART. The DBD name is the partition name. The master database name is also shown. You may check the data set name prefix, free space, and high key specifications. The list also includes the partition ID, the previous partition, and the following

*Example 5-8 RECON listing after INIT.PART (part 1)*

The INIT.PART command also creates all of the DBDS records for your partitions. Certain values have automatically been created. Example 5-9 on page 68 lists the DBDS records.

The DBDS record includes the following information:

- ▶ The data set name field (DSN) is composed of the data set name prefix field followed by the suffix .A00001. The suffix is the concatenation of dot, the DSGROUP parameter in the DBD source (A, if you have only one data set group), and the generated 5-byte partition ID.
- ▶ The DBD field is the partition name you chose.
- ▶ The DD name field (DDN) is the concatenation of the partition name and the DSGROUP value. A partition name must be unique in a set of RECONS.

In the listing two additional DBDS records also appear. The first is for the indirect list data set (ILDS). It is suffixed by .L00001. The second is for the PHIDAM primary index and is suffixed by .X00001. You may not change these suffixes.

Example 5-9 RECON listing after partition definition (part 2)

-----			
DBDS			
DSN=IMSPSA.IMOA.NORDDB.A00001	TYPE=PART		
DBD=NORD1	DDN=NORD1A	DSID=001 DBORG=HIDAM DSORG=OSAM	
CAGRP=**NULL**	GENMAX=2	IC AVAIL=0 IC USED=0	DSSN=00000000
NOREUSE	RECOVPD=0	OTHER DDN=**NULL**	
DEFLTJCL=**NULL**	ICJCL=ICJCL	OICJCL=OICJCL	RECOVJCL=RECOVJCL
RECVJCL=ICRCVJCL			
FLAGS:	COUNTERS:		
IC NEEDED	=OFF		
RECOV NEEDED	=OFF		
RECEIVE NEEDED	=OFF	EEQE COUNT	=0
-----			
DSN=IMSPSA.IMOA.NORDDB.L00001	TYPE=PART		
DBD=NORD1	DDN=NORD1L	DSID=003 DBORG=INDEX DSORG=VSAM	
FLAGS:	COUNTERS:		
RECOV NEEDED	=OFF	EEQE COUNT	=0
-----			
DBDS			
DSN=IMSPSA.IMOA.NORDDB.X00001	TYPE=PART		
DBD=NORD1	DDN=NORD1X	DSID=005 DBORG=INDEX DSORG=VSAM	
FLAGS:	COUNTERS:		
RECOV NEEDED	=OFF	EEQE COUNT	=0
-----			

### 5.3 DBRC change commands for HALDB

You can use the CHANGE.DB, CHANGE.PART and CHANGE.DBDS DBRC commands for changing the information related to HALDB databases in RECON.

### 5.3.1 CHANGE.DB

HALDB support for the CHANGE.DB command was added to IMS Version 7 by APAR PQ52858 and to IMS Version 8 by APAR PQ59888. The CHANGE.DB command is used to change the information about a database or partition. You may use the CHANGE.DB command with the DBD for the master database name as well as with the DBD for a partition database name. These are shown in Example 5-10.

*Example 5-10 CHANGE.DB for master and partition database*

---

```
CHANGE.DB DBD(NORDDDB) NONRECOV
```

```
CHANGE.DB DBD(NORD1) PINIT
```

---

In the first case, changes apply to all partitions. In the second, changes only apply to the specified partition. The keywords you may use depend on whether you specify the master or partition database, as shown in Table 5-3.

*Table 5-3 Keywords for CHANGE.DB*

Valid keywords for:	Master	Partition
AUTH I NOAUTH	NO	YES
BACKOUT I NOBACK	NO	YES
READON I READOFF	NO	YES
SHARELVL	YES	NO
NONRECOV I RECOVABL	YES	NO
TYPEFP I TYPEIMS	NO	NO
GSGNAME I NOTCOVER	YES	NO
RCVTRACK I DBTRACK	YES	NO
UNAUTH	NO	YES
PINIT I NOPINIT	YES	YES
PARTSEL I HIKEY	YES	NO
ICREQ I NOICREQ	YES	NO

### 5.3.2 CHANGE.PART

The CHANGE.PART command was added to IMS Version 7 by APAR PQ52858 and to IMS Version 8 by APAR PQ59888. You can use the CHANGE.PART command to change the attributes of a partition. The changes apply to all the DBDSs of the partition. This command may change all of the attributes that you may set with an INIT.PART command (see “INIT.PART” on page 64). The following example changes the data set name prefix of all DBDSs of partition NORD1 to IMSPSA.IM0B.NORDDDB (Example 5-11 on page 70).

*Example 5-11 CHANGE.PART to change the DSN prefix*

---

```
CHANGE.PART DBD(NORDDDB) PART(NORD1) DSNPREFIX(IMSPSA.IMOB.NORDDDB)
```

---

We can change some attributes, such as skeletal JCL member names, for a single DBDS of a partition by using a CHANGE.DBDS command. It would be a good practice to always do a LIST.PART before the CHANGE.PART command.

Note that if you change the randomizer parameters in any way, you should have unloaded the partition first. If you have not, you are not able to read the partition. The change of the randomizer parameters turns the *partition init needed* flag on and the partition is unreadable.

### 5.3.3 CHANGE.DBDS

Use the CHANGE.DBDS command to change some of the attributes of a single DBDS.

The information is stored in a DBDS record in RECON. You cannot change the data set name or DD name with this command. Keywords DDNNEW or DSN are not allowed for HALDB data sets. The naming conventions apply to all DBDSs of a partition and cannot be overridden for a single DBDS.

You have to distinguish between data data sets and index or ILDS data sets when using other keywords.

For ILDS and index data sets only, the keywords listed in Example 5-12 are valid.

*Example 5-12 Valid keywords for ILDS and PHIDAM primary index*

---

```
ADDEQE | DELEQE  
RECOV | NORECOV
```

---

This means that you may only change the information about error queue elements and the recovery needed status for PHIDAM primary index and ILDS data sets.

For data data sets you can use more keywords. They are listed in Example 5-13.

*Example 5-13 Valid keywords for data data sets in HALDB partitions*

---

```
ADDEQE | DELEQE  
DEFLTJCL | NODEFLT  
GENMAX  
ICJCL  
ICON | ICOFF  
NOREUSE | REUSE
```

---



```
OICJCL
RECOV | NORECOV
RECOVJCL
RECOVPD
RECVJCL
```

---

Example 5-14 is an example of changing the status of a data set from recovery needed to recovery needed off.

*Example 5-14 CHANGE.DBDS*

```
CHANGE.DBDS DBD(NORD1) DDN(NORD1A) NORECOV
```

---

## 5.4 DBRC delete commands for HALDB

You can use DELETE.DB and DELETE.PART DBRC commands for deleting HALDB databases from RECON.

### 5.4.1 DELETE.DB

HALDB support for the DELETE.DB command was added to IMS Version 7 by APAR PQ52858 and to IMS Version 8 by APAR PQ59888. Use a DELETE.DB command to delete a database from the RECONS. The command also deletes all information related to the database that has been recorded in the RECONS.

Example 5-15 shows the deletion of database NORDDDB and its related information from the RECONS. A partition name cannot be specified. If the database is a HALDB database, the master and all its partitions are deleted. If the database or any of its partitions is in use, the command fails, and none of the RECON records are deleted.

*Example 5-15 Delete a database*

```
DELETE.DB DBD(NORDDDB)
```

---

### 5.4.2 DELETE.PART

The DELETE.DB command was added to IMS Version 7 by APAR PQ52858 and to IMS Version 8 by APAR PQ59888. Use a DELETE.PART command to delete a partition from the RECONS. The command also deletes all information related to that partition that has been recorded in the RECONS. Note that if you delete a partition, you will not be able to use the recovery utility to put the partition back.

Example 5-16 shows the deletion of partition NORD1 and its related information from the RECONS. If the partition or any other partition affected by the deletion is authorized, the command fails and none of the RECON records are deleted.

*Example 5-16 Delete a partition*

---

```
DELETE.PART DBD(NORDDB) PART(NORD1)
```

---

**Restriction:** You cannot use a DELETE.DBDS command to delete a partition DBDS.

## Partition initialization

All HALDB partitions must be initialized before they are used. This chapter describes the partition initialization function and the utilities that initialize partitions. We provide examples showing the use of these utilities.

## 6.1 Partition initialization function

HALDB partitions must be initialized before they may be used. This function does not exist with non-HALDB databases. Initialization is done either by the Database Prereorganization utility or the Database Partition Data Set Initialization utility. Alternatively, the IBM High Performance Load tool may be used to do partition initialization. Initialization makes a partition usable but does not place any database segments in the partition. After initialization a partition is empty.

When creating a new HALDB database or migrating a non-HALDB database to HALDB, you may create one or more partitions that contain no data. For example, you might have a partition for root keys ranging from 4,000,001 through 5,000,000, but not have any roots in this range. Later, a batch program or online transaction might attempt to get or insert a root in this range. Partition initialization ensures that the partition may be processed by such a program.

Partition initialization is required for a partition in the following three cases:

- ▶ The partition is loaded by an initial load program. An initial load program uses PROCOPT=L.
- ▶ The partition is loaded by HD Reload when migrating from a non-HALDB database.
- ▶ The partition is accessed by an application program before any segments are loaded in it by initial load or a reload utility.

Initialization is not required for a normal reorganization reload. A partition does not have to be initialized before a non-migration use of HD Reload loads segments in the partition. This is true even when the partition's data sets have been deleted and redefined after the unload. On the other hand, initialization is required for partitions with no data.

Partition initialization is used with all HALDB databases including secondary indexes (PSINDEX).

Partition initialization writes the partition ID number and the initial reorganization number in PHDAM and PHIDAM partitions. The initial reorganization number is one. The numbers are written in the first four bytes of the first block of the first data set. This is the bit map block. The first four bytes are used for the DBRC usage indicator in non-HALDB databases. Since DBRC registration is always required with HALDB, the DBRC usage indicator function is not used with HALDB. For PHDAM partitions, a dummy record is written and deleted. For PHIDAM partitions, the high keys (all X'FF's) record is written. This record is stored in every PHIDAM partition. For PSINDEX partitions, one record is written and then deleted. These actions make the high-used-RBA non-zero.

## 6.2 DBRC flags used with partition initialization

Partition initialization uses two flags in the RECONS. These are the *partition init needed* and the *image copy needed* flags.

### 6.2.1 Partition initialization required (PINIT) flag

A partition in a HALDB database may have a *partition init needed* (PINIT) flag set in the DBRC RECONS. The flag indicates that the partition requires initialization. When either the Database Prereorganization utility or the Partition Initialization utility is run against the database, it initializes only those partitions with their PINIT flags set. There is one exception to this. We discuss this exception in 6.4.1, “Unconditional partition initialization” on page 78.

The PINIT flag is set in five ways:

- ▶ The flag is set when a partition is defined.
- ▶ The flag is set when explicitly turned on by a DBRC CHANGE.DB command. The command specifies the PINIT keyword. Example 6-1 shows the command to turn on the flag for all partitions in the PEOPLE database. Example 6-2 shows the command to turn on the flag for only the PEO01 partition in the PEOPLE database.
- ▶ The flag is set when making changes to partition boundaries such that data may be moved either into the partition or out of the partition. This only applies to databases that use key range partitioning.
- ▶ The flag for a partition is set when its partition string is changed. This only applies to databases using a partition selection exit routine.
- ▶ The flags for all partitions are set when the partition selection exit routine name is changed. This only applies to databases using a partition selection exit routine.

The setting of the flag is shown in a listing of the partition database record in the RECONS as PARTITION INIT NEEDED =NO or PARTITION INIT NEEDED =YES.

*Example 6-1 Setting the PINIT flag for master database PEOPLE*

---

```
CHANGE.DB DBD(PEOPLE) PINIT
```

---

*Example 6-2 Setting the PINIT flag for partition PEO01*

---

```
CHANGE.DB DBD(PEO01) PINIT
```

---

## 6.2.2 Image copy needed flag

Partition initialization sets the *image copy needed* flag for each partition that is initialized. The image copy needed flag does not prevent authorization for initial loads. It does prevent authorizations for updates. If you initialize but do not initially load a partition, it must be image copied before any subsystem will be given authorization to update it.

## 6.3 Database Prereorganization utility

The Database Prereorganization utility (DFSURPR0) may be used to initialize partitions. This function was added to the utility in IMS Version 7.

As when used with non-HALDB databases, the utility also creates a control data set (CDS). The CDS is not required by other utilities when they process HALDB databases. With HALDB the CDS is merely used to disable any attempted use of the Database Scan and Database Prefix Resolution utilities. Nevertheless, the DFSURCDS DD statement is required with the Prereorganization utility.

The DBIL, DBR, or DBM control statement is used to indicate which database or databases should have their partitions initialized. All of these control statements do the same initialization for HALDB partitions. We used the DBIL statement in our testing. You may specify multiple databases on the statement. Database names must be eight bytes, left justified, and separated by commas. Example 6-3 shows the control statement we used to initialize the partitions in databases PEOPLE and PEOSKSI. Since PEOPLE is only six bytes, we placed two blanks before the following comma.

*Example 6-3 Prereorganization DBIL= control statement*

---

```
//SYSIN      DD *  
DBIL=PEOPLE  ,PEOSKSI  
/*
```

---

A DFSVSAMP DD statement with definitions for buffer pools is required when running Prereorganization to initialize partitions. The buffer pools must contain buffers large enough to hold the OSAM blocks or VSAM CIs that are written during the process. Large pools are not advantageous.

## 6.4 Database Partition Data Set Initialization utility

The Database Partition Data Set Initialization utility (DFSUPNT0) may be used to initialize partitions. It is an alternative to the Database Prereorganization utility.

There are two ways to execute this utility. It may use a utility (ULU) region under the control of DFSRRC00 or it may execute as a stand-alone program. Example 6-4 shows the JCL that may be used to execute as a utility region. Example 6-5 shows the JCL that may be used to execute as a stand-alone program. When executed as a utility region, the utility can initialize the partitions for only a single database. The database is specified as the third subparameter in the PARM specification. In Example 6-4 the partitions in database PEOPLE are initialized.

*Example 6-4 Using a ULU region with DFSUPNT0*

---

```
//PARTINIT EXEC PGM=DFSRRC00,REGION=1024K,
//          PARM='ULU,DFSUPNT0,PEOPLE,,,,,,,,,Y,N'
//DFSRESLB DD DSN=IMSPSA.IMS0.SDFSRESL,DISP=SHR
//IMS      DD DISP=SHR,DSN=JOUK03.HALDB.DBDLIB
//RECON1   DD DISP=SHR,DSN=IMSPSA.IM0A.RECON1
//RECON2   DD DISP=SHR,DSN=IMSPSA.IM0A.RECON2
//RECON3   DD DISP=SHR,DSN=IMSPSA.IM0A.RECON3
//DFSVSAMP DD *
VSRBF=8192,10
IOBF=(4096,10)
/*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSIN    DD DUMMY
```

---

When executed as stand-alone, the databases to be initialized must be specified in the SYSIN data set. Multiple databases may be initialized with one execution. This is shown in Example 6-5 where the partitions in both PEOPLE and PEOSKSI are initialized.

*Example 6-5 Using a stand-alone region with DFSUPNT0*

---

```
//PARTINIT EXEC PGM=DFSUPNT0,REGION=1024K
//DFSRESLB DD DSN=IMSPSA.IMS0.SDFSRESL,DISP=SHR
//IMS      DD DISP=SHR,DSN=JOUK03.HALDB.DBDLIB
//RECON1   DD DISP=SHR,DSN=IMSPSA.IM0A.RECON1
//RECON2   DD DISP=SHR,DSN=IMSPSA.IM0A.RECON2
//RECON3   DD DISP=SHR,DSN=IMSPSA.IM0A.RECON3
//DFSVSAMP DD *
VSRBF=8192,10
IOBF=(4096,10)
/*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSIN    DD *
PEOPLE
PEOSKSI
```

---

/\*

---

A DFSVSAMP DD statement with definitions for buffer pools is required when running the Database Partition Data Set Initialization utility. The buffer pools must contain buffers large enough to hold the OSAM blocks or VSAM CIs that are written during the process. Large pools are not advantageous

This utility does not create the control data set (CDS) that the Prereorganization utility creates. This is not a problem since the CDS is not used with HALDB.

### 6.4.1 Unconditional partition initialization

The Database Partition Data Set Initialization utility may be used to initialize partitions even when their *partition init needed* (PINIT) flag is not set. This function is called unconditional partition initialization. It was added to IMS Version 7 by APAR PQ49638 and to IMS Version 8 by PQ55002. It allows a user to delete and define all of the data sets in a database and then initialize them without having to issue any DBRC CHANGE.DB commands.

As mentioned in 6.1, “Partition initialization function” on page 74, initialization is not required for a normal reorganization reload if each partition contains data. This is true even when the partitions’ data sets have been deleted and redefined after the unload. Partition initialization is required for empty partitions. In these cases, unconditional partition initialization may be useful.

Unconditional partition initialization is invoked by including an INITALL control statement after a DFSOVRDS DD statement. Example 6-6 shows these statements.

*Example 6-6 DFSOVRDS DD statement with the INITALL control statement*

---

```
//DSOVRDS DD *  
INITALL  
/*
```

---

Unconditional partition initialization is not supported by the Database Prereorganization utility (DFSURPR0). This capability is only provided by the Database Partition Data Set Initialization utility.



## Partition selection

In this chapter we explain your choices for partition selection. We describe the advantages and disadvantages for using key range partitioning versus the use of partition selection exit routines. We provide guidance on writing partition selection exit routines.

## 7.1 Choosing the type of partition selection

HALDB allows you to choose between two types of partition selection. Key range partitioning assigns a range of keys to each partition. Alternatively, you may write a partition selection exit routine that uses its own criteria for assigning keys to partitions. In both types, only the key of the root segment is used to assign a record to a partition. Both types may be used with any HALDB database type. These are PHDAM, PHIDAM, and PSINDEX.

### 7.1.1 Key range partitioning

Key range partitioning is usually preferred. With key range partitioning, a high key is assigned to each partition. The partition holds all records with keys higher than the previous partition and up to the value of its high key.

Key range partitioning is the simpler method to implement because you do not have to write an exit routine. You only have to assign a high key to each partition.

In PHIDAM and PSINDEX databases, the records are in key sequence across the entire database. This makes them compatible with HIDAM and non-HALDB secondary index databases. When a partition selection exit routine is used for PHIDAM or PSINDEX databases, records are in key sequence within a partition, but not across partitions. This would make these databases incompatible with HIDAM and non-HALDB secondary index databases. Applications that require database records to be in key sequence would not function correctly.

IMS is aware of the impact of the addition of partitions, their deletion, or changes in their boundaries. IMS can determine which partitions require initialization after these changes. Figure 7-1 illustrates this. In the figure, the change of the high key for partition B from 40000 to 50000 would require the movement of records from partition C to partition B. Both of these partitions would have to be unloaded before the change. Both would have to be initialized and reloaded. IMS sets the *partition init needed* flag for both partitions when the high key for partition B is changed. This is not true when you use an exit routine. The logic of the exit routine determines which partitions require initialization after these changes. You are responsible for understanding this and initializing the correct partitions.

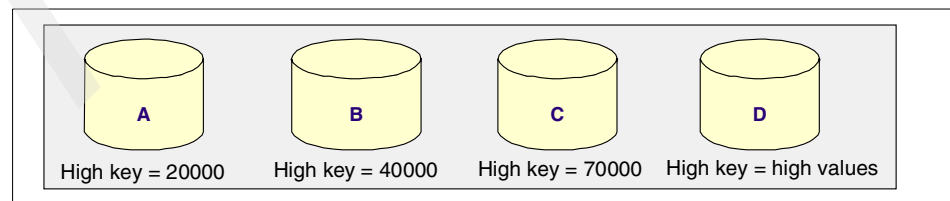


Figure 7-1 Key range partitioning

We recommend that you assign a high values key to a partition. If you do not and if an application attempts to insert a root with a key higher than the highest key defined, the call will receive an FM status code. Most applications are not coded to handle this status code. A high values key is one with X'FF' for each byte.

There is a case where key range partitioning will require the addition of partitions or changes to their key ranges over time. This occurs when records that are added to the database have increasing keys. Examples include keys created by incrementing the previous key and keys based on dates or times. As records are added, you will need to split the last partition into multiple partitions. As records are deleted, you may want to consolidate partitions containing lower values.

### 7.1.2 Partitioning with an exit routine

You may use a partition selection exit routine to assign records to partitions. Only the key of the root segment is available from the database record for the exit routine. Since writing and maintaining the exit routine increases your responsibilities, we recommend that you use an exit routine only if key range partitioning does not meet your needs.

An exit routine would allow you to assign records to a partition based on a value other than the high order part of the key. Figure 7-2 illustrates a use of an exit routine. In this figure a warehouse code is in the fifth and sixth bytes of the eight-byte key. The partition selection exit routine assigns records to partitions based on the warehouse code. The installation plans on processing data by warehouse. By isolating data for a warehouse in one partition, they can simplify the processing and improve its performance.

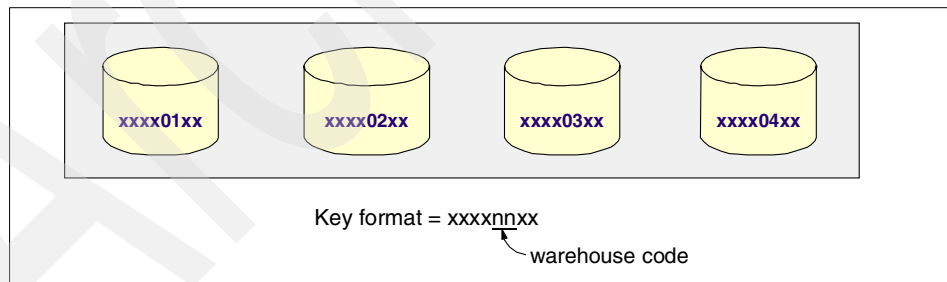


Figure 7-2 Partitioning with an exit routine

The HALDB Conversion and Maintenance Aid product includes an exit routine that will do this kind of partition selection. If you use it, you do not have to write the exit routine. You only have to specify what part of the key is to be used and the values for each partition.

Another use of a partition selection exit routine is to isolate some database records because of their special characteristics. For example, the sizes of most records in a PHDAM database may be fairly uniform, but others may be much larger. These unusually large records could otherwise cause space use problems within partitions. If the keys of these large records are known, an exit routine could recognize their keys and place them in a partition with different space characteristics. The partition might have many fewer records spread across the same amount of space. It might even have its own specialized randomization routine.

In the last paragraph in 7.1.1, “Key range partitioning” on page 80, we discuss a database whose key values continued to increase. With key range partitioning this could present a problem. You would have to continually adjust the partitions. An exit routine might solve this problem. If there is no need for key sequence processing, an exit routine could distribute the records across partitions evenly. For example, it might assign the records based on the low order part of the key. Then each partition would always receive a consistent percentage of the records.

Databases whose records must be processed in key sequence must use key range partitioning. Typically, a partition selection exit routine would assign records such that moving from one partition to the next might cause a decrement in the key value. The keys are sequential within the partitions but not from one partition to the next. This may have an impact on the application if it expects the keys to always increase. Records in HIDAM databases and non-HALDB secondary indexes are always in key sequence. When they are migrated to PHIDAM or PSINDEX do not use a partition selection exit routine unless you are sure that key sequence processing is not required.

## 7.2 Writing partition selection exit routines

You may write a partition selection exit routine to select partitions for database records in a HALDB database. The exit routine name must match what you specify in the Partition Definition utility or the DBRC INIT.DB command. The exit routine provides several functions. Documentation on writing the exit routine can be found in the manuals *IMS Version 7 Customization Guide*, SC26-9427, and *IMS Version 8 Customization Guide*, SC27-1294.

### 7.2.1 Partition selection exit routine functions

The partition selection routine functions are divided into two categories. The first category is partition selection. The functions in partition selection are:

- ▶ Select target partition.
- ▶ Select first partition.

- ▶ Select next partition.

The second category of functions is for partition structure management awareness. These functions are:

- ▶ Partition structure initialization
- ▶ Partition structure termination
- ▶ Partition structure modification

## **Partition selection functions**

When a program issues a call, IMS determines which of these functions is required and invokes the exit routine for the function.

### ***Select target partition***

This function is used for calls qualified on the root segment. The root key is passed to the exit routine. The routine selects the partition where the key is assigned.

### ***Select first partition***

This function is used for unqualified calls with no previous position in the database. The routine selects the first partition.

### ***Select next partition***

This function is used when IMS reaches the end of a partition. IMS passes the partition ID of the last partition processed to the exit routine. The exit either selects the next partition or indicates that the end of the database has been reached.

## **Partition structure management awareness functions**

These functions allow the exit routine to adjust to changes in the environment, such as the addition or deletion of partitions.

### ***Partition structure initialization***

This function is invoked when a database is first referenced. The exit routine can do any initialization processing it requires. For example, it could load a module or acquire storage.

### ***Partition structure termination***

This function is invoked when a database is no longer used. For example, when a /DBR command is issued for the database, the exit is driven for this function. The exit routine can do any termination processing it requires. For example, it could delete any modules it has loaded or release any storage it has required.

### ***Partition structure modification***

This function is invoked when DBRC authorization determines that the database definition has been changed. Examples of these changes include adding a new partition, deleting a partition, changing a partition string value, or changing the randomization parameters for a PHDAM partition. The exit routine can make any necessary adjustments due to the change.

## **7.2.2 Information passed to and from the exit routine**

The information passed to the exit routine depends on the function invoked. Information is passed in two areas. Register 1 points to a parameter list that contains the addresses of these areas.

The Exit Communication Area contains an indication of the function requested, provides the input data, and contains storage for the exit routine to indicate its chosen action. For example, a select target partition request includes the key of the root segment. In this case, the exit routine would place the selected partition ID in the feed back field. This area is mapped by the DFSPECA DSECT, which is documented in the HALDB Partition Selection Exit Routine chapter of the *IMS Version 8 Customization Guide*, SC27-1294, and included in the DFSPSE00 sample program.

The second area contains the Partition Definition Area prefix and entries. The prefix contains a pointer to the first entry and the number of entries. It also contains five words of storage that are available for your use. There is an entry for each partition. They are in adjacent storage locations. Each entry contains information about a partition including its name, partition ID, partition string length, and partition string address. The DFSPDA DSECT maps the prefix and the DFSPDAE DSECT maps an entry. These DSECTs can be found in *IMS Version 8 Customization Guide*, SC27-1294, and they are also included in the DFSPSE00 sample program.

## **7.2.3 Sample exit routine (DFSPSE00)**

IMS provides a sample partition selection exit routine (DFSPSE00). In IMS Version 7 it is in the ADFSSRC library. In IMS Version 8 it is in the ADFSSMPL and SDFSSMPL libraries. This sample is not intended for actual use. It implements key range partitioning by using the partition string for each partition as its high key. If you want this type of partitioning there is no need to have an exit routine. The sample demonstrates the interface and control blocks used by the exit. We highly recommend that you examine this sample before writing your own exit routine.

## User modification to the sample exit routine

The sample could be the basis for writing a routine to support other types of partitioning. For example, if you wish to implement partitioning as shown in Figure 7-2 on page 81, you can modify the sample. The modification needs to compare the warehouse code field in the key with the partition strings. Each warehouse code would be the partition string for its partition. Example 7-1 shows the modification to DFSPSE00 that we made to implement this partitioning scheme.

The modification is in the select target partition function of the routine that is at label PSETRGT. The modification is surprisingly easy. It requires only the addition of one instruction. This is a load address (LA) instruction to change the address for the compare. Since the field with the warehouse code is at offset four in the key, we add four bytes to the key address. We do not have to modify the length of the compare. The module bases the compare length on the size of the largest partition string, not the size of the key as it is defined in the DBD. This is what our partitioning scheme needs. When defining the partitions for use with this exit routine, we specify two-byte partition strings. Each string is a two-byte warehouse code.

### Example 7-1 Modification to DFSPSE00

```
***** 04340000
*      SELECT TARGET PARTITION                                * 04350000
*                                                                * 04360000
*      THE PARTITION EXIT COMMUNICATION AREA (PECKEY) SPECIFIES * 04370000
*      THE CURRENT DL/I CALL KEY. THIS KEY IS USED TO SELECT THE * 04380000
*      TARGET PARTITION. THE PARTITION DEFINED WITH A HIGH KEY   * 04390000
*      EQUAL TO OR GREATER THAN THE CURRENT DL/I CALL KEY IS    * 04400000
*      SELECTED.                                                * 04410000
*                                                                * 04420000
*      PARTITION ID RETURNED TO IMS FOR SELECTED PARTITION      * 04430000
***** 04440000
PSETRGT DS      OH                                           04450000
        LTR     R7,R7          PKT EXIST                     04460000
        BZ      PSEE804        NO, ERROR                     04470000
        L       R0,PDAUSR3     NUMBER OF PKT ENTRIES         04480000
        L       R2,PECKEY      DL/I CALL KEY ADDRESS         04490000
***** 04491001
* PSEWAREH MODIFICATION                                       04491101
* BEGIN MODIFICATION TO SELECT PARTITION ON SUBSET OF KEY    04491202
* SUBSET OF KEY IS AT OFFSET 4                                04491302
*                                                                04491402
*      LA       R2,4(,R2)                                       04491501
*                                                                04491602
* THE IC INSTRUCTION BELOW INSERTS THE PARTITION STRING LENGTH 04491702
* IN THE EX INSTRUCTION WHICH IS USED TO DO THE COMPARE      04491802
* OUR PARTITION DEFINITION USES ONLY TWO BYTES FOR THE STRINGS 04491902
* END OF MODIFICATION TO SELECT PARTITION ON SUBSET OF KEY    04492001
***** 04493001
```

	SR	R3,R3		04500000
		SPACE		04510000
PSET3000	DS	OH		04520000
	IC	R3,PKTKEYL	KEY LENGTH -1	04530000
	EX	R3,PTKEY	COMPARE PART HI-KEY	04540000
	BNH	PSET4000	PARTITION FOUND	04550000
	AR	R7,R6	NEXT PKT ENTRY	04560000
	BCT	R0,PSET3000	CHECK NEXT ENTRY	04570000
	B	PSEE402	ERROR	04580000
		SPACE		04590000
PSET4000	DS	OH		04600000
	LH	R1,PKTID	TARGET PARTITION ID	04610000
	STH	R1,PECFDB2	.	04620000
		SPACE		04630000
		*****		04640000
*		SELECT TARGET PARTITION PROCESSING COMPLETE		* 04650000
*		RETURN TO IMS WITH PECRC SET TO ZERO		* 04660000
		*****		04670000
	B	PSEEXIT	RETURN TO IMS	04680000

---

DFSPSE00 contains a write to operator (WTO) instruction in its partition structure initialization function that is at label PSEINIT. This could be useful in a test environment, but should be eliminated before the exit routine is used in a production environment. The instruction may either be deleted from the code or converted to a comment.





## Part 2

# Migration

In this part we describe the process used to migrate databases to HALDB. We include secondary indexes and logically related databases. We provide examples of the migration of databases.

The HALDB Conversion and Maintenance Aid is a product that simplifies the migration process and the maintenance of HALDB databases. We describe this tool and show you how to use it.



## Migration from non-HALDB to HALDB

In this chapter we describe the process for migrating a current non-HALDB database to HALDB. We describe the different processes used for migrating databases with logical relationships or secondary indexes and those that use PDB. We show you how to migrate user-partitioned databases.

## 8.1 General migration considerations

A HDAM or HIDAM database may be migrated to HALDB with the following steps:

- ▶ Unload of the existing database using HD Unload and specifying MIGRATE=YES
- ▶ DBDGEN as a HALDB database
- ▶ Deletion of database information from the RECONs
- ▶ Definition of the partitions
- ▶ Allocation of database data sets
- ▶ Partition initialization
- ▶ Reload as a HALDB database using HD Reload
- ▶ Image copy of the database data sets

Secondary indexes are created when their indexed database is migrated. They are created with the following steps:

- ▶ DBDGEN as a HALDB secondary index
- ▶ Deletion of secondary index database information from the RECONs
- ▶ Definition of partitions
- ▶ Allocation of database data sets
- ▶ Sort of output file from the unload of the indexed database
- ▶ Load of the secondary index using HD Reload
- ▶ Image copy of the database data sets

The migration of databases without logical relationships or secondary indexes is explained in 8.3, “Migrating simple databases” on page 96.

Changed and additional steps for the migration of databases with secondary indexes are explained in 8.5, “Migrating databases with secondary indexes” on page 101.

If databases are logically related, they must be migrated together. Changed steps for the migration of these databases are explained in 8.6, “Migrating databases with logical relationships” on page 114.

### 8.1.1 DBD changes

You must modify the DBD for any database that you migrate to HALDB. This section explains the modifications that are required.

## **HIDAM index DBDs**

There is no DBD for a PHIDAM primary index. When a HIDAM database is migrated to PHIDAM, the DBD for the HIDAM Index is discarded. IMS gets the information required to generate the PHIDAM primary index from the PHIDAM DBD.

## **DBD statement**

There are the following three changes to the DBD statement for HALDB:

- ▶ There are new values for the ACCESS parameter. They are PHDAM, PHIDAM, and PSINDEX. The value should be changed to the appropriate value.
- ▶ The optional PSNAME parameter has been added. It is used to specify a default partition selection exit routine. If you are not going to use the exit routine, do not specify this parameter.
- ▶ The RMNAME parameter has a slightly different meaning. With PHDAM databases it is optional. It is used to define default randomizer values for partitions. These values may be changed for individual partitions when they are defined.

## **DATASET statement**

The DATASET statement is not used in HALDB DBDs. The entities it defines for non-HALDB databases are handled differently with HALDB as follows:

- ▶ DDNAMEs are created by the definition of partitions.
- ▶ Data set groups are defined by the DSGROUP parameter on SEGM statements.
- ▶ Free space and OSAM block size specifications are made when partitions are defined.
- ▶ The SCAN parameter is not specified with HALDB DBDs. HALDB operates as if SCAN=0 were specified. This is also the recommended value for non-HALDB databases.

## **SEGM statement**

There are the following three changes to the SEGM statement for HALDB:

- ▶ The valid specifications for the PTR parameter have changed.  
  
HALDB does not use hierarchic pointers. Any use of HIER, H, HIERBWD, or HB keywords with the PTR parameter must be changed to TWIN, T, TWINBWD, or TB.  
  
PHIDAM does not support the use of twin forward only (TWIN or T) pointers for root segments. If you have HIDAM roots using twin forward only pointers,

the keyword for the PTR parameter on the SEGM statement should be changed to NOTWIN, NT, TWINBWD, or TB.

HALDB does not use symbolic pointers. Changes required for logical relationships are explained in “Changing pointer options for logical relationships” on page 115.

- ▶ The specification of data set groups for HALDBs is done with the DSGROUP parameter.

You may not want to maintain multiple data set groups. If you do want multiple data set groups, you must specify the DSGROUP parameter on the SEGM statement for any segment that is not in the first data set group. The valid values for DSGROUP are the letters A through J. A is the first data set group, B is the second, and J is the tenth.

- ▶ The BYTES parameter for a secondary index segment must be increased by four bytes if a /SX field is used as a subsequence field. We explain this further in “/SX subsequence fields” on page 102.

### **LCHILD statement**

The LCHILD statement is not used to define the primary index in a HIDAM DBD. When converting a HIDAM DBD to PHIDAM, this LCHILD statement should be deleted.

When converting a secondary index DBD to a HALDB PSINDEX DBD, there are three possible changes required:

- ▶ If PTR=SYMB is specified in a secondary index, it must be changed to PTR=SNGL or omitted. PTR=SNGL is the default and only valid specification for PSINDEX LCHILD statements.
- ▶ If PTR=SYMB is specified in a HDAM or HIDAM indexed database, it must be changed to PTR=INDX.
- ▶ An RKSIZE parameter must be specified on the LCHILD statement in the secondary index DBD. This is the size of the root's key in the target database.

When converting a logical relationship using symbolic pointing, you must omit the PTR=SYMB specification on the LCHILD statement for the logical relationship.

The changes for LCHILD statements used to define logical relationships using virtual pairing are explained in “Making changes to the DBDs for physical pairing” on page 116.

### **XDFLD statement**

HALDB does not support shared secondary indexes. If the CONST parameter is specified in an XDFLD statement of an indexed database, it must be deleted. The

CONST parameter is used to specify the character associated with a shared secondary index. Each HALDB secondary index must be stored in its own secondary index database. Separate PSINDEX databases must be defined for each shared secondary index being converted to HALDB.

### **FIELD statement**

The BYTES parameter of the sequence field for a secondary index segment must be increased by four bytes if a /SX field is used as a subsequence field. We explain this further in “/SX subsequence fields” on page 102.

## **8.2 HALDB Migration Aid utility (DFSMAID0)**

The HALDB Migration Aid utility (DFSMAID0) may be used to understand partitioning requirements for a database. The utility reads an existing database and reports on the sizes and boundaries of potential HALDB partitions. The existing database may be HDAM, HIDAM, PHDAM, PHIDAM, a non-HALDB secondary index, or a HALDB secondary index (PSINDEX). HDAM, HIDAM, and secondary index databases are read to understand the requirements for a migration. PHDAM, PHIDAM, and PSINDEX databases are read to understand the requirements for repartitioning an existing HALDB database.

The utility reports for non-HALDB secondary indexes must be used with care. Some of the information reported is not accurate. We explain the special considerations for using DFSMAID0 with secondary indexes in 8.5.2, “Estimating the sizes of secondary index partitions” on page 107.

You choose the type of analysis that the utility does. There are three options. First, you may specify the high key values for potential partitions. The utility reports the space required for each of these partitions. Second, you may specify the maximum number of bytes of segment data stored in each partition. The utility reports the number of partitions required and their high keys. Third, you may specify the number of equal-sized partitions to be used. The utility reports the space required and the high key for each partition. In all cases, the utility reports the total amount of space required for all segments in the database.

### **8.2.1 Reports**

There is a separate report for each partition and one for the database. Example 8-1 on page 94 shows a typical report for a partition. The lowest key and highest key in the partition are reported in both character (EBCDIC) and hexadecimal formats. Each segment type in the database is listed. The segments column reports the number of segments. The bytes column reports the number of bytes those segments require in HALDB. The prefix-incr column

reports the increase in sizes for the segments due to growth in the size of their prefixes. The length-incr column reports the increase reports on the space required to store the new physical logical children when converting from virtual pairing. If the database being analyzed does not contain virtual logical children, the values in this column will be zeros. The last line of the report contains the values for all segments combined.

*Example 8-1 Sample DFSMAID0 report for a partition*

partition 1 :				
minimum key =				
+0000 d2c1c1f1 f1f2f3f4  KAA11234				
maximum key =				
+0000 d2f2f3f9 f9f2f3f4  K2399234				
	segments	bytes	prefix-incr	length-incr
1) 'PRODUCT '	31567	4040576	252536	0
2) 'INVENT'	103781	8094918	830248	0
3) 'ORDQTY'	171182	10955648	1369456	0
4) 'MFGSPECS'	51115	10938610	408920	0
SUM)	357645	34029752	2861160	0

### 8.2.2 Adjusting the sizes from the reports

You must add to the sizes of the partitions reported by the utility when allocating space. The values in the reports do not include space for PHDAM root anchor points (RAPs), bit maps, free space element anchor points (FSEAPs), or free space. Typically, space for RAPs, bit maps, and FSEAPs is not significant. Free space may be. In fact, you may want to increase the amount of free space in your database. With more free space, you may be able to decrease the frequency of required reorganizations or eliminate them.

We explain how to use the report with secondary indexes in 8.5.2, “Estimating the sizes of secondary index partitions” on page 107.

### 8.2.3 Control statements

SYSIN control statements are used to choose the type of analysis.



## KR control statement

KR control statements are used for key range analysis. You specify a statement for each partition except one with high values. When doing key range analysis, the utility always assumes that a partition with high values is to be included. Keys may be specified as either character or hexadecimal. Example 8-2 shows the specification of keys in both formats. This produces reports for four partitions.

### *Example 8-2 Key range control statements*

---

```
KR=C*2500000*  
KR=C*5000000*  
KR=X*F7F5F0F0F0F0F0*
```

---

## MAX control statement

A MAX control statement is used for analyzing the number of partitions required if each is limited to the number of bytes of segment data specified. Only one MAX control statement may be specified. Values up to 2147483647 may be used. Higher values are reduced to 2147483647. Example 8-3 shows the specification of a maximum size of 250,000,000 bytes. You should include a SAMPLE control statement when using MAX to analyze large databases.

The MAX statement should not be used when reading a non-HALDB secondary index. The number of bytes reported for secondary indexes is inaccurate, therefore, the MAX statement would provide misleading information.

### *Example 8-3 Maximum bytes control statement*

---

```
MAX=250000000
```

---

## NBR control statement

A NBR control statement is used for analyzing the size and high keys for a specified number of equally sized partitions. Only one NBR control statement may be specified. Values from one to 1001 may be specified. Example 8-4 shows the specification of 12 partitions. You should include a SAMPLE control statement when using NBR to analyze large databases.

### *Example 8-4 Number of partitions control statement*

---

```
NBR=12
```

---

## SAMPLE control statement

A SAMPLE control statement may be used with a MAX or NBR control statement. It improves the efficiency of the execution by analyzing a subset of the records in the database. For large databases, it may significantly reduce the time

required for execution by reducing the amount of data stored and sorted. Sample values of 20,000 are reasonable for most analyses. Example 8-5 shows the specification of a sample size of 20,000 database records. SAMPLE values up to 699049 are valid. If you specify a SAMPLE statement, it must be the first control statement.

*Example 8-5 Sample control statement*

---

```
SAMPLE=20000
```

---

The SAMPLE control statement has an optional SEED parameter. It is used to change the records that are sampled. Normally, you do not need to specify a seed. If you want to verify an analysis that used a SAMPLE statement, you may run it again with a different SEED value and compare the reports. The default for SEED is 1.

## 8.3 Migrating simple databases

Databases that have no secondary indexes or logical relationships may be migrated one at a time or in groups. The steps for the migration are:

- ▶ Unload of the existing database using HD Unload and specifying MIGRATE=YES
- ▶ DBDGEN as a HALDB database
- ▶ Delete database information from the RECONs
- ▶ Definition of the partitions
- ▶ Allocation of database data sets
- ▶ Partition initialization
- ▶ Reload as a HALDB database using HD Reload
- ▶ Image copy of the database data sets

When you migrate a database, you will probably want to keep the same database name. This avoids the need to change PSBs that reference the database. In our examples and explanations we have assumed that the same name will be used for the HALDB.

### 8.3.1 Unloading the existing database

The HD Reorganization Unload utility (DFSURGU0) is used to unload the existing HDAM or HIDAM database. The unload must use the DBD that defines the non-HALDB database. The MIGRATE=YES control statement must be

specified. With this statement, HD Unload creates a HALDB prefix for each unloaded segment in its output file. The prefix includes a generated ILK to be associated with the segment. OSAM sequential buffering should be used with OSAM database data sets. This will typically improve the performance of the unload.

### 8.3.2 Saving existing database information

Before changing the DBD or deleting information from the RECONS, we recommend that you save your DBD source and back up your RECONS. If you need to fall back after attempting a migration, you will need this information.

### 8.3.3 Changing the DBD

The DBD for your database must be changed to a HALDB database organization type. You might have to change other DBD specifications. These considerations are explained in 8.1.1, “DBD changes” on page 90.

### 8.3.4 Deleting database information from the RECONS

The database information in the RECONS must be deleted. This must be done before the HALDB database is registered. This registration occurs when the partitions are defined. The deletion may be done with a DBRC DELETE.DB command. Figure 8-6 shows the command we used to delete information about the NORDDDB database. If you are migrating a HIDAM database, you will want to delete the HIDAM index information eventually. This does not have to be done now. The index for a PHIDAM database is generated automatically and does not use the HIDAM index database name.

*Example 8-6 DBRC command to delete a database from the RECONS*

---

```
DELETE.DB DBD(NORDDDB)
```

---

After the DELETE.DB command is processed, the information in the RECONS about the non-HALDB database is deleted. You cannot use the RECONS to restore this version of the database. You may want to copy the RECONS before issuing the command. Then, if necessary, you could use the copy with a GENJCL.RECOV command and the old DBD to restore the database to its non-HALDB state.

### 8.3.5 Defining the partitions

The partitions are defined either with the Partition Definition utility (PDU) or with DBRC INIT commands. The use of the PDU is explained in Chapter 4, “Partition

Definition utility” on page 45. The use of DBRC INIT commands is explained in Chapter 5, “Batch definition of HALDB” on page 61. Before defining the partitions you will need to decide on your partition boundaries. The HALDB Migration Aid utility (DFSMAIDO) can provide useful information to help make these decisions. Its use is explained in 8.2, “HALDB Migration Aid utility (DFSMAIDO)” on page 93.

### 8.3.6 Allocating database data sets

You are responsible for allocating the database data sets used by the databases. The data set names are created from the data set name prefix that you specify when you define a partition and the data set name algorithm. This is explained in 1.9, “Naming conventions” on page 17. You must use these names when you allocate your data sets.

You may be able to use the information generated by the HALDB Migration Aid utility (DFSMAIDO) to assist you in sizing the data sets. It reports the number of bytes of segment data in each partition. You will need additional space for segments to be added, free space, and bit maps.

When migrating to HALDB you may want to increase your free space parameters. Non-HALDB databases sometimes have suboptimal free space values due to data set size limitations. Additional free space may allow you to reorganize your databases or partitions less frequently. In some cases, additional free space could eliminate the need for reorganizations.

Each PHDAM or PHIDAM partition includes an ILDS. This is required even when there are no secondary indexes or logical relationships. In this case, you can allocate a very small amount of space for the data set, such as one track.

All ILDSs have fixed-length 50-byte records. Keys are nine bytes at zero offset. Example 8-7 shows IDCAMS DEFINE statements to allocate an ILDS.

*Example 8-7 Sample DEFINE for an ILDS*

---

```

DEFINE CLUSTER(                               -
  NAME(JOUK03.HALDB.DB.PEOPLE.L00001) -
  INDEXED                                     -
  CYL(1 1)                                   -
  RECORDSIZE(50 50)                         -
  SHAREOPTIONS(3 3)                         -
  SPEED                                      -
  KEY(9,0)                                   -
  FREESPACE(10,10)                         -
  CONTROLINTERVALSIZE(8192)                -
  VOLUMES(TOTIMN)                          -

```

)

You must specify REUSE on the DEFINE for all HALDB VSAM data sets other than ILDSs.

DBDGEN output is useful when allocating PHIDAM primary indexes. The output of DBDGEN for the PHIDAM database reports required parameters for the IDCAMS define. This is shown following the label RECOMMENDED VSAM DEFINE CLUSTER PARAMETERS. The required parameters are the KEY values, RECORDSIZE values, INDEXED, and REUSE. Example 8-8 shows IDCAMS DEFINE statements to allocate a PHIDAM primary index data set.

*Example 8-8 Sample DEFINE for a PHIDAM primary index data set*

```
DEFINE CLUSTER( -  
  NAME(JOUK03.HALDB.DB.RZL.X00001) -  
  INDEXED -  
  CYL(10 5) -  
  RECORDSIZE(20 20) -  
  SHAREOPTIONS(3 3) -  
  REUSE -  
  KEY(14,5) -  
  FREESPACE(25,10) -  
  CONTROLINTERVALSIZE(8192) -  
  VOLUMES(TOTIMN) -  
)
```

Appendix A, “Sample GENJCL.USER skeletons for HALDB allocation” on page 281, explains how DBRC GENJCL.USER may be used to create jobs to allocate database data sets.

### 8.3.7 Initializing the partitions

The partitions in the HALDB database must be initialized. Either the Database Prereorganization utility or the HALDB Database Data Set Initialization utility may be used. We recommend the use of the Prereorganization utility. Partition initialization is explained in Chapter 6, “Partition initialization” on page 73.

### 8.3.8 Reloading the HALDB database

The HD Reorganization Reload utility (DFSURGL0) is used to load the HALDB database. The input to Reload is the output from the Unload utility. When there are no secondary indexes or logical relationships defined, the Reload utility does not update the ILDS. Nevertheless, an ILDS must be available for each partition.

**Note:** Message IEC161I 152-061 is issued for ILDSs because these data sets are empty when opened. This does not indicate a problem.

### 8.3.9 Image copying the database data sets

The *image copy needed* flag is set for each database data set other than ILDSs and PHIDAM primary indexes. Image copies should be taken of the flagged data sets.

## 8.4 ILDS creation options

HD Reload creates an entry in the ILDS for a partition for each segment that is the target of a secondary index or logical relationship pointer. These entries are used to heal pointers when they are first used after the load. HD Reload has three options for the creation of these ILDS entries: They may be created as the target segments are loaded; they may be created in a multithread process after all target segments have been loaded; and they may be created outside of the execution of HD Reload.

### 8.4.1 HD Reload with no control statement

When you do not specify a control statement in the SYSIN data for HD Reload, an ILDS entry is created when a target of a secondary index or logical relationship is inserted in the database. These updates could be slow for three reasons. First, there may be many CI and CA splits. The inserts in the ILDS are done in VSAM update mode. When a CI or CA is filled, it must be split by VSAM. Second, they may be random. The ILDS keys are based on the ILK, which is based on the location of the target segment in the non-HALDB. Third, this is a single thread process. These writes are not overlapped with other reload processing.

### 8.4.2 HD Reload with an ILDSMULTI control statement

The ILDSMULTI control statement was added to IMS Version 7 by APAR PQ36991 and to IMS Version 8 by PQ54227. When you specify an ILDSMULTI control statement in the SYSIN data, HD Reload uses a multi-thread process. ILDS entries are not written when target segments are loaded. Instead, they are saved in data spaces. When HD Reload has completed the load of the database, it sorts the ILDS entries by key within partition. It creates a separate process for each partition. Each process writes the ILDS entries in VSAM load mode. This process can be much more efficient than creating the entries as the targets are

loaded. First, the writes of the ILDS entries to different partitions are overlapped. Second, the writes are sequential. Third, the writes are done in load mode. This avoids the overhead of CI and CA splits. There is an additional benefit of load mode: Free space may be created. Future reorganizations could benefit from having this free space for their updates to the ILDSs.

Do not specify ILDSMULTI if the database has no logical relationships or secondary indexes defined.

The ILDSMULTI control statement may be used only during migrations. It is invalid if the input file was not created by an HD Unload execution using either MIGRATE=YES or MIGRATX=YES.

### 8.4.3 HD Reload with a NOILDS control statement

The NOILDS control statement was added to IMS Version 7 by APAR PQ36991 and to IMS Version 8 by PQ54227. When you specify a NOILDS control statement in the SYSIN data, HD Reload does not create entries in the ILDSs. They must be created by a separate process. The HALDB Index/ILDS Rebuild utility (DFSPREC0) is used to do this. Separate executions of DFSPREC0 are required for each partition. These executions may be done in parallel and on different machines. Obviously, the use of NOILDS improves the performance of HD Reload since it does not have to write the ILDS entries. On the other hand, DFSPREC0 must read the partitions to create the ILDSs.

The NOILDS control statement is not limited to migrations. It may be used for any execution of HD Reload.

## 8.5 Migrating databases with secondary indexes

IMS does not support HALDB secondary indexes into non-HALDB databases or non-HALDB secondary indexes into HALDB databases. When you migrate a database that has secondary indexes, you must also migrate any secondary indexes that it has.

There are two standard ways to migrate databases with secondary indexes. First, you can migrate the database and create the HALDB secondary indexes from the migrated source segments. This method uses the MIGRATX=YES control statement with HD Unload. We recommend this method. Second, you can migrate the database and migrate the secondary indexes. This technique requires that each secondary index be unloaded. The MIGRATE=YES control statement is used for the unload of the indexed database and the unload of each secondary index. This method has two disadvantages. Each secondary index must be read. When reading the secondary index, the indexed database also

must be read. These extra reads can substantially elongate the migration process. For this reason, we recommend that you not use this method. We only explain the use of the first method.

There is an alternative to the two standard migration methods. This requires the use of the IMS Index Builder tool. This tool can build your secondary indexes from the indexed database. You do not have to migrate the secondary indexes when you use Index Builder. Instead, you can migrate the indexed database and then use Index Builder to create the secondary indexes.

### 8.5.1 Changing DBDs for secondary indexes

You must change the DBDs for your secondary index databases before they are loaded as HALDB secondary indexes. You may also need to change the DBDs of some of your indexed databases. We explain all changes required for the migration to HALDB in 8.1.1, “DBD changes” on page 90. Specific changes for secondary indexes are explained here.

PSINDEX must be specified on the ACCESS parameter of the DBD statement for the secondary index.

RKSIZE must be specified on the LCHILD statement of the secondary index.

#### /SX subsequence fields

If you use a /SX field to create unique keys for a secondary index, you must adjust your secondary index DBD. In non-HALDB secondary indexes a /SX field contains a four-byte RBA. In a HALDB secondary index a /SX field contains a eight-byte ILK. The larger size of the field requires two changes.

- ▶ The BYTES parameter on the SEGM statement in the secondary index DBD must be increased by four.
- ▶ The BYTES parameter on the FIELD statement for the sequence field in the secondary index DBD must be increased by four.

Changes are not required to the XDFLD statement or the /SX FIELD statement in the indexed DBD. The BYTES parameter on the /SX FIELD statement is not required and is ignored if it is specified.

Example 8-9 shows the DBD for an indexed HDAM database with a /SX field defined.

*Example 8-9 HDAM DBD with /SX field*

---

DBD	NAME=VEHICLE,ACCESS=(HDAM,OSAM),	X
	RMNAME=(DFSHDC40,2,500,)	
	DATASET DD1=VEHICLE1,BLOCK=1648,SCAN=0	



```

SEGM  NAME=AUTO,BYTES=54,PTR=TB
FIELD NAME=(ID,SEQ,U),BYTES=10,START=1
FIELD NAME=MAKE,BYTES=20,START=11
FIELD NAME=MODEL,BYTES=20,START=31
FIELD NAME=YEAR,BYTES=4,START=51
FIELD NAME=/SX1
LCHILD NAME=(MAKEMOD,VEHSI),PTR=INDX
XDFLD NAME=MMIDX,SRCH=(MAKE,MODEL),SUBSEQ=/SX1
DBDGEN
FINISH
END

```

Example 8-10 shows the DBD for a non-HALDB secondary index that uses the /SX field defined in Example 8-9 on page 102.

Example 8-10 Non-HALDB secondary index DBD using a /SX field

```

DBD    NAME=VEHSI,ACCESS=INDEX
DATASET DD1=VEHSI1,DEVICE=3390,SIZE=8192
SEGM  NAME=MAKEMOD,BYTES=44,PARENT=0
FIELD NAME=(NAMES,SEQ,U),BYTES=44,START=1
LCHILD NAME=(AUTO,VEHICLE),INDEX=MMIDX
DBDGEN
FINISH
END

```

Example 8-11 shows the DBD from Example 8-9 on page 102 after it has been converted to HALDB. The only changes are those made for all HDAM DBDs when they are converted to PHDAM. That is, the ACCESS parameter on the DBD STATEMENT is changed to PHDAM and the DATASET statement is deleted.

Example 8-11 PHDAM DBD with /SX field

```

DBD    NAME=VEHICLE,ACCESS=(PHDAM,OSAM),
RMNAME=(DFSHDC40,2,500,)
SEGM  NAME=AUTO,BYTES=54,PTR=TB
FIELD NAME=(ID,SEQ,U),BYTES=10,START=1
FIELD NAME=MAKE,BYTES=20,START=11
FIELD NAME=MODEL,BYTES=20,START=31
FIELD NAME=YEAR,BYTES=4,START=51
FIELD NAME=/SX1
LCHILD NAME=(MAKEMOD,VEHSI),PTR=INDX
XDFLD NAME=MMIDX,SRCH=(MAKE,MODEL),SUBSEQ=/SX1
DBDGEN
FINISH

```

END

---

Example 8-12 shows the DBD from Example 8-10 on page 103 after it has been converted to HALDB. As with other secondary index DBD conversions, the ACCESS parameter on the DBD statement is changed to PSINDEX, the DATASET statement is deleted, and the RKSIZE parameter is added to the LCHILD statement. Since a /SX field is used as a subsequence field, the BYTES parameters on the SEGM and FIELD statements are incremented by four.

*Example 8-12 HALDB secondary index DBD using a /SX field*

---

```
DBD    NAME=VEHSI,ACCESS=PSINDEX
SEGM   NAME=MAKEMOD,BYTES=48,PARENT=0
FIELD  NAME=(NAMES,SEQ,U),BYTES=48,START=1
LCHILD NAME=(AUTO,VEHICLE),INDEX=MMIDX,RKSIZE=10
DBDGEN
FINISH
END
```

---

## Symbolic pointers

You cannot have symbolic pointers in a HALDB secondary index. If you currently have symbolic pointers, you must eliminate them. Symbolic pointers are defined with a PTR=SYMB parameter on the LCHILD statements in the non-HALDB secondary index and the indexed database DBDs. You must change the PTR=SYMB specification in the indexed database DBD to PTR=INDX and either omit the PTR parameter in the secondary index DBD or specify PTR=SNGL in it. An illustration of these changes is shown in the following examples.

Example 8-13 shows the non-HALDB secondary index DBD with symbolic pointing defined. Example 8-14 on page 105 shows the indexed HDAM DBD. Example 8-15 on page 105 shows the secondary index DBD after it has been converted to HALDB. PTR=SYMB is deleted from the LCHILD statement. RKSIZE is added to it. Symbolic pointers are stored in the data area of index segments. Since it is not in the HALDB secondary index, the BYTES parameter on the SEGM segment is reduced by the size of the symbolic pointer. Example 8-16 on page 105 shows the indexed database DBD after it has been converted to HALDB. PTR=INDX is specified on the LCHILD segment for the secondary index relationship.

*Example 8-13 DBD for secondary index using symbolic pointing*

---

```
DBD    NAME=CONTRSI,ACCESS=INDEX
DATASET DD1=CONTSI,DEVICE=3390,SIZE=8192
SEGM   NAME=CONTR,BYTES=26,PARENT=0
FIELD  NAME=(CONTRNUM,SEQ,U),BYTES=8,START=1,TYPE=C
LCHILD NAME=(CONTRACT,ENGAGEM),INDEX=CONTRIDX,PTR=SYMB
DBDGEN
```

```
FINISH
END
```

---

*Example 8-14 DBD for indexed database with symbolic pointing*

---

```
DBD    NAME=ENGAGEM,ACCESS=HDAM,RMNAME=(DFSHDC40,1,500,824)
DATASET DD1=ENGAMDAM,BLOCK=1648,SCAN=0
SEGM   NAME=CLIENT,BYTES=100,PTR=TWIN
FIELD  NAME=(CLNUM,SEQ,U),BYTES=10,START=1,TYPE=C
SEGM   NAME=CONTRACT,PARENT=CLIENT,BYTES=60,PTR=TWIN
FIELD  NAME=(CONTRNO,SEQ,U),BYTES=8,START=1,TYPE=C
LCHILD NAME=(CONTR,CONTRSI),PTR=SYMB
XDFLD  NAME=CONTRIDX,SRCH=CONTRNO
DBDGEN
FINISH
END
```

---

*Example 8-15 DBD for HALDB secondary index*

---

```
DBD    NAME=CONTRSI,ACCESS=PSINDEX
SEGM   NAME=CONTR,BYTES=8,PARENT=0
FIELD  NAME=(CONTRNUM,SEQ,U),BYTES=8,START=1,TYPE=C
LCHILD NAME=(CONTRACT,ENGAGEM),INDEX=CONTRIDX,RKSIZE=10
DBDGEN
FINISH
END
```

---

*Example 8-16 DBD for HALDB indexed database*

---

```
DBD    NAME=ENGAGEM,ACCESS=PHDAM,RMNAME=(DFSHDC40,1,500,824)
SEGM   NAME=CLIENT,BYTES=100,PTR=TWIN
FIELD  NAME=(CLNUM,SEQ,U),BYTES=10,START=1,TYPE=C
SEGM   NAME=CONTRACT,PARENT=CLIENT,BYTES=60,PTR=TWIN
FIELD  NAME=(CONTRNO,SEQ,U),BYTES=8,START=1,TYPE=C
LCHILD NAME=(CONTR,CONTRSI),PTR=INDX
XDFLD  NAME=CONTRIDX,SRCH=CONTRNO
DBDGEN
FINISH
END
```

---

If you have applications that process a secondary index using symbolic pointers as a database, you will want to make other changes to your DBDs. We explain these changes in 11.5.2, “Secondary indexes with symbolic pointers” on page 202.

## Non-unique keys

HALDB secondary indexes must have unique keys. Non-HALDB secondary indexes may have non-unique keys. Non-unique keys are specified with an M as the third subparameter of the NAME parameter for the key field in a secondary index DBD. Non-unique keys require an overflow data set to hold duplicate keys. This is specified in the OVFLW parameter of the DBD DATASET statement. If your non-HALDB secondary index DBD has these definitions, they must be changed as part of the conversion. Example 8-17 shows a non-HALDB secondary index DBD with non-unique keys. Example 8-18 shows the DBD for the secondary index after it has been converted to HALDB. Since the index key must be unique, you will probably need to add a subsequence field. Typically, you would do this by adding a /SX system-related field to the XDFLD statement in the indexed database DBD. This would add eight bytes to the length of the sequence field in the index. In the examples, the BYTES parameter in the FIELD statement is increased by eight for this conversion.

*Example 8-17 Non-HALDB secondary index DBD with non-unique keys*

---

```
DBD      NAME=XSI3,ACCESS=INDEX
DATASET  DD1=XSI301,OVFLW=XSI302
SEGM     NAME=XSNAM,BYTES=6,PARENT=0
FIELD    NAME=(XSNAME,SEQ,M),START=1,BYTES=6
LCHILD   NAME=(PERF,XPER01),INDEX=NAMX1,POINTER=SNGL
DBDGEN
FINISH
END
```

---

*Example 8-18 HALDB secondary index DBD with unique keys*

---

```
DBD      NAME=XSI3,ACCESS=PSINDEX
SEGM     NAME=XSNAM,BYTES=14,PARENT=0
FIELD    NAME=(XSNAME,SEQ,U),START=1,BYTES=14
LCHILD   NAME=(PERF,XPER01),INDEX=NAMX1,POINTER=SNGL,RKSIZE=12
DBDGEN
FINISH
END
```

---

## Shared secondary indexes

You cannot have a shared secondary index with HALDB. Shared indexes are defined with a CONST parameter on the XDFLD statement of the indexed database. You must delete this parameter when you migrate to the database.

## 8.5.2 Estimating the sizes of secondary index partitions

HALDB secondary index records are often much larger than non-HALDB secondary index records. There are two reasons for this. First, the pointers are different. Each HALDB secondary index record includes the 28-byte extended pointer set (EPS). Non-HALDB secondary index records have either a 4-byte direct pointer or a symbolic pointer. Symbolic pointers are the length of the concatenated key. Second, the root key of the target segment is stored in the HALDB secondary index record. This is not done with non-HALDB secondary index records. When allocating your HALDB secondary index partitions, you must account for this increased size.

You must define partition boundaries for secondary indexes. The HALDB Migration Aid (DFSMAID0) utility may be used to report on secondary indexes.

The total size of the secondary index is usually easy to estimate. All secondary index entries are fixed length. The DBDGEN reports this length. It is the RECORDSIZE in the RECOMMENDED VSAM DEFINE CLUSTER PARAMETERS of the HALDB DBDGEN output listing. The number of entries does not change during the migration. You can find the current number of entries from the output of several utilities. When you recreate a secondary index as part of a reorganization process, the HISAM Reload utility or any tool that you use reports this information. Alternatively, you can use the REC-TOTAL value from LISTCAT to find the number of records in the existing secondary index data set. Finally, you may use the HALDB Migration Aid utility to find the number of records. The number of records, their length, and free space requirements may be used to estimate the size of an entire HALDB secondary index. For example, if you had 1,000,000 entries, a record length of 48 bytes, and wanted an additional 25 percent for free space, the index would require 60,000,000 bytes.

The size requirements of each partition will depend on the number of index entries in the partition. You may estimate this from the output of the HALDB Migration Aid utility (DFSMAID0) or from other means. If you already know the number of secondary index entries in a key range you do not need to use the Migration Aid utility. Otherwise, you should use the utility.

The Migration Aid utility provides accurate information about the number of records in a key range and key range boundaries. It does not provide accurate information about the number of bytes required for a segment or partition. For this reason, you should not use the MAX control statement when reading a secondary index. If you know the number of partitions you want, use the NBR control statement. If you know the high keys that you want, use KR control statements. When an NBR statement is used, the high keys that are reported for each partition are accurate. When using KR control statements, the number of segments reported for each partition is accurate.

Example 8-19 on page 108 shows some of the output of the Migration Aid utility for a secondary index. This was generated with a NBR=4 control statement. We have omitted the reports for partitions 1, 2, and 3. Only the report for partition 4 and the total database are shown. The numbers of segments reported in the segments column are correct. Partition 4 will contain 158,518 secondary index segments and the entire index will contain 634,078 segments. The information in the bytes and prefix-incr columns is wrong. Nevertheless, all we need is the number of segments in a partition to calculate its space requirements.

Example 8-19 DFSMAID0 output for a secondary index

partition 4 :

minimum key =

+0000 f0f0f1f6 f0f8f1f1 f5f0f0f0 f0f0f1f3

0016081150000013

+0010 f1f7

17

maximum key =

+0000 f0f0f2f1 f1f0f3f0 f0f0f0f0 f0f0f0f5

0021103000000005

+0010 f4f6

46

	segments	bytes	prefix-incr	length-incr
1) 'ORDRCUST'	158518	3804432	1268144	0
SUM)	158518	3804432	1268144	0

-----

sum of partitions:

segments

bytes

prefix-incr

length-incr

1) 'ORDRCUST'	634078	15217872	5072624	0
SUM)	634078	15217872	5072624	0

-----

You may use the record size reported by DBDGEN and the number of segments for a partition to estimate the size requirement for the partition. Of course, you can add free space and room for expansion of the partition.

**Restriction:** DFSMAID0 reports of bytes and prefix-incr for secondary indexes should never be used. They are inaccurate. DFSMAID0 may be used to find high keys and the number of secondary index entries for each partition. These are accurate.

### 8.5.3 Unloading databases with secondary indexes

When you unload a database with secondary indexes for migration to HALDB, you can create unload files for its secondary indexes. This is done when you specify the `MIGRATX=YES` control statement with `HD Unload`. This technique does not read the secondary indexes. DD statements for the secondary index data sets are not required. The information to create secondary index entries is generated from the source segments. The unload files for secondary indexes must be sorted before they are used to load the secondary indexes. The output of `HD Unload` includes sort control statements to be used for the sorts. Figure 8-1 illustrates a use of `MIGRATX=YES` to migrate a database and its two secondary indexes.

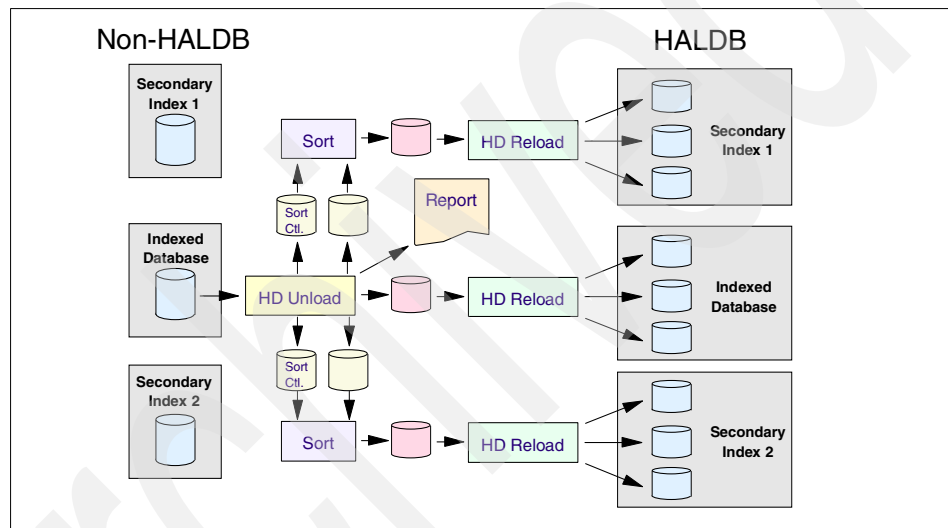


Figure 8-1 Using `MIGRATX=YES` to migrate a database with secondary indexes

OSAM sequential buffering should be used with OSAM database data sets. This will typically improve the performance of the unload.

The `MIGRATX=YES` technique does not preserve any user data in the secondary indexes. This is rarely a problem because most installations do not have user data. They avoid user data because it is lost when the reorganization of a non-HALDB database requires the rebuilding of its secondary indexes. These rebuilds do not preserve user data.

The `SYSPRINT` listing of `HD Unload` includes a Work File Statistics report. It tells you which output files are to be used for each secondary index. This is not based on secondary index names. You must use the report to understand this relationship. The utility assigns `DFSWRK01` to the first secondary index

relationship it encounters in the unload. The report also lists the number of segments in the secondary index files and the sort field sizes and offsets. A sample report is shown in Example 8-20. In it DFSWRK01 contains the records for secondary index STUNUM, and DFSWRK02 contains the records for secondary index STUCRT.

*Example 8-20 Work file statistics report with MIGRATX=YES*

W O R K F I L E S T A T I S T I C S					
SINAME	WFNAME	SFNAME	RCDTOTAL	OFFSET	LENGTH
STUNUM	DFSWRK01	DFSSRT01	000087012	0124	0006
STUCRT	DFSWRK02	DFSSRT02	000010702	0124	0010

You must sort each secondary index output file. Use the sort control statements in the DFSSRTnn files for these sorts. Example 8-21 shows sample JCL for the sorting of the STUNUM secondary index. The SORTIN DD specifies the DFSWRK01 output from HD Unload. The SYSIN DD specifies the DFSSRT01 output from HD Unload. The output of this sort is input to HD Reload for the STUNUM secondary index.

*Example 8-21 Sample JCL to sort a secondary index output file*

```
//*****  
//*   SORT THE STUNUM SEC. INDEX RECORDS FROM MIGRATX=YES  
//*****  
//*  
//SORT01  EXEC PGM=SORT,REGION=2048K,PARM='CORE=MAX'  
//SORTLIB DD DSN=SYS1.SORTLIB,DISP=SHR  
//SYSOUT  DD SYSOUT=*  
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,(50,5),,CONTIG)  
//SORTWK02 DD UNIT=SYSDA,SPACE=(CYL,(50,5),,CONTIG)  
//SORTWK03 DD UNIT=SYSDA,SPACE=(CYL,(50,5),,CONTIG)  
//SYSIN   DD DSN=JOUK03.STUDENT.MIGR.SRT01,DISP=OLD  
//SORTIN  DD DSN=JOUK03.STUDENT.MIGR.WRK01,DISP=OLD  
//SORTOUT DD DSN=JOUK03.STUDENT.MIGR.WRK01.SORTED,DISP=(NEW,CATLG),  
//        UNIT=3390,VOL=SER=TOTIMN,  
//        SPACE=(CYL,(50,10),RLSE)
```

**The IMS Index Builder alternative**

If you use the IMS Index Builder, you have the option of not migrating your secondary indexes. Instead, you can migrate just your indexed database and use Index Builder to create the secondary indexes. Since you do not need the unload files for the secondary indexes, do not use MIGRATX=YES. Instead, use



MIGRATE=YES. This is the technique that is used when there are no secondary indexes. It is explained in 8.3.1, “Unloading the existing database” on page 96.

## 8.5.4 Allocating secondary index data sets

DBDGEN output is useful when allocating secondary index data sets. The output of DBDGEN for the secondary index reports required parameters for the IDCAMS define. This is shown following the label RECOMMENDED VSAM DEFINE CLUSTER PARAMETERS. The report includes misleading text that states \*NOTE1 - THIS IS THE VSAM DEFINITION OF A HALDB PRIMARY INDEX. Actually, it is the VSAM definition of a HALDB secondary index. The required parameters are the KEY values, RECORDSIZE values, INDEXED, and REUSE. Example 8-22 shows IDCAMS DEFINE statements to allocate a secondary index data set.

*Example 8-22 Sample DEFINE for a secondary index data set*

---

```
DEFINE CLUSTER(                                -
  NAME(JOUK03.HALDB.DB.PE0SKSI.A00001) -
  INDEXED                                     -
  RECORDSIZE(86 86)                           -
  CYL(20 5)                                    -
  SHAREOPTIONS(3 3)                           -
  REUSE                                       -
  KEY(29,50)                                   -
  FREESPACE(10,10)                             -
  CONTROLINTERVALSIZE(4096)                     -
  VOLUMES(TOTIMN)                               -
)
```

---

## 8.5.5 Loading databases and their secondary indexes

As with other databases, HD Reload is used to load databases with secondary indexes and their secondary indexes. When a segment that is the target of a secondary index is loaded, an entry is created in the ILDS for its partition. Otherwise, there are no special considerations for loading indexed databases.

The sorted output files from HD Unload are used as input to HD Reload when creating a secondary index as part of the migration. Before loading a secondary index, you must modify its DBD, define its partitions, allocate its data sets, and initialize its partitions. These are the same steps required for migrating any other databases.

HD Reload has options for creating ILDS entries. We explain these options in 8.4, “ILDS creation options” on page 100.

## The IMS Index Builder alternative

If you use the IMS Index Builder, you do not use HD Reload to create the secondary indexes. Instead, Index Builder reads the migrated indexed database, and creates the secondary indexes.

### Converting to /SX fields for uniqueness

If you add a /SX field to create uniqueness you need to add three steps before you reload the secondary index. These steps are required to place the unload records in the correct subsequence order. The steps are executed in place of the sort explained in 8.5.3, “Unloading databases with secondary indexes” on page 109. The steps are executed before the execution of HD Reload for the secondary index. These steps are:

1. Split the file into three separate files containing the header record, the unload records, and the trailer record. Example 8-23 shows sort JCL and control statements for this step.

*Example 8-23 Step to split the unload records*

---

```
//SORTIN DD DSN=UNLOAD.OUTPUT,DISP=SHR
//HEADER DD DSN=HEADER.FILE,DISP=(NEW,PASS)
//TRAILER DD DSN=TRAILER.FILE,DISP=(NEW,PASS)
//ULCOPY DD DSN=UNLOAD.COPY,DISP=(NEW,PASS)
//SYSIN DD *
OPTION COPY
OUTFIL INCLUDE=(5,2,CH,EQ,X'0080'),FNAMES=HEADER
OUTFIL INCLUDE=(5,2,CH,EQ,X'0090'),FNAMES=TRAILER
OUTFIL SAVE,FNAMES=ULCOPY
RECORD TYPE=V
```

---

2. Sort the file containing unload records in the /SX subsequence order. You must calculate the offset and the size of the sort field. The offset is 63 bytes plus the size of the root key of the indexed database. The sort field size is the size of the secondary index search field plus eight bytes. Example 8-24 shows sort JCL and control statements for this step. In this example, the offset is 73 bytes and the sort field size is 18 bytes.

*Example 8-24 Step to sort the unload records*

---

```
//SORTIN DD DSN=UNLOAD.COPY,DISP=SHR
//SORTOUT DD DSN=UNLOAD.SORTED1,DISP=(,CATLG),
// UNIT=SYSDA,VOL=SER=000000,SPACE=(CYL,(1,5))
//SYSIN DD *
SORT FIELDS=(73,18,CH,A),FILSZ=E1000
RECORD TYPE=V
END
```

---

3. Merge the records into one file. Example 8-25 shows sort JCL and control statements for this step.

*Example 8-25 Step to merge the unload records*

---

```
//SORTIN DD DSN=UNLOAD.HEADER,DISP=(OLD,DELETE),
//          UNIT=SYSDA,VOL=SER=000000,SPACE=(CYL,(1,5))
//          DD DSN=UNLOAD.SORTED1,DISP=(OLD,DELETE),
//          UNIT=SYSDA,VOL=SER=000000,SPACE=(CYL,(1,5))
//          DD DSN=UNLOAD.TRAILER,DISP=(OLD,DELETE),
//          UNIT=SYSDA,VOL=SER=000000,SPACE=(CYL,(1,5))
//SORTOUT DD DSN=UNLOAD.SORTED2,DISP=(NEW,KEEP),
//          UNIT=SYSDA,VOL=SER=000000,SPACE=(CYL,(1,5))
//SYSOUT DD SYSOUT=A
//SYSIN DD *
OPTION COPY
END
```

---

## 8.5.6 Secondary index pointers after the migration

After the migration reload the secondary index pointers are not healed. That is, the direct pointers cannot be used. When the pointers are needed, the ILDS of the target partition must be used to find the target segment.

## 8.5.7 Using MIGRATE=YES with secondary indexes

We highly recommend that you migrate databases with secondary indexes by using the MIGRATX=YES control statement with HD Unload. The MIGRATE=YES control statement is valid, but should not be used.

When an HD Unload MIGRATE=YES control statement is used to unload a database with secondary indexes, output files are not created for the secondary indexes. The secondary indexes must be unloaded with separate executions of HD Unload. When a secondary index is unloaded, the indexed database must also be read. Each secondary index record requires a read of the indexed database. These reads are random. This can elongate the unload process significantly. Figure 8-2 on page 114 illustrates the use of MIGRATE=YES to migrate a database and its two secondary indexes. There are three executions of HD Unload. Each execution must read the indexed database.

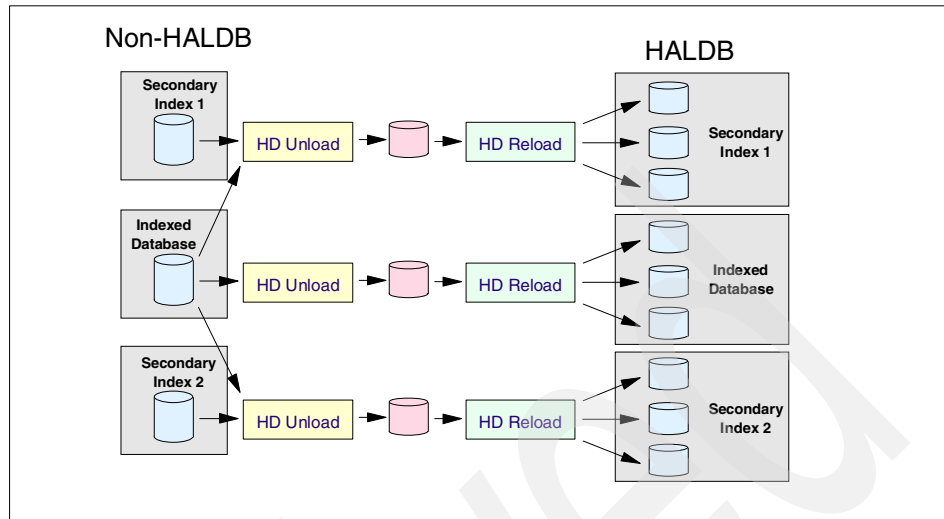


Figure 8-2 Using MIGRATE=YES to migrate a database with secondary indexes

## 8.6 Migrating databases with logical relationships

IMS does not support logical relationships between HALDB and non-HALDB databases. Databases that are logically related to each other must be migrated together. This section describes the special considerations for migrating logically related databases. If a database also has secondary indexes, the considerations discussed in “Migrating databases with secondary indexes” on page 101 must also be observed.

### 8.6.1 Changing DBDs with logical relationships

There are three types of logical relationships for non-HALDB databases. They are:

- ▶ Unidirectional
- ▶ Bidirectional with physical pairing
- ▶ Bidirectional with virtual pairing

HALDB supports unidirectional and bidirectional with physical pairing. It does not support virtual pairing. The following types of migrations of logical relationships are supported:

- ▶ Unidirectional to unidirectional
- ▶ Bidirectional with physical pairing to bidirectional with physical pairing
- ▶ Bidirectional with virtual pairing to bidirectional with physical pairing

## 8.6.2 Changing DBDs with logical relationships

The DBDs for databases with logical relationships may have to be modified when converting to HALDB. There are two reasons for this. First, some pointer options are not supported with HALDB. Second, HALDB does not support virtual pairing with bidirectional logical relationships.

The DBDs for logical databases are not changed when the physical databases they reference are migrated. DBDs for logical databases have ACCESS=LOGICAL in their DBD statements.

### Changing pointer options for logical relationships

Logically related databases may require changes to pointers in addition to those required for databases that do not have logical relationships.

HALDB does not use symbolic pointers. Any logical relationship implemented with symbolic pointing must be changed. This is done by adding the LPARNT keyword to the PTR parameter on the SEGM statement for the logical child. Even though symbolic pointing is not used, the concatenated key of the logical parent is stored in the logical child. This is done no matter what keywords you use in the PARENT parameter. That is, PHYSICAL or P will be used even if you specify VIRTUAL or V. To avoid the warning message that is issued when VIRTUAL or V is specified, you should specify the PHYSICAL or P keyword.

### Recognizing virtual pairing

You may determine if you have virtual pairing by examining your DBDs. If your HDAM or HIDAM database contains a SEGM statement with a SOURCE parameter, this segment is a virtual logical child. The first subparameter of SOURCE identifies the real logical child. The third subparameter identifies the database in which the real logical child resides. In Example 8-26 the NAMESKIL segment in the DBD contains a SOURCE parameter. It is a virtual logical child. The source parameter references the SKILNAME segment in the SKILLINV database in Example 8-27 on page 116. SKILNAME is a real logical child.

*Example 8-26 HIDAM DBD with a logical relationship*

---

```
DBD    NAME=PAYROLDB,ACCESS=HIDAM
DATASET DD1=PAYHIDAM,BLOCK=1648,SCAN=3
SEGM   NAME=NAMEMAST,PTR=TWINBWD,RULES=(VVV),          X
        BYTES=150
LCHILD NAME=(INDEX,INDEXDB),PTR=INDX
LCHILD NAME=(SKILNAME,SKILLINV),PAIR=NAMESKIL,PTR=DOUBLE
FIELD  NAME=(EMPLOYEE,SEQ,U),BYTES=60,START=1,TYPE=C
SEGM   NAME=NAMESKIL,PARENT=NAMEMAST,PTR=PAIRED,        X
        SOURCE=((SKILNAME,DATA,SKILLINV))
FIELD  NAME=(TYPE,SEQ,U),BYTES=21,START=1,TYPE=C
```

```
DBDGEN
FINISH
END
```

---

*Example 8-27 HDAM DBD with a logical relationship*

---

```
DBD    NAME=SKILLINV, ACCESS=HDAM, RMNAME=(DFSHDC40, 1, 500, 824)
DATASET DD1=SKILHDAM, BLOCK=1648, SCAN=0
SEGM   NAME=SKILMAST, BYTES=31, PTR=TWINBWD
FIELD  NAME=(TYPE, SEQ, U), BYTES=21, START=1, TYPE=C
SEGM   NAME=SKILNAME,                                     X
       PARENT=((SKILMAST, DBLE), (NAMEMAST, P, PAYROLDB)), X
       BYTES=80, PTR=(LPARNT, LTWINBWD, TWIN)
FIELD  NAME=(EMPLOYEE, SEQ, U), START=1, BYTES=60, TYPE=C
DBDGEN
FINISH
END
```

---

## Making changes to the DBDs for physical pairing

To convert a virtually paired logical relationship to physical pairing, the following must be done:

- ▶ Add an LCHILD statement under the parent of the segment that was the real logical child. In Example 8-26 on page 115 this is the SKILMAST segment. In this LCHILD statement do the following:
  - The new LCHILD statement must have a NAME parameter that specifies the segment that had the SOURCE parameter. In the example this is NAMESKIL.
  - The PAIR parameter must specify the segment that was specified in the SOURCE parameter. This was the real logical child. In the example this is SKILNAME.
- ▶ Find the LCHILD statement that references the segment that was the real logical child. In Example 8-26 on page 115 this is the LCHILD statement in the PAYROLDB database that includes NAME=(SKILNAME, SKILLINV). In this LCHILD statement do the following:
  - a. Delete the PTR parameter.
  - b. If there is a RULES parameter, delete it.
- ▶ Find the SEGM statement that defined the virtual logical child. In Example 8-26 on page 115 this is the NAMESKIL segment. In this SEGM statement make the following changes:
  - a. Add a BYTES parameter. The lengths of the fixed intersection data in the paired logical children must be the same. The BYTES parameter includes both the fixed intersection data and the logical parent's concatenated key

(LPCK). The BYTES value for this segment may be calculated by taking the BYTES value from the paired logical child's SEGM statement, adding the LPCK size for this segment, and subtracting the LPCK size for the paired logical child. In the example, the paired logical child for the NAMESKIL segment is the SKILNAME segment. Its size is 80 bytes. The LPCK for the NAMESKIL segment is the TYPE field in the SKILMAST segment. Its size is 21 bytes. The LPCK for the SKILNAME segment is the EMPLOYEE field in the NAMEMAST segment. Its size is 60 bytes. The BYTES parameter for the NAMESKIL segment should be 41, which is 80 + 21 - 60.

- b. Delete the SOURCE parameter.
  - c. Change the PARENT parameter so that it also references its logical parent. In the example this is the SKILMAST segment. The PARENT parameter should also include the PHYSICAL or P keyword.
  - d. Change the PTR parameter specifications. Include PAIRED. Include TWIN, T, TWINBWD, TB, NOTWIN, or NT. Do not use NOTWIN or NT unless you will have a maximum of only one instance of this segment type under a parent.
- Find the SEGM statement for segment that was the real logical child. In Example 8-26 on page 115 this is the SKILNAME segment. In this SEGM statement make the following changes:
- Change the PARENT parameter so that neither SNGL nor DBLE is specified. If VIRTUAL or V was specified, change this to PHYSICAL or P.
  - Change the PTR parameter specifications. Delete LTWIN, LT, LTWINBWD, or LTB keywords if they exist. Specify both the LPARNT and PAIRED keywords.

Example 8-28 and Example 8-29 on page 118 show the DBDs from Example 8-26 on page 115 and Example 8-27 on page 116 after they have been converted to HALDBs with physical pairing. In addition to the changes made for logical relationships, other changes for the conversion to HALDB have been made. These are changes in the ACCESS parameter on the DBD statements and deletion of the DATASET statements.

*Example 8-28 PHIDAM DBD with a logical relationship*

DBD	NAME=PAYROLDB,ACCESS=PHIDAM	
SEGM	NAME=NAMEMAST,PTR=TWINBWD,RULES=(VVV),	X
	BYTES=150	
LCHILD	NAME=(SKILNAME,SKILLINV),PAIR=NAMESKIL	
FIELD	NAME=(EMPLOYEE,SEQ,U),BYTES=60,START=1,TYPE=C	
SEGM	NAME=NAMESKIL,	X
	PARENT=((NAMEMAST),(SKILMAST,P,SKILLINV)),	X
	BYTES=41,PTR=(TWIN,PAIRED)	

```

FIELD NAME=(TYPE,SEQ,U),BYTES=21,START=1,TYPE=C
DBDGEN
FINISH
END

```

---

*Example 8-29 PHDAM DBD with a logical relationship*

---

```

DBD  NAME=SKILLINV,ACCESS=PHDAM,RMNAME=(DFSHDC40,1,500,824)
SEGM NAME=SKILMAST,BYTES=31,PTR=TWINBWD
FIELD NAME=(TYPE,SEQ,U),BYTES=21,START=1,TYPE=C
LCHILD NAME=(NAMESKIL,PAYROLDB),PAIR=SKILNAME
SEGM NAME=SKILNAME, X
      PARENT=((SKILMAST),(NAMEMAST,P,PAYROLDB)), X
      BYTES=80,PTR=(TWIN,PAIRED)
FIELD NAME=(EMPLOYEE,SEQ,U),START=1,BYTES=60,TYPE=C
DBDGEN
FINISH
END

```

---

### 8.6.3 Unloading logically related databases for migration

When a migration unload of a database with a logical child is executed, its logically related database may also be read. This is done when any of the following conditions exists:

- ▶ Physical pairing is used.
- ▶ The VIRTUAL option is specified (the logical parent's concatenated key is not stored in the logical child).
- ▶ Symbolic pointing is used.
- ▶ The database being unloaded contains the virtual logical child.

Stated another way, there are only two cases where the logically related database is not read. They are:

- ▶ The database being unloaded contains the real logical child of a virtually paired relationship that uses direct pointing, and the logical parent's concatenated key is stored in this logical child.
- ▶ The logical relationship is unidirectional using direct pointing, and the logical parent's concatenated key is stored in the logical child.

When reading the logically related database, either the logical parent or the paired logical child of each logical child is read. That is, each instance of a logical relationship requires a read of the related data. These reads are random. Each read may require a physical I/O. This can greatly increase the elapsed time of the unload.



Buffer pools must be provided for any logically related database that is read by the unload. The DFSVSAMP DD statement for HD Database Unload (DFSURGU0) must include definitions for these buffers.

Figure 8-3 illustrates a migration of two logically related databases to HALDBs.

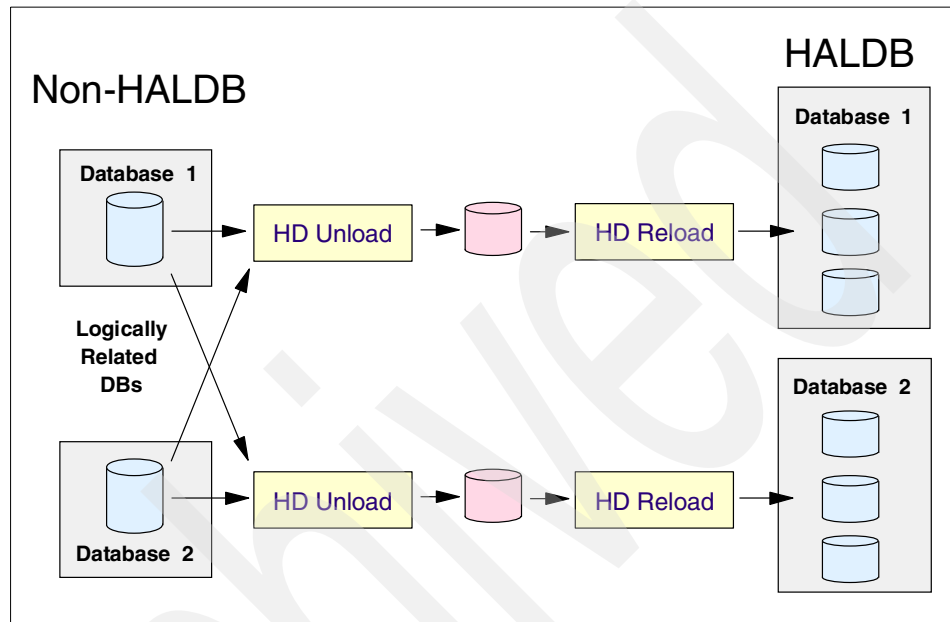


Figure 8-3 Migrating databases with logical relationships

### 8.6.4 Loading logically related databases for migration

As with other databases, HD Reload is used to load databases with logical relationships during the migration. When a logical parent of a unidirectional relationship is loaded or a logical child of a bidirectional relationship is loaded, an entry is created in the ILDS for its partition. Otherwise, there are no special considerations for loading logically related databases during a migration.

HD Reload has options for creating ILDS entries. We explain these options in 8.4, “ILDS creation options” on page 100.

### 8.6.5 Logical relationship pointers after the migration

After the migration reload the logical relationship pointers are not healed. That is, the direct pointers cannot be used. When the pointers are needed, the ILDS of the target partition must be used to find the logical parent or paired logical child.

## 8.7 Migrating from PDB or PDF

Some installations have used the IMS/ESA Partition Support Product (PDB) or similar capabilities in the Partitioned Database Facility (PDF) product from Neon Enterprise Software, Inc. to partition databases before HALDB was available. These databases may be migrated to HALDB. The same migration techniques may be used with either PDB or PDF.

### 8.7.1 Partition selection

PDB partition selection depends on the type of database organization. HIDAM, HISAM, and secondary indexes are partitioned by key range. The HDAM randomization routine selects the partition for HDAM. The randomizer can use either of two techniques. First, it can assign a range of RAPs to a partition. Second, it can have two stages. In the first stage a partition is selected. In the second stage a RAP within the partition is selected.

When migrating from PDB to HALDB you may want to maintain the same partition boundaries. If so, a migration of a HIDAM database or secondary index would use key range partitioning with HALDB. A migration of an HDAM database would use a HALDB partition selection exit routine which assigns a root to a partition just as the randomization routine assigned a root to a PDB RAP range. This means that the logic used in the randomization routine would be moved to the partition selection exit routine.

When migrating from PDB to HALDB you may want to change partition boundaries. If so, you can use any scheme acceptable to HALDB. Most users choose key range partitioning, so a typical migration from HDAM with PDB to PHDAM would change partitioning to key range partitioning.

### 8.7.2 DBD changes

PDB databases are defined by DBDs. These DBDs do not contain DATASET statements, but contain PART statements. There is a PART statement for each partition. When migrating from PDB to HALDB you should delete these PART statements from the DBD. Other changes are the same that you would make for other IMS databases as described in 8.1.1, “DBD changes” on page 90. The PART statements for HIDAM, HISAM, and secondary index PDB databases contain the high key value. If you want use the same partition boundaries with HALDB, you should use these values in the Partition Definition utility or your DBRC INIT.PART statements.

If HISAM is used with PDB, it must be converted to PHIDAM.

### 8.7.3 Using standard IMS utilities for the migration

You may migrate a PDB HDAM or HIDAM database as you would migrate other HDAM or HIDAM databases. This uses the IMS HD Unload and IMS HD Reload utilities. HD Unload will read an entire PDB database. It cannot be used to unload a single partition. Using this technique you cannot migrate partitions in parallel. If you have secondary indexes you must use the MIGRATX=YES statement with HD Unload.

You cannot use this technique to migrate a PDB HISAM database to HALDB. The MIGRATE=YES and MIGRATX=YES control statements are not valid when unloading a HISAM database.

### 8.7.4 Using application programs for the migration

If you maintain the same partition boundaries when migrating to HALDB, you may want to migrate the partitions of your PDB database in parallel. One way of doing this is by using application programs. Each instance of an application program would read a different partition and create an output file of its segments. Another program reads the file and inserts in the segments in the HALDB database. Different instances of this program would read the different files. Figure 8-4 illustrates this technique.

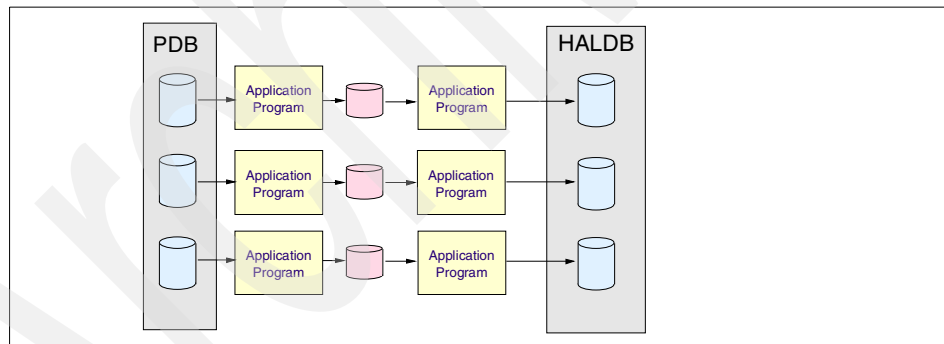


Figure 8-4 Migration from PDB using application programs with PROCOPT=L

If you modify your partition boundaries, the load programs cannot use PROCOPT=L. If you are migrating from PDB HDAM to PHDAM, you are unlikely to maintain the same partition boundaries. If so, you cannot use PROCOPT=L for each program. Only one program can load segments (PROCOPT=L) in a partition. Multiple programs may insert segments in the same partition when using PROCOPT=I. Figure 8-5 on page 122 illustrates this technique.

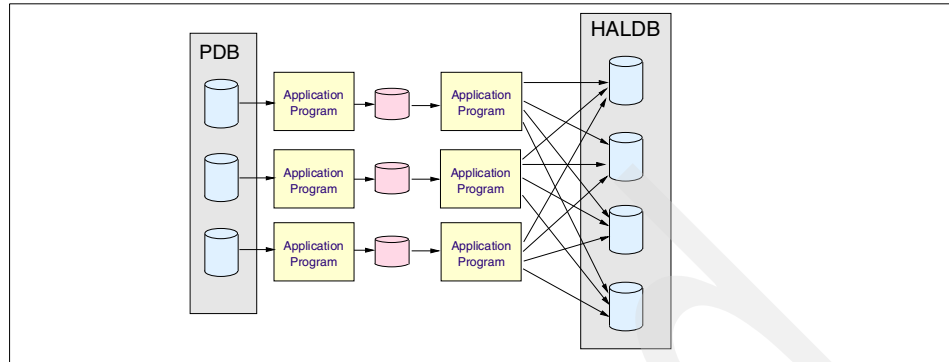


Figure 8-5 Migration from PDB using application programs with PROCOPT=I

These techniques are valid for migrating PDB HISAM databases to PHIDAM.

### 8.7.5 Using PSU Unload and application programs for the migration

PDB has its own unload and reload utilities. They are PSU Unload and PSU Reload. They are used to reorganize individual partitions or reorganize partitions in parallel. The PSU Unload utility does not have the capability to create an unload data set with HALDB headers. In other words, it does not have the capabilities the HD Unload has when MIGRATE=YES or MIGRATX=YES is used. There are no such control statements for the PSU Unload utility. Nevertheless, you may be able to use PSU Unload to avoid writing the application unload programs shown in Figure 8-4 on page 121 and Figure 8-5. This is shown in Figure 8-6 on page 123. Each instance of PSU Unload creates an unload file. This file is used as input to the application program that inserts the segments in the HALDB database. The file contains a header record, one record for each segment, and a trailer record. The segment record includes the segment name and the segment data. The output of PSU Unload may be mapped by using the DSECT in the IMS DFSURGUF macro in SDFS MAC. If the same partition boundaries are maintained, the load programs may use PROCOPT=L. If not, PROCOPT=I must be used to avoid loading a partition multiple times.

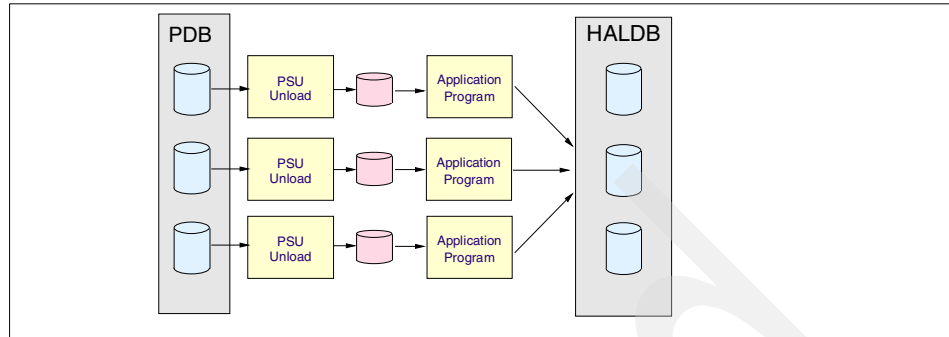


Figure 8-6 Migration from PDB using PSU Unload

This technique is valid for migrating PDB HISAM databases to PHIDAM.

You cannot use HD Reload or PSU Reload to load the HALDB when using PSU Unload to unload the PDB database. Migration with HD Reload requires that the input file contain HALDB prefixes. PSU Unload does not create these prefixes. PSU Reload cannot load a HALDB database.

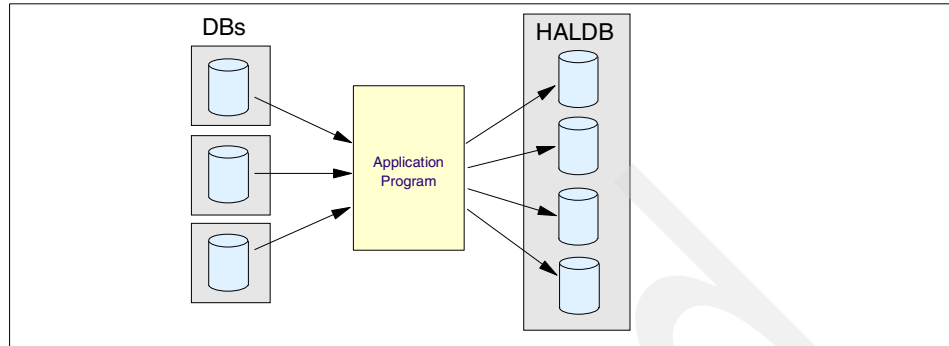
## 8.8 Migrating from user partitioning

Some installations have done their own partitioning. Typically, they did this to address database size limitations before PDB or HALDB was available. This technique is called user partitioning. With this technique the installation creates multiple IMS databases. Each database contains a portion of the records that the application wants to view as one database.

If you have user partitioning, you will probably want to migrate these databases to a single HALDB database. The migration may be done with user-written application programs or a combination of the user programs and the HD Unload utility.

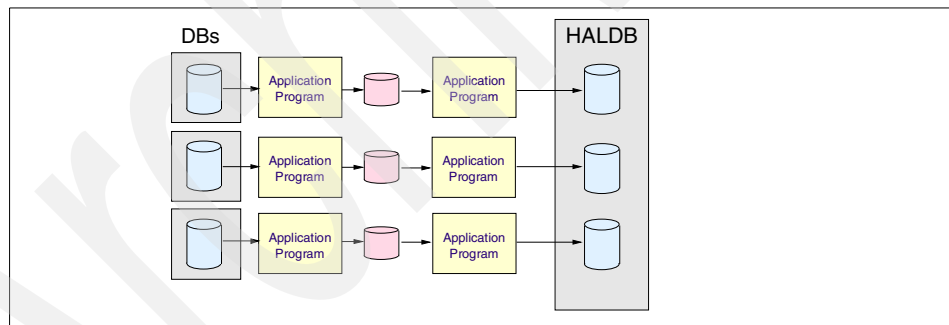
### 8.8.1 Using application programs for the migration

You may write one or more application programs to migrate a user-partitioned database to HALDB. Figure 8-7 on page 124 illustrates the use of one application program. The program reads the segments from the existing databases and inserts them in the HALDB database. PROCOPT=L may be used for the HALDB database. Since the application must have PCBs for each of these databases, the HALDB database must have a different DBD name from any of the non-HALDB databases. Partition boundaries may be different in the two implementations.



*Figure 8-7 Migration from user partitioning with a single program*

Alternatively, you may use multiple application program executions to migrate from user partitioning. Figure 8-8 illustrates an implementation of this technique. Separate instances of the same application program are used to read the segments in existing databases. Each instance creates an output file of its segments. These files are read by separate instances of another application program, which inserts the segments in the HALDB using PROCOPT=L. Segments loaded by different instances of the insert programs cannot be placed in the same partition. You cannot load a partition more than once.



*Figure 8-8 Migration from user partitioning with multiple load programs*

If segments from two different user-partitioned databases might be inserted in the same HALDB partition, PROCOPT=I must be used. PROCOPT=I avoids the restriction that a partition cannot be loaded more than one time. This technique is illustrated in Figure 8-9 on page 125.

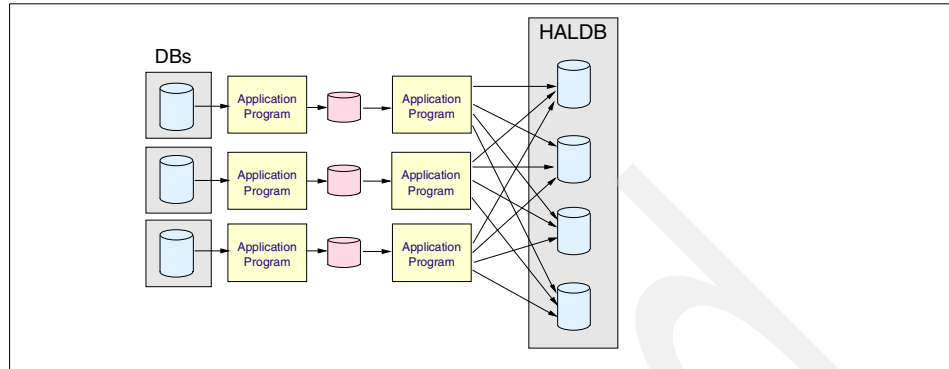


Figure 8-9 Migration from user partitioning with multiple insert programs

## 8.8.2 Using HD Unload and application programs for the migration

The techniques in Figure 8-8 on page 124 and Figure 8-9 may be modified to use the HD Unload utility. This eliminates the need to write an application program to read the databases. This is shown in Figure 8-10. Each instance of HD Unload creates an unload file. This file is used as input to the application program that inserts the segments in the HALDB database. The file contains a header record, one record for each segment, and a trailer record. The segment record includes the segment name and the segment data. The output of HD Unload may be mapped by using the DSECT in the IMS DFSURGUF macro in SDFSMAAC. Do not include a MIGRATE=YES or MIGRATX=YES control statement for the unloads. This would add extra overhead to their executions.

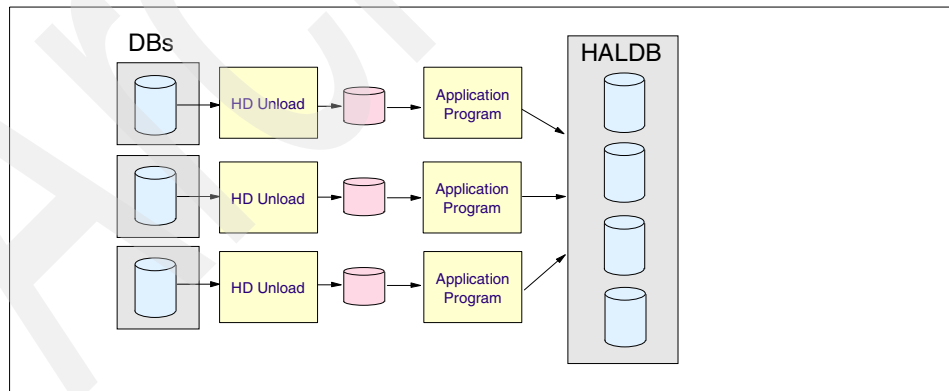


Figure 8-10 Migration from user partitioning using HD Unload

### 8.8.3 Warning on the use of HD Reload

HD Reload should not be substituted for the application programs shown in Figure 8-10 on page 125. When you use HD Reload in conjunction with HD Unload with a MIGRATE=YES or MIGRATX=YES control statement, the ILKs created for inserted segments are based on their locations in the non-HALDB databases. These ILKs use the relative byte address (RBA). These RBAs would not be unique with user partitioning since the same RBA could exist in multiple databases. If two segments with the same ILK were ever placed in the same partition, the HALDB self-healing pointer scheme would fail. This is a database integrity exposure.

## 8.9 Fallback to non-HALDB

IMS provides a capability to convert a HALDB database to its previous state as a non-HALDB database.

If you have not updated the database after a migration to HALDB, you would restore the non-HALDB database from the last image copy and any necessary logs. Of course you would also have to change the DBD to its former specifications and delete and reregister the database to DBRC.

If you have updated the database after a migration to HALDB, you may not be able to fall back. If the data will no longer fit in the space limitations of a non-HALDB database, fallback would not be possible. If the fallback is possible, the steps for this fallback are the following:

1. Unload the database using HD Reload with a FALLBACK=YES control statement.
2. Perform DBDGEN.
3. Delete the HALDB information for the database from DBRC.
4. Register the non-HALDB database to DBRC.
5. Execute Database Prereorganization using the DBR control statement.
6. Reload the database using HD Reload.

The database data sets must be image copied after the fallback.

If the database has secondary indexes, its secondary indexes must be rebuilt.

If the database has logically related databases they must be converted at the same time. See 8.9.2, "Fallback with logical relationships" on page 127, for additional considerations with logically related databases.



### 8.9.1 Fallback with secondary indexes

Secondary indexes are rebuilt as part of the fallback process for the indexed database. Fallback uses the same process that you use when reorganizing the non-HALDB database. This can be the execution of Prefix Resolution, HISAM Unload, and HISAM Reload, or the execution of IMS Index Builder, or the use of any tool that builds secondary indexes from the indexed database.

### 8.9.2 Fallback with logical relationships

The fallback process for logically related databases includes the normal utilities for resolving logical relationships as part of a reorganization of a non-HALDB database. This is the execution of Prereorganization, Prefix Resolution, and Prefix Update.

If your non-HALDB databases used virtual pairing, the fallback process does not allow you to fall back to your previous definitions. The migration to HALDB required a conversion to physical pairing. Fallback maintains physical pairing. Your new non-HALDB DBDs must include physical pairing definitions. The additional space required for the physically paired segments could prevent a fallback.



## Migration examples

This chapter provides the following migration examples:

- ▶ Migration of a HIDAM database without any secondary indexes to PHIDAM
- ▶ Migration of a HIDAM database with a secondary index to PHIDAM with PSINDEX
- ▶ Migration of a PDB HIDAM database to PHIDAM

**Note:** These examples do not include provisions for fallback. You should include them. As a minimum, you should be prepared to restore the database to its state before the migration. This may be accomplished by making image copies of the non-HALDB database data sets, copying the RECONs, and saving copies of the non-HALDB DBDs. These copies could be used to restore the database. See 8.9, “Fallback to non-HALDB” on page 126, for information on falling back and including updates made while the database was HALDB.

## 9.1 Migration of HIDAM database to PHIDAM

In this example, we migrate the New Order (NORDDDB) database from HIDAM to PHIDAM. It has no secondary indexes. The database has over 190,000 segments. The migration includes the following steps.

1. Image copy HIDAM database data sets.
2. Run the Migration Aid utility.
3. Unload the HIDAM database.
4. Save database information.
5. Delete the database from the RECONs.
6. DBDGEN for HALDB.
7. Define partitions using PDU.
8. Allocate HALDB data sets.
9. Initialize HALDB partitions.
10. Load HALDB database.
11. Image copy HALDB database data sets.
12. ACBGEN.

### 9.1.1 Image copy of non-HALDB database data sets

We take image copies of the original non-HALDB database data sets before we start the migration to HALDB.

### 9.1.2 Running the Migration Aid utility

Before we start the actual migration, we run the HALDB Migration Aid utility (DFSMAID0) to analyze the existing database. We run it several times varying the input parameters. Example 9-1 shows the JCL we use for DFSMAID0. (Note that you need to specify the DD cards for the databases in the JCL if you do not have the fix for APAR PQ63751 for IMS Version 7 or PQ68573 for IMS Version 8.)

*Example 9-1 JCL for HALDB Migration Aid (DFSMAID0) utility*

---

```
//JOUK01H JOB (999,P0K),JOUK01,CLASS=A,NOTIFY=&SYSUID,
//          MSGLEVEL=(1,1),MSGCLASS=X
/*JOBPARM SYSAFF=SC42
//*
//MAID0 EXEC PGM=DFSRRCO0,PARM='ULU,DFSMAID0,NORDDDB,,0,,,,,,,,,N,N'
//STEPLIB DD DSN=IMSPSA.IMOA.MDALIB,DISP=SHR
//          DD DSN=IMSPSA.IMSO.SDFSRESL,DISP=SHR
//DFSRESLB DD DSN=IMSPSA.IMSO.SDFSRESL,DISP=SHR
//NORDDD DD DSN=IMSPSA.IMOA.NORDDDB,DISP=SHR
//NOIXDD DD DSN=IMSPSA.IMOA.NORDPI,DISP=SHR
//IMS DD DSN=IMSPSA.IMSO.DBDLIB,DISP=SHR
//SYSPRINT DD SYSOUT=*
```

```
//SYSUDUMP DD SYSOUT=*
//DFSVSAMP DD DSN=IMSPSA.IMOA.PROCLIB(DFSVMON),DISP=SHR
//SYSIN DD *
DBNAME=NORDDB NBR=4
```

---

We run our first analysis using a control statement with NBR=4. This gives the key ranges and statistics for four partitions of equal size. Example 9-2 shows the output of this execution of DFSMAID0.

The main purpose of the first run is to get a rough idea of the possible partition characteristics.

*Example 9-2 The output of DFSMAID0 with NBR=4 specified*

---

```
-----
-
DFSMAID0 1.00 partition report for HIDAM database 'NORDDB ',
  Fri Feb 14 02:28:08 2003
  root key length = 14 bytes:

DFSMAID0 SYSIN:
  " DBNAME=NORDDB NBR=04
  "
    nbr = 4 ;

-----
-
segment types in database 'NORDDB ':

  1) 'NORDER ' ( dmbvldfg=00x sdbtflg=04x sdbptds=60x
)

-----
-
partition 1 :

  minimum key =
    +0000 f0f0f0f1 f0f1f0f0 f0f0f2f1 f0f1 |00010100002101 |
  maximum key =
```

	+0000	f0f0f0f6	f0f3f0f0	f0f0f2f4	f4f3	00060300002443	
			segments	bytes	prefix-incr	length-incr	
1) 'NORDER'			48270	1158480	386160	0	
SUM)			48270	1158480	386160	0	

partition 2 :

minimum key =

+0000	f0f0f0f6	f0f3f0f0	f0f0f2f4	f4f4	00060300002444	
-------	----------	----------	----------	------	----------------	--

maximum key =

+0000	f0f0f1f1	f0f5f0f0	f0f0f2f9	f0f2	00110500002902	
-------	----------	----------	----------	------	----------------	--

			segments	bytes	prefix-incr	length-incr	
1) 'NORDER'			48270	1158480	386160	0	
SUM)			48270	1158480	386160	0	

partition 3 :

minimum key =

+0000	f0f0f1f1	f0f5f0f0	f0f0f2f9	f0f3	00110500002903	
-------	----------	----------	----------	------	----------------	--

maximum key =

+0000	f0f0f1f6	f0f8f0f0	f0f0f2f2	f0f8	00160800002208	
-------	----------	----------	----------	------	----------------	--

			segments	bytes	prefix-incr	length-incr	
1) 'NORDER'			48270	1158480	386160	0	
SUM)			48270	1158480	386160	0	

partition 4 :

minimum key =

+0000 f0f0f1f6 f0f8f0f0 f0f0f2f2 f0f9				00160800002209	
maximum key =					
+0000 f0f0f2f1 f1f0f0f0 f0f0f3f0 f0f0				00211000003000	
		segments	bytes	prefix-incr	length-incr
1) 'NORDER '		48268	1158432	386144	0
SUM)		48268	1158432	386144	0
-----					
-					
sum of partitions:					
		segments	bytes	prefix-incr	length-incr
1) 'NORDER '		193078	4633872	1544624	0
SUM)		193078	4633872	1544624	0
-----					

The report provides the minimum and maximum key values, the number of the segments, and the number of the bytes needed for each partition. It also provides a summary for all the partitions.

Another way to run the Migration Aid utility is to specify the maximum size in bytes for each partition. Then the utility gives the number of partitions needed and the key ranges. This is done by using the control statement MAX=nnnn. If you specify a value that is larger than 2147483647, then 2147483647 (2G - 1) will be used. This method could be useful, for example, when you want a partition to fit on a single volume. In our next test, when we specify MAX=1000000. This produces seven partitions for our database. We could have guessed this by summing the bytes and prefix-incr values from the first report and dividing that by one million.

For the purposes of this example, we think that four is the suitable number of partitions, but we want to have high keys to be set based on warehouse numbers. Our database has the warehouse number in the first four bytes of the key. These values range from 0001 to 0021. We want the partitions to be about the same size. We take the maximum key values for the partitions from the first run and round them to the closest warehouse limit. We run the Migration Aid utility one more time.

The key ranges that we use for this job are the following:

```
KR=C'00059999999999'  
KR=C'00109999999999'  
KR=C'00159999999999'
```

This job produces a report that shows the number of segments and the number of bytes needed for each of the four partitions. Note that we only specify three values for the high keys. The last partition has the maximum value for its high key (16 bytes of X'FF'). Example 9-3 shows the output of the job.

*Example 9-3 The output of DFSMAID0 with key ranges specified*

---

```
DFSMAID0 1.00 partition report for HIDAM database 'NORDDDB ',  
Fri Feb 14 02:36:14 2003  
root key length = 14 bytes:
```

```
DFSMAID0 SYSIN:  
"KR=C'00059999999999"  
"  
"KR=C'00109999999999"  
"  
"KR=C'00159999999999"  
"
```

partitioning keys in effect:

```
+0000 f0f0f0f5 f9f9f9f9 f9f9f9f9 f9f9 |00059999999999 |  
+0000 f0f0f1f0 f9f9f9f9 f9f9f9f9 f9f9 |00109999999999 |  
+0000 f0f0f1f5 f9f9f9f9 f9f9f9f9 f9f9 |00159999999999 |
```

---

segment types in database 'NORDDDB ':

```
1) 'NORDER ' ( dmbvldfg=00x sdbtflg=04x sdbptds=60x )
```

---

partition 1 :

minimum key =

```
+0000 f0f0f0f1 f0f1f0f0 f0f0f2f1 f0f1 |00010100002101 |
```



maximum key =

+0000 f0f0f0f5 f1f0f0f0 f0f0f3f0 f1f7 |00051000003017 |

	segments	bytes	prefix-incr	length-incr
1) 'NORDER '	46107	1106568	368856	0
SUM)	46107	1106568	368856	0

partition 2 :

minimum key =

+0000 f0f0f0f6 f0f1f0f0 f0f0f2f1 f0f1 |00060100002101 |

maximum key =

+0000 f0f0f1f0 f1f0f0f0 f0f0f3f0 f2f6 |00101000003026 |

	segments	bytes	prefix-incr	length-incr
1) 'NORDER '	45930	1102320	367440	0
SUM)	45930	1102320	367440	0

partition 3 :

minimum key =

+0000 f0f0f1f1 f0f1f0f0 f0f0f2f1 f0f1 |00110100002101 |

maximum key =

+0000 f0f0f1f5 f1f0f0f0 f0f0f3f0 f2f5 |00151000003025 |

	segments	bytes	prefix-incr	length-incr
1) 'NORDER '	46205	1108920	369640	0
SUM)	46205	1108920	369640	0

partition 4 :

minimum key =

+0000 f0f0f1f6 f0f1f0f0 f0f0f2f1 f0f1  00160100002101				
maximum key =				
+0000 f0f0f2f1 f1f0f0f0 f0f0f3f0 f0f0  00211000003000				
	segments	bytes	prefix-incr	length-incr
1) 'NORDER '	54836	1316064	438688	0
SUM)	54836	1316064	438688	0
-----				
sum of partitions:				
	segments	bytes	prefix-incr	length-incr
1) 'NORDER '	193078	4633872	1544624	0
SUM)	193078	4633872	1544624	0
-----				
=====				

### 9.1.3 Unloading the HIDAM database

The next step in the conversion is to unload the database. For this example we use the standard IMS HD Reorganization Unload utility (DFSURGU0) and specify MIGRATE=YES in our SYSIN data. Example 9-4 shows the JCL used for this job.

*Example 9-4 JCL for unload of HIDAM database*

```
//NORDUNLJ JOB CLASS=A,MSGCLASS=X,REGION=OM
//*
//ULU EXEC PGM=DFSRRCOO,PARM='ULU,DFSURGU0,NORDDDB,,,,,,N'
//STEPLIB DD DSN=IMSPSA.IMOA.MDALIB,DISP=SHR
// DD DSN=IMSPSA.IMS0.SDFSRESL,DISP=SHR
//DFSRESLB DD DSN=IMSPSA.IMS0.SDFSRESL,DISP=SHR
//IMS DD DSN=IMSPSA.IMS0.DBDLIB.OLD,DISP=SHR
// DD DSN=IMSPSA.IMS0.PSBLIB,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//DFSURGU1 DD DSN=IMSPSA.NORDDDB.UNLOAD,DISP=OLD
//NORDDDD DD DSN=IMSPSA.IMOA.NORDDDB,DISP=SHR
//NOIXDD DD DSN=IMSPSA.IMOA.NORDPI,DISP=SHR
//DFSVSAMP DD DSN=IMSPSA.IMOA.PROCLIB(DFSVMOS),DISP=SHR
//SYSIN DD *
MIGRATE=YES
```

```

/*
//DFSCTL DD *
SBPARM ACTIV=COND
/*

```

---

From the SYSPRINT, we check the number of unloaded segments and the return code. These are shown in Example 9-5.

*Example 9-5 Output of unload job for HIDAM database*

---

```

TOTAL SEGMENTS IN DATA BASE =          193,078
DFS339I      FUNCTION DU HAS COMPLETED NORMALLY RC=0

```

---

## 9.1.4 Saving database information

We take a backup of the RECON data sets and the DBD source. We would need this information if we need to fall back to our original HIDAM database.

## 9.1.5 Deleting the database from the RECONS

Because we are converting the database from HIDAM to PHIDAM using the same database name, we need to delete the database information from the RECONS. We must do this before we can register the database as HALDB. Example 9-6 shows the DBRC utility job to delete the HIDAM database, NORDDDB, and its primary index, NOIX.

*Example 9-6 JCL for deleting database from the RECON*

---

```

//DBRC      EXEC PGM=DSPURX00
//STEPLIB   DD DISP=SHR,DSN=IMSPSA.IMS0.SDFSRESL
//          DD DISP=SHR,DSN=IMSPSA.IM0A.MDALIB
//SYSPRINT   DD SYSOUT=*
//SYSUDUMP   DD SYSOUT=*
//IMS       DD DISP=SHR,DSN=IMSPSA.IMS0.DBDLIB
//JCLPDS     DD DISP=SHR,DSN=IMSPSA.IM0A.PROCLIB
//JCLOUT     DD SYSOUT=(A,INTRDR)
//JCLOUTS    DD SYSOUT=*
//SYSIN      DD *
            DELETE.DB DBD(NOIX)
            DELETE.DB DBD(NORDDDB)
/*

```

---

Example 9-7 on page 138 shows the SYSPRINT from the successful deletion of the database information from the RECONS.

*Example 9-7 The output of DBRC database deletion*

---

```
IMS VERSION 8 RELEASE 1 DATA BASE RECOVERY CONTROL
DELETE.DB DBD(NOIX)
DSP0203I COMMAND COMPLETED WITH CONDITION CODE 00
DSP0220I COMMAND COMPLETION TIME 2003.045 19:04:12.4 -05:00
IMS VERSION 8 RELEASE 1 DATA BASE RECOVERY CONTROL
DELETE.DB DBD(NORDDB)
DSP0203I COMMAND COMPLETED WITH CONDITION CODE 00
DSP0220I COMMAND COMPLETION TIME 2003.045 19:04:12.6 -05:00
IMS VERSION 8 RELEASE 1 DATA BASE RECOVERY CONTROL
DSP0211I COMMAND PROCESSING COMPLETE
DSP0211I HIGHEST CONDITION CODE = 00
***** BOTTOM OF DATA *****
```

---

### 9.1.6 Changing the DBD and running the DBDGEN

We make the following changes to convert our DBD from HIDAM to PHIDAM:

- ▶ Change ACCESS=HIDAM to ACCESS=PHIDAM in the DBD statement.
- ▶ Remove the DATASET statement. The Partition Definition utility is used to define the database data sets.
- ▶ Remove the LCHILD statement for the primary index. The primary index definition for the PHIDAM database is automatically generated from the PHIDAM definition.
- ▶ We do not have data set groups, so we do not have to code a DSGROUP parameter on our SEGM statements.

**Note:** We discard the DBD for the primary index. The primary index definition is generated from the information in the PHIDAM DBD.

Example 9-8 shows the DBD source for the database before it is converted to HALDB.

*Example 9-8 DBD source for NORDDB HIDAM database*

---

```
DBD NAME=NORDDB,ACCESS=(HIDAM,OSAM)
*
DATASET DD1=NORDDDD,DEVICE=3390,SIZE=26624,FRSPC=(2,19)
*
SEGM NAME=NORDER,BYTES=14,PARENT=0,PTR=TB
FIELD NAME=(NOKEY,SEQ,U),BYTES=14,START=1,TYPE=C
*
WWWDD00000000
*
4 2 8
FIELD NAME=NOWID,BYTES=4,START=1,TYPE=C
FIELD NAME=NODID,BYTES=2,START=5,TYPE=C
```

```

FIELD NAME=NOOID,BYTES=8,START=7,TYPE=C
*
LCHILD NAME=(INDXSEG,NOIX),PTR=INDX
*
DBDGEN
FINISH
END

```

---

Example 9-9 shows the DBD source for the database after it is converted to HALDB.

*Example 9-9 DBD source for NORDDB PHIDAM database*

```

DBD NAME=NORDDB,ACCESS=(PHIDAM,OSAM)
*
SEGM NAME=NORDER,BYTES=14,PARENT=0,PTR=TB
FIELD NAME=(NOKEY,SEQ,U),BYTES=14,START=1,TYPE=C
*
WWWDD00000000
*
4 2 8
FIELD NAME=NOWID,BYTES=4,START=1,TYPE=C
FIELD NAME=NODID,BYTES=2,START=5,TYPE=C
FIELD NAME=NOOID,BYTES=8,START=7,TYPE=C
*
DBDGEN
FINISH
END

```

---

### 9.1.7 Defining partitions with PDU

We use the HALDB Partition Definition utility (PDU) to define the partitions of the HALDB database. We use the high key values that we found by running the HALDB Migration Aid utility (DFSMAID0), as explained in 9.1.2, “Running the Migration Aid utility” on page 130. For more details on how to use the Partition Definition utility, see Chapter 4, “Partition Definition utility” on page 45.

### 9.1.8 Allocate HALDB data sets

We use IDCAMS to allocate our database data sets.

- We use IDCAMS ALLOCATE for our OSAM data sets. We specify the block size that we define in PDU.

**Note:** IDCAMS cannot be used to allocate multi-volume OSAM data sets. Multi-volume OSAM data sets should be allocated in the reload step. One of the benefits of HALDB is that you can define partition sizes so that data sets always fit on one volume. This can eliminate potential multi-volume OSAM problems.

- ▶ We use IDCAMS DEFINE for our primary index clusters. We use the information from the RECOMMENDED VSAM DEFINE CLUSTER PARAMETERS in the SYSPRINT output of the PHIDAM DBDGEN. We specify our desired CI sizes in the DEFINES.
- ▶ We use IDCAMS DEFINE for our ILDSs. Since the NORDDDB database has no secondary indexes or logical relationships, the ILDS space allocations can be very small. Example 9-10 shows our DEFINE statements for an ILDS.

*Example 9-10 VSAM definition for defining ILDS*

---

```

DEFINE CLUSTER (NAME(IMSPSA.IMOA.NORDDDB.L00001) -
  VOLUMES(TOTIML) -
  CYLINDERS (1 1) -
  RECORDSIZE(50 50) -
  FREESPACE(25 10) -
  SHR(3 3) -
  SPEED -
  KEYS(9 0)) -
  INDEX(NAME(IMSPSA.IMOA.NORDDDB.L00001.INDEX)) -
  DATA(NAME(IMSPSA.IMOA.NORDDDB.L00001.DATA) -
  CISZ(4096) )

```

---

### 9.1.9 Initializing HALDB partitions

We must initialize the partitions. When we defined them, the PARTITION INIT NEEDED flags were turned on. We use the Database Prereorganization utility (DFSURPRO) to initialize the partitions. Example 9-11 shows the JCL for this job.

*Example 9-11 JCL for partition initialization using DFSURPRO*

---

```

//PRERSTEP EXEC PGM=DFSRRCO0,
//          PARM=(ULU,DFSURPRO,,,,,,,,,Y)
//STEPLIB DD DISP=SHR,DSN=IMSPSA.IMOA.MDALIB
//          DD DISP=SHR,DSN=IMSPSA.IMS0.SDFSRESL
//IMS      DD DISP=SHR,DSN=JOUK04.DBDLIB
//DFSRESLB DD DSN=IMSPSA.IMS0.SDFSRESL,DISP=SHR
//RECON1   DD DSN=IMSPSA.IMOA.RECON1,DISP=SHR
//RECON2   DD DSN=IMSPSA.IMOA.RECON2,DISP=SHR
//RECON3   DD DSN=IMSPSA.IMOA.RECON3,DISP=SHR

```

```
//SYSUDUMP DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//PRINTDD DD SYSOUT=*
//DFSURCDS DD DISP=OLD,DSN=IMSPSA.IMS0.DFSURCD5
//DFSVSAMP DD DSN=IMSPSA.IM0A.PROCLIB(DFSVSM0S),DISP=SHR
//SYSIN DD *
DBIL=NORDDB
/*
```

### 9.1.10 Loading HALDB database

We use the HD Reorganization Reload utility (DFSURGL0) with DBRC=Y to load the database. The database data sets are allocated dynamically using information in the RECONS. Example 9-12 shows the JCL for loading the database.

Example 9-12 JCL for loading the HALDB database

```
//LOAD EXEC PGM=DFSRR00,PARM='ULU,DFSURGL0,NORDDB,,,,,,,,,Y'
//STEPLIB DD DISP=SHR,DSN=IMSPSA.IMS0.SDFSRESL
//IMS DD DISP=SHR,DSN=JOUK04.DBDLIB
//RECON1 DD DISP=SHR,DSN=IMSPSA.IM0A.RECON1
//RECON2 DD DISP=SHR,DSN=IMSPSA.IM0A.RECON2
//RECON3 DD DISP=SHR,DSN=IMSPSA.IM0A.RECON3
//SYSPRINT DD SYSOUT=*
//DFSUINPT DD DISP=SHR,DSN=IMSPSA.NORDDB.UNLOAD
//DFSURWF1 DD DUMMY
//DFSURCDS DD DUMMY
//DFSVSAMP DD DSN=IMSPSA.IM0A.PROCLIB(DFSVSM0S),DISP=SHR
//DFSCTL DD *
SBPARM ACTIV=COND
//
```

Example 9-13 shows output from the reload.

Example 9-13 Output of HD Reload

SEGMENT LEVEL STATISTICS				
TOTAL SEGMENTS BY SEGMENT TYPE				
SEGMENT NAME	SEGMENT LEVEL	RELOADED	DIFFERENCE	
NORDER	1	193,078		
TOTAL SEGMENTS IN DATA BASE				
UNLOADED	SEGMENTS	IN DATA RELOADED	BASE DIFFERENCE	

```

DATABASE NORDDDB  HAS BEEN SUCCESSFULLY RELOADED BY FUNCTION DR
FUNCTION DR HAS COMPLETED NORMALLY RC=00
*****
***** BOTTOM OF DATA *****

```

---

### 9.1.11 Image copy HALDB data sets

At the completion of the reload, the *image copy needed* flags are set for all of the database data sets. We run an image copy job for each partition data set.

We do not image copy the ILDS or primary index data sets. These data sets do not use image copies for recoveries. Instead, if they are lost they are rebuilt using the HALDB Index/ILDS Rebuild utility (DFSPREC0). There are no image copy needed flags for these data sets.

### 9.1.12 ACBGEN

Finally, we run an ACBGEN. The ACBGEN process for HALDB is the same as for non-HALDB databases.

## 9.2 Migration of a database with a secondary index

In this example, we migrate the customer (CUSTDB) database from HIDAM to PHIDAM. The database has approximately 630000 segments. It has a secondary index. The migration is done using the MIGRATX=YES control statement with unload.

The procedure is similar to the previous migration example in 9.1, “Migration of HIDAM database to PHIDAM” on page 130.

1. Image copy the HIDAM database and secondary index data sets.
2. Run the Migration Aid utility.

We run it against both the HIDAM database and the secondary index. See 8.5.2, “Estimating the sizes of secondary index partitions” on page 107, for information on using the output of the utility with secondary indexes.

3. Unload the HIDAM database.

Only the HIDAM database is unloaded. We do not unload the secondary index.

We execute HD Unload with a SYSIN control statement specifying MIGRATX=YES. We include DD statements for DFSWRK01 and DFSSRT01.



These are output files. The DFSWRK01 data set contains the records used to load the HALDB secondary index. The DFSSRT01 data set contains the sort control statements used to sort the DFSWRK01 data set before the reload of the HALDB secondary index.

Example 9-14 shows the JCL used for the unload step.

*Example 9-14 JCL for unload with MIGRATX=YES*

---

```
//UNLOAD EXEC PGM=DFSRRCOO,PARM='ULU,DFSURGUO,CUSTDB,,0,,,,,,,,,N,N'
//STEPLIB DD DISP=SHR,DSN=IMSPSA.IMOA.MDALIB
// DD DISP=SHR,DSN=IMSPSA.IMSO.SDFSRESL
//DFSRESLB DD DSN=IMSPSA.IMSO.SDFSRESL,DISP=SHR
//IMS DD DISP=SHR,DSN=IMSPSA.IMSO.DBDLIB
//SYSPRINT DD SYSOUT=*
//CUSTDD DD DISP=SHR,DSN=IMSPSA.IMOA.CUSTDB
//CUSTSIDD DD DISP=SHR,DSN=IMSPSA.IMOA.CUSTSI
//DFSURGU1 DD DISP=(NEW,CATLG),DSN=JOUK04.CUSTDB.UNLOAD,UNIT=SYSDA,
// SPACE=(CYL,(750,250),RLSE)
//DFSWRK01 DD DISP=(,CATLG,DELETE),DSN=IMSPSA.CUSTDB.DFSWRK01,
// UNIT=SYSDA,SPACE=(CYL,(15,5))
//DFSSRT01 DD DISP=(,CATLG,DELETE),DSN=IMSPSA.CUSTDB.DFSSRT01,
// UNIT=SYSDA,SPACE=(TRK,(1,1))
//DFSVSAMP DD DSN=IMSPSA.IMOA.PROCLIB(DFSVMOS),DISP=SHR
//SYSIN DD *
MIGRATX=YES
/*
//DFSCTL DD *
SBPARM ACTIV=COND
```

---

If there were more than one secondary index, we would include more DFSWRKnn and DFSSRTnn DD statements.

#### 4. Sort the secondary indexes files.

We sort the DFSWRK01 output file from the previous step. The records in the DFSSRT01 file are used as sort control statements in SYSIN. The sorted output file will be used as an input to the load of the secondary index.

*Example 9-15 JCL for sorting secondary index file*

---

```
//SORT EXEC PGM=SORT
//SORTWK01 DD UNIT=3390,SPACE=(CYL,(50,10),,CONTIG)
//SORTWK02 DD UNIT=3390,SPACE=(CYL,(50,10),,CONTIG)
//SORTWK03 DD UNIT=3390,SPACE=(CYL,(50,10),,CONTIG)
//SYSOUT DD SYSOUT=*
//SORTIN DD DSNAME=IMSPSA.CUSTDB.DFSWRK01,DISP=(OLD,PASS)
//SORTOUT DD DSNAME=JOUK04.CUSTDB.SORTED,DISP=(,CATLG),
// UNIT=SYSDA,SPACE=(CYL,(15,5))
//SYSIN DD DSNAME=IMSPSA.CUSTDB.DFSSRT01,DISP=OLD
```

---

5. Save database information.

We back up the RECONs and save the source for our HIDAM and secondary index DBDs.

6. Delete the HIDAM database and the secondary index from RECONs.

7. DBDGEN of PHIDAM and PSINDEX databases.

Example 9-16 shows the DBD source for the customer database before conversion to HALDB.

*Example 9-16 DBD source for CUSTDB HDAM database*

```
*****
* "CUSTDB" DATA BASE:  CUSTOMER
*****
      DBD NAME=CUSTDB,ACCESS=(HDAM,OSAM),RMNAME=(RANDCUST,11,54900)
*
      DATASET DD1=CUSTDD,DEVICE=3390,SIZE=8192
*
      SEGM NAME=CUSTOMER,BYTES=679,PARENT=0
      FIELD NAME=(CKEY,SEQ,U),BYTES=10,START=1,TYPE=C
*
      WWWDDCCCC
*
      4    2  4
      FIELD NAME=CWID,BYTES=4,START=1,TYPE=C
      FIELD NAME=CDID,BYTES=2,START=5,TYPE=C
      FIELD NAME=CID,BYTES=4,START=7,TYPE=C
      FIELD NAME=/CKCID,BYTES=4,START=7,TYPE=C
      FIELD NAME=CFIRST,BYTES=16,START=11,TYPE=C
      FIELD NAME=CMIDDLE,BYTES=2,START=27,TYPE=C
      FIELD NAME=CLAST,BYTES=16,START=29,TYPE=C
      FIELD NAME=CSTREET1,BYTES=20,START=45,TYPE=C
      FIELD NAME=CSTREET2,BYTES=20,START=65,TYPE=C
      FIELD NAME=CCITY,BYTES=20,START=85,TYPE=C
      FIELD NAME=CSTATE,BYTES=2,START=105,TYPE=C
      FIELD NAME=CZIP,BYTES=9,START=107,TYPE=C
*
      LCHILD NAME=(CUSTNAME,CUSTSI),PTR=INDX
      XDFLD NAME=XCNAME,SRCH=(CWID,CDID,CLAST,CFIRST),          X
      SUBSEQ=(/CKCID)
*
      DBDGEN
      FINISH
      END
```

Example 9-17 on page 145 shows the DBD source for the secondary index before conversion to HALDB.

\*\*\*\*\*

Example 9-18 shows the DBD source for the customer database after conversion to HALDB. The original versions of changed statements have been converted to comments. Our original database has a specialized randomizer that is sensitive to the number of RAPs in the database. Since we do not need a specialized randomizer, we change to DFSHDC40.

\*\*\*\*\*

Chapter 9. Migration examples 145



12. Image copy the PHIDAM and PSINDEX data sets.
13. ACBGEN.

## 9.3 Migrating a PDB database

In this example, we migrate an IMS/ESA Partition Support Product (PDB) database. The database is a PDB version of the New Order (NORDDDB) database from our example in 9.1, “Migration of HIDAM database to PHIDAM” on page 130. We migrate the PDB HIDAM database to PHIDAM.

The procedure is similar to the migration example in 9.1, “Migration of HIDAM database to PHIDAM” on page 130.

1. Image copy the database data sets.
2. Run the Migration Aid utility.
3. Unload the PDB HIDAM database.

We use the MIGRATE=YES control statement in the HD Reorganization Unload job. We cannot use the PDB Unload utility. It does not support the creation of an output file with HALDB headers for the records.

Each PDB partition has its own DD statement in the unload JCL. Example 9-20 shows the JCL for unloading the PDB HIDAM database.

*Example 9-20 JCL for unloading PDB HIDAM database*

---

```
//UNLOAD EXEC PGM=DFSRR00,PARM='ULU,DFSURGU0,NORDDB,,0,,,,,,,,,N,N'
//STEPLIB DD DISP=SHR,DSN=IMSPSA.IM0A.MDALIB
// DD DISP=SHR,DSN=IMSPSA.IMS2.SDFSRESL
//IMS DD DISP=SHR,DSN=IMSPSA.IMS0.DBDLIB
//SYSPRINT DD SYSOUT=*
//NOIXDD DD DISP=SHR,DSN=IMSPSA.IM0A.NORDPI
//NORDD1 DD DISP=SHR,DSN=IMSPSA.IM0A.NORDD1
//NORDD2 DD DISP=SHR,DSN=IMSPSA.IM0A.NORDD2
//NORDD3 DD DISP=SHR,DSN=IMSPSA.IM0A.NORDD3
//NORDD4 DD DISP=SHR,DSN=IMSPSA.IM0A.NORDD4
//DFSURGU1 DD DISP=(NEW,CATLG),DSN=JOUK04.NORDDB.UNLOAD,UNIT=SYSDA,
// SPACE=(CYL,(750,250),RLSE)
//DFSVSAMP DD DSN=IMSPSA.IM0A.PROCLIB(DFSVM0S),DISP=SHR
//SYSIN DD *
MIGRATE=YES
```

---

4. Save the database information.
5. Delete the database from the RECONS.
6. DBDGEN for HALDB.

The PART statements are deleted from the DBD source. In PDB these are used to describe partitions.

Example 9-21 shows the DBD definition for the PDB HIDAM database.

*Example 9-21 DBD source for NORDDDB PDB HIDAM database*

---

```

      DBD NAME=NORDDDB,ACCESS=(HIDAM,OSAM)
*
      PART DD1=NORDD1,SIZE=26624,FRSPC=(2,19),HIGHKEY=00059999999999
      PART DD1=NORDD2,SIZE=26624,FRSPC=(2,19),HIGHKEY=00109999999999
      PART DD1=NORDD3,SIZE=26624,FRSPC=(2,19),HIGHKEY=00159999999999
      PART DD1=NORDD4,SIZE=26624,FRSPC=(2,19),HIGHKEY=99999999999999
*
      SEGM NAME=NORDER,BYTES=14,PARENT=0,PTR=TB
      FIELD NAME=(NOKEY,SEQ,U),BYTES=14,START=1,TYPE=C
      WWWDD00000000
*
      4    2 8
      FIELD NAME=NOWID,BYTES=4,START=1,TYPE=C
      FIELD NAME=NODID,BYTES=2,START=5,TYPE=C
      FIELD NAME=NOOID,BYTES=8,START=7,TYPE=C
*
      LCHILD NAME=(INDXSEG,NOIX),PTR=INDX
*
      DBDGEN
      FINISH
      END

```

---

Example 9-22 shows the DBD definition for the HALDB PHIDAM database.

*Example 9-22 DBD source for NORDDDB HALDB PHIDAM database*

---

```

      DBD NAME=NORDDDB,ACCESS=(PHIDAM,OSAM)
*
      SEGM NAME=NORDER,BYTES=14,PARENT=0,PTR=TB
      FIELD NAME=(NOKEY,SEQ,U),BYTES=14,START=1,TYPE=C
      WWWDD00000000
*
      4    2 8
      FIELD NAME=NOWID,BYTES=4,START=1,TYPE=C
      FIELD NAME=NODID,BYTES=2,START=5,TYPE=C
      FIELD NAME=NOOID,BYTES=8,START=7,TYPE=C
*
      DBDGEN
      FINISH
      END

```

---

7. Define partitions using PDU.
8. Allocate the HALDB data sets.

9. Initialize the HALDB partitions.
10. Load the HALDB database.
11. Image copy the HALDB database data sets.
12. ACBGEN.

Archived





# Using the HALDB Conversion and Maintenance Aid

IMS High Availability Large Database Conversion and Maintenance Aid for z/OS™ (product number 5655-K47) is a product in the IBM Data Management Tools for IMS portfolio. IMS HALDB Conversion and Maintenance Aid is an ISPF application that allows you to analyze, model, and convert existing IMS database structures to the HALDB format.

IMS HALDB Conversion and Maintenance Aid creates DBD and DBRC definitions, allocates HALDB partition data sets, and creates the conversion JCL. Batch functions support the conversion process. This chapter provides an overview of the product and examples of its use.

**Note:** This chapter is based on a pre-GA version of IMS High Availability Large Database Conversion and Maintenance Aid for z/OS Version 2 product and may not apply as such when the product becomes generally available. We recommend that you consult the product documentation for more current information. For example, the title line on all ISPF panels has changed to reflect the final product name.

## 10.1 HALDB Conversion and Maintenance Aid product

The IMS HALDB Conversion and Maintenance Aid is designed to ease the conversion process from existing full function to HALDB databases. All necessary steps are created and then presented to you through an ISPF interface. You submit the steps and check the output for zero return codes. The IMS HALDB Conversion and Maintenance Aid then creates the next action. The process continues until the conversion is complete. The HALDB Conversion and Maintenance Aid Version 2 provides enhancements to the HALDB Conversion Aid Version 1 (product number 5655-I01). Both products include the following:

- ▶ Support for conversion of HDAM, HIDAM, HISAM, and secondary indexes to HALDB. HISAM is converted to PHIDAM.
- ▶ Support for migration from PDB or PDF.
- ▶ Conversion of DBDs.
- ▶ Allocation of database data sets.
- ▶ Allows user to choose the level of control for migration: Cruise control (largely automatic) or manual (step-by-step) or in between.
- ▶ Inclusion of a generic partition selection exit IHCPSEL0.
  - Based on subset of the root key
  - Specification of offset and length
  - Uses data string in DBRC definition

The HALDB Conversion and Maintenance Aid Version 2 provides the following additional functions:

- ▶ It can create DBD source from the DBD load library. This allows the DBD source libraries to be discarded.
- ▶ It provides a stand-alone EPS pointer healer utility. This allows you to heal all secondary index pointers after a reorganization.
- ▶ It has a maintenance function that helps in maintaining (splitting or combining partitions) your HALDB database once it has been converted from full function.
- ▶ It can improve the performance when initially loading databases with the secondary indexes. When a user application program creates a HALDB database, the product provides a capability not to insert secondary indexes immediately. They are saved in temporary files during the load process. Once the database is loaded, all indexes are sorted and loaded to the index KSDS file(s).
- ▶ It provides a capability to allow initial load programs to insert logical child segments. That is, it removes the normal HALDB restriction that does not

allow logical child segments to be inserted when a user application program is initially loading a database with logical relationships.

- ▶ It provides a function that can be used to enable BA status codes for applications that do not issue the INIT STATUS call. This will prevent those applications from abending with a U3303 abend. The second feature of this function allows you to define a retry mechanism to retry the call after receiving a BA status code. The third feature limits the BA handling to specified databases only.

Section 10.4, “Additional functions” on page 176, describes these new functions.

### 10.1.1 The conversion process overview

The conversion process for a database with IMS HALDB Conversion and Maintenance Aid consists of the following phases:

1. Phase 1 collects the necessary data and requests the DBDs that are to be converted.

The user selects the environment, that is, a set of RECONS, DBDLIBs, and other IMS data sets, and sets up a project for the database(s) to be converted by entering the answers to the questions presented in ISPF panels. For many questions, one option is to select **Ask during the conversion**. If selected, the user is prompted during the implementation phase to answer the question.

If the database has relationships, such as logical relationships or secondary indexes, those DBDs are automatically included. A collect job is created for every primary DBD, in order to gather the information.

2. Phase 2 prepares the conversion process.

The partition boundaries are set and the file allocations are prepared. During this phase IMS HALDB Conversion and Maintenance Aid reads old databases for modeling and calculates partition sizes or partition high keys based on the modeling information. IMS HALDB Conversion and Maintenance Aid provides an efficient alternative to the HALDB Migration Utility (DFSMAID0).

3. Phase 3 unloads the “old” databases.
4. Phase 4 implements the IMS HALDB Conversion and Maintenance Aid.

During the implementation phase, IMS HALDB Conversion and Maintenance Aid performs the following functions:

- It defines partitions in DBRC. It uses data from reading the old database to establish boundaries. It allows the user to adjust boundaries. All DBRC information for the old database is removed for DBRC.

- It generates the new DBD for the database(s). When converting the DBD, IMS HALDB Conversion and Maintenance Aid is able to handle virtual pairing to physical pairing conversion and multiple data set groups. The user has the choice to keep groups or combine them. IMS HALDB Conversion and Maintenance Aid also saves the old definitions.
- It allocates the database data sets. It uses the data from reading the old database for sizing. It allows the user to adjust sizes and other parameters.
- It initializes the partitions.
- It creates the jobs for the reload and is able to select between MIGRATE and MIGRATX based on whether the converted database(s) has secondary indexes or not.
- It creates image copies.
- It creates an optional EPS pointer heal job that you can run.

For all the jobs, the JCL is presented to the user. The user submits the jobs. The jobs maybe modified before they are submitted. An ISPF panel informs the user of the end of each phase.

5. Phase 5 is a reminder that after databases are converted, ACBGENs and online changes are required.

IMS HALDB Conversion and Maintenance Aid does not perform those actions.

The following full-function DBDs cannot be converted by using IMS HALDB Conversion and Maintenance Aid:

- ▶ HSAM
- ▶ Non-unique secondary indexes with five SUBSEQ parameters
- ▶ Logical related databases type 4; that is, those that appear to be virtually paired without a virtual segment

## 10.2 Activating the tool

After the SMP/E installation of HALDB Conversion and Maintenance Aid, there are only a few things you need to do before you can start using the product:

- ▶ Modify the startup CLIST for ISPF invocation and make it available to the users of the product.
- ▶ Allocate the environment description data set.

## 10.2.1 Modifying the startup CLIST

A sample CLIST for invoking the HALDB Conversion and Maintenance Aid in an ISPF environment is provided. You can find the sample CLIST in the product sample library *hlq.SIHCSAMP* where *hlq* is the high-level qualifier specified during the product installation. The CLIST has the name IHCXHAL in the sample library.

You may want to copy this CLIST to a CLIST library that is already included in your TSO allocation and modify it there. In the sample CLIST, the only things you need to modify are the ISPF library names for the product target libraries unless you are using the product default names. The names in the CLIST must match the target library names.

By copying the CLIST to the general CLIST library, the start command is available to all users who have the library defined in their TSO allocation. Then you do not have to specify the library name where the CLIST resides when using the start command. In our case we copied the CLIST into library SYS1.OS390.CLIST and gave the name \$IHCXHAL for it. The other way is to modify the CLIST in the SIHCSAMP library and specify the library name in the start command when starting the tool:

```
ex 'hlq.SIHCSAMP(IHCXHAL)'
```

## 10.2.2 Allocating the environment description data set

Before using the HALDB Conversion and Maintenance Aid, you must allocate the environment description data set. It is a VSAM key sequenced data set (KSDS) that the conversion process uses to define and maintain environment-related information. You can find the sample member IHCXKSDS in the product sample library (*hlq.SIHCSAMP*). Change the library names, the data set name of the KSDS to be allocated, and the volume information to meet your requirements, and run the modified JCL.

**Note:** The HALDB Conversion and Maintenance Aid load library must be APF authorized before running this allocation job.

## 10.3 Migration example with HALDB Conversion tool

This example explains how to use the IMS HALDB Conversion and Maintenance Aid tool to convert an existing HDAM database, CUSTDB, and its secondary index, CUSTSI, to PHDAM and PSINDEX. You have to perform the following tasks:

- Define an IMS environment.

- ▶ Define a conversion project.
- ▶ Process a conversion project.

To start the IMS HALDB Conversion and Maintenance Aid, use the CLIST that you had modified as described in the 10.2.1, “Modifying the startup CLIST” on page 155. In our case we can start the tool by using the command \$IHCXHAL from the ISPF command option or from any TSO command line by entering the command:

```
TSO $IHCXHAL
```

### 10.3.1 Define an IMS environment

Use the main menu (Figure 10-1) to select all major IMS HALDB Conversion and Maintenance Aid actions. When you use the tool the first time, an environment has not yet been defined and you must select option 1.

```

----- IMS HALDB Conversion Aid -----
Command ==>

  _  Select Function                                Current Settings
                                           Environment:
                                           IMS Version: 81

1  Add-Modify-Delete Environment
2  Select an Environment
3  Projects in Progress

4  Convert from Full Function to HALDB
5  Split or Consolidate HALDB Partitions

6  Database Utilities
7  Other Utilities
8  DBRC for HALDB

```

Figure 10-1 Main menu

After selecting option 1 you get the panel about environment maintenance (Figure 10-2 on page 157), where you choose option 2 to add a new environment.

```
----- IMS HALDB Conversion Aid -----  
Command ==>  
  
      Environment Maintenance  
  
    _ Select Function  
  
      1 Display Environment(s)  
      2 Add a new Environment  
      3 Delete Environment(s)  
      4 Change Environment Settings
```

*Figure 10-2 Environment maintenance*

Now you see a panel where you have to enter a name and a description for your environment. The next panel (Figure 10-3) shows an example.

```
----- IMS HALDB Conversion Aid -----  
Command ==>  
  
      Define a new Environment  
  
Environment Name: IM00A  
Description:      IMS V8 TEST
```

*Figure 10-3 Definition of the new environment*

In the next step you set definitions for your project (see Figure 10-4 on page 158).

```

----- IMS HALDB Conversion Aid -----
Command ==>

Project Definition Settings

Multiple DBDs:      2  1 Yes, multiple DBDs to one HALDB
                   2  2 No

Project Method:     2  1 Partitioning evaluation separate
                   2  2 Partitioning evaluation during UNLOAD

JCL Generation:     1  1 Single JOBS for each phase
                   2  2 One multistep JOB

```

Figure 10-4 Project definition

To set up an environment, the tool must know which data sets it should use (Figure 10-5). Specify your RESLIB, and perhaps a second data set for user program library (user exits, randomizer, etc.). For MACLIB enter the name of your MACLIB, and for LOADLIB the name of the IHC load library. Enter the data set name and member name that contains your DFSVSAMP member. Specify your MDALIB that is used to extract data set names of the databases and where the RECON MDA are located if you want them dynamically allocated. Otherwise specify the RECONS you want to use in the next panel. After that you see a panel where you define your DBDLIBs (up to nine).

```

----- IMS HALDB Conversion Aid -----
Command ==>

Datasets
RESLIB  :  IMSPSA.IMS0.SDFSRESL
        :
MACLIB  :  IMSPSA.IMS0.SDFSMACT
LOADLIB :  IHC.V2PTF2.SIHCLOAD
DFSVSAMP :  IMSPSA.IM0A.PROCLIB
           Member name: DFSVSM0R
MDALIB  1:  IMSPSA.IM0A.MDALIB
        2:
RECON MDA located in:  4  1 RESLIB
                       2  2 MDALIB 1
                       3  3 MDALIB 2
                       4  4 Specify RECON datasets

```

Figure 10-5 Defining data sets



The HALDB Conversion and Maintenance Aid is designed to run with preset defaults. However, you can override defaults to accommodate the needs of your environment. The following panels describe available options. You can get help by pressing PF1. Default conversion rules are shown in Figure 10-6.

```

----- IMS HALDB Conversion Aid -----
Command ==>

                                DBD Conversion Rules

Multi DSG:                      2      1 To single DSG
                                       2 Leave as is
                                       3 Ask during Conversion

Convert VSAM to OSAM: 2      1 Yes
                                       2 No
                                       3 Ask during Conversion

Change DBD name:                2      1 Yes
                                       2 No

Selection Exit:                 2      1 Yes
                                       2 No
                                       3 Ask during Conversion

Heal Index Pointer:             1      1 Yes
                                       2 No
                                       3 Ask during Conversion

```

Figure 10-6 Conversion defaults

This panel contains the following fields:

- ▶ **Multi DSG:** Change to 1 if you want to convert multiple data set groups to a single one.
- ▶ **Convert VSAM to OSAM:** You may specify whether you want to convert VSAM to OSAM data sets. OSAM advantages are typically reduced CPU time and the availability of OSAM sequential buffering.
- ▶ **Change DBD name:** If you change the name you must also change the PSBs and logical DBDs (this tool does not perform these functions). If names are not changed, DBRC information for old databases is deleted.
- ▶ **Selection exit:** Specify if you want to add one.
- ▶ **Heal index pointer:** Specify whether to heal the index pointers. The migration process leaves the PSINDEX pointers in a healing needed status. If you want them healed specify option 1, which will create the necessary JCL to run the pointer healing utility for every PSINDEX database in the project.

If you choose option 3 for any field, the value for that option will be Ask during Conversion. These rules apply to all of your projects in the selected environment if you do not change them in your project definition.

The Save Source Statements panel is used to enable the saving of source statements (Figure 10-7). This function saves the source for new DBDs and the IDCAMS used for allocating the data sets into the specified data sets. These data sets must be allocated before making the selection.

```
----- IMS HALDB Conversion Aid -----
Command ===>

                        Save Source Statements

DBD Source:      1  1  No
                  2  Yes
Source File:      _____

IDCAMS Source:   1  1  No
                  2  Yes
                  3  Yes, but no DELETE file
Member Name:     3  1 DBD Name
                  2  Partition Name
                  3  DD Name
Del File:        _____
Def File:        _____
```

*Figure 10-7 Save source statements*

You can also specify the member name for the IDCAMS statements. With option 1 (DBDNAME) all IDCAMS statements for that DBD become part of a member with this name. With option 2 (partition name) all statements of that partition are included in a member with the partition name. With option 3 (DD name) the IDCAMS statement per DD name is saved. Option 3 provides maximum flexibility. We recommend that you save the IDCAMS statements. This allows you to delete and allocate the data sets very easily. It is not necessary to save the DBD source because you have the capability to see and reassemble it by choosing the appropriate panel option.

The Partition Definition Rules (Figure 10-8 on page 161) panel sets partition defaults at data collection time.

```

----- IMS HALDB Conversion Aid -----
Command ===>

Partition Definition Rules

Combine Database records:      5000 (number of database records)

Main DB Partition:      2      1 Fixed Number of Partitions      1
                             2 Fixed Partition Size (MB)          2048
                             3 Ask during Conversion
                             4 Specify High Keys

Index Partition:          2      1 Fixed Number of Partitions      1
                             2 Fixed Partition Size (MB)          1024

PDB Conversion:          2      1 Use existing Keys or Partitions
                             2 Create new Partition boundaries
                             3 Ask during Conversion

Additional Partitioning Layouts to be created?
                             2      1 Yes
                             2 No, use the one created during Collect

```

Figure 10-8 Partition Definition Rules

It contains the following fields:

► Combine Database records

Here you define how many database records should be combined for one data point. These data points are used to analyze potential partition boundaries and sizes. This parameter influences the performance of the creation of new partitioning layouts. The smaller the specified parameter, the greater the number of records stored in the project data set during the collection phase. More records allow for a finer granularity in the analyses.

► Main DB Partition

Specify how a partitioning layout is to be created during data collection. You can choose between number of partitions, size of partitions, specifying the high keys, or delaying the selection until the creation of the JCL.

► Index Partition

Specify the number of PSINDEX partitions to be created or their size.

► PDB Conversion

You have to decide whether the existing ranges should be used or new partition boundaries are to be created.

► Additional Partitioning Layouts to be created?

This function is available for the main database only.

Next you have to define the partition naming rules (Figure 10-9). A unique partition name must be created for each partition in your installation. The tool does not check the validity.

With Partition DSN specify the left-most portion of the partition data set name (for naming conventions see 1.9, “Naming conventions” on page 17), which applies to all data sets in your environment.

```
----- IMS HALDB Conversion Aid -----
Command ==>

Partition Naming Rules

Partition Name:  1      1 DBD name (up to 6) and 0-9,A-Z
                    2 DBD name (up to 6) and A-Z
                    3 Ask for partition name during Conversion
                    4 Ask for constant (6 char) during Conversion
                    5 DBD pattern ..***** and 0-9,A-Z
                      like ..****.*

Partition DSN:  1      1 High qualifier and DBD name
                    2 High qualifier and Partition name
                    3 Ask during Conversion

                    High-level qualifier
                    IMSPSA.IM0A
```

Figure 10-9 Naming

Now some space allocation panels follow. The first one defines the space allocation rules for PHDAM and PHIDAM databases (Figure 10-10 on page 163). In these panels the term *cloning* means that the old definitions should be taken. Similar panels for PSINDEX data sets and ILDS follow (not shown).

```

----- IMS HALDB Conversion Aid -----
Command ==>

                Space Allocation Rules
                for PHDAM and PHIDAM

HDAM RAA Size:   1      1 Clone from DBD
                  2 (1) and increase 10 percent
                  3 Ask during Conversion

Overflow Size:   1      1 Clone from original dataset
                  2 (1) and increase 10 percent
                  3 Ask during Conversion

Byte limit:      1      1 Same as old DBD
                  2 None
                  3 Ask during Conversion

SMS Allocation:   3      1 Yes Dataclass: _____
                  2 No, ask for Volume serial
                  3 Yes, no Dataclass

```

Figure 10-10 Rules for PHDAM and PHIDAM

JCL defaults for the JOB statement you submit for all conversion JCLs must be entered.

If you have the tools High Performance Unload or High Performance Reload you may use them instead of the standard HD Unload and Reload utilities. One panel offers the selection of the Unload utility or HP Unload (Figure 10-11). A similar one offers the selection of the Reload utility or HP Reload.

```

----- IMS HALDB Conversion Aid -----
Command ==>

                Utility Setup for UNLOAD

1      Select Unload type
        1 Standard IMS (DFSURGU0)
        2 High Performance Unload

1      Specify Dataset Name Rule for Unload File
        1 High Level Qualifier.dbd.timestamp
        2 High Level Qualifier.dbd.GDG
        3 Prompt for Dataset Name
          Specify High Level Qualifier
          imspsa

          imspsa.hpunload

```

Figure 10-11 Select utility for unload

The following panel is for the utility set up for backout (Figure 10-12). It provides some defaults for the standard Image Copy utility (DFSUDMP0). These are the number of copies you want and the naming scheme.

```
----- IMS HALDB Conversion Aid -----
Command ==>

      Utility Setup for BACKUP

      1  Select Backup type
        1  Standard IMS (DFSUDMP0)

      1  Select No. of copies (1 or 2)

      1  Specify Dataset Name Rule
        1  High Level Qualifier.dbd.ddname.IC.timestamp
        2  High Level Qualifier.ddname.IC.timestamp
        3  High Level Qualifier.dbd.ddname.IC.GDG
        4  High Level Qualifier.ddname.IC.GDG
        5  Use the one created by GENJCL.IC
           High Level qualifier
           YOUR.QUALIFIER
```

Figure 10-12 Utility set up for backup

Now the environment is set and you can start defining your project.

### 10.3.2 Define a project

We want to convert a full-function database to a new full-function HALDB, so we have to enter option 4 of the main menu (Figure 10-1 on page 156). From the Conversion panel we choose option 1 (Figure 10-13 on page 165) to add a new project.

```
..... IMS HALDB Conversion Aid .....  
Command ==>  
  
Convert Full Function to HALDB  
  
_ Select Function Current Settings  
Environment: IM0A  
Project:  
IMS Version: 81  
  
1 Add-Delete Conversion Project  
2 Projects in Progress  
3 Select a Project  
  
4 Start or Continue with the Current Project
```

*Figure 10-13 Conversion Full Function to HALDB*

Next we follow the guided panel tour to set some other defaults before returning to the Conversion panel. Basically you see the panels you already know from defining the environment. Now we use them for determining the values for our individual project. We may change the shown defaults or not. For example, if for this particular database we need a partition selection exit routine, we change the conversion default to Selection Exit: 1 (according to Figure 10-6 on page 159; 1 means YES). Then the following panel is shown (Figure 10-14 on page 166). We may specify which part of the root key the selection exit should use (offset and length) and how the partitions are to be defined.

```

----- IMS HALDB Conversion Aid -----
Command ===>

Specify a Partition Selection Exit for CUSTDB

IHCPSEL0 Partition Selection Module
  5 Key Offset (Offset from start of the key)
  2 String Length
    Root segment key length is 10
    (Offset length must not exceed root key)

2 Specify Strings
  1 Yes, prompt for strings
  2 No, number of partitions is 4

Use the HELP key for more information

```

Figure 10-14 Specify a Partition Selection Exit

If we want to be prompted, a panel will be presented where we may define our key strings. In a summary panel we get all strings we have entered and we may change them (Figure 10-15). In our migration example, however, we do not use the partition selection exit.

```

----- IMS HALDB Conversion Aid ----- Row 1 to 4 of 4
Command ===>

Keystings for DBD CUSTDB

Selections: "D" for DELETE      Key Length: 2
            "C" for CHANGE      Key Offset: 5
            "I" for INSERT

S      Key value
-----
Char: 03
Hex : F0F3

Char: 06
Hex : F0F6

Char: 09
Hex : F0F9

Char: .
Hex : FF

```

Figure 10-15 Key strings defined for CUSTDB partition selection exit

One other default we might want to change for a specific database is additional layouts. Perhaps we do not know the partition ranges and want to check which



would be the best choice. We select the appropriate row in Figure 10-8 on page 161 and specify 1 to view the predefined partition layout or create additional ones. We see a selection panel (Figure 10-16) and we must then decide which layout to use.

```
----- IMS HALDB Conversion Aid -----
Command ==>

Partition Layout Selection
DBD: CUSTDB

- Select Function
  1 View a stored layout
  2 Create a new layout
  3 Set the final layout
```

Figure 10-16 Additional Layout Selection

Our database CUSTDB should have four partitions, so the layout stored looks like Figure 10-17.

```
----- IMS HALDB Conversion Aid ----- Row 1 to 4 of 4
Command ==>

Partition layout
DBD: CUSTDB
```

Partition Number	Root Segments	Total Bytes	Data Bytes	Prefix Bytes
1	157500	109147500	106942500	2205000
Key: 0006031500				
Hex: FFFFFFFF				
0006031500				
2	157500	109147500	106942500	2205000
Key: 0011053000				
Hex: FFFFFFFF				
0011053000				
3	157500	109147500	106942500	2205000
Key: 0016081500				
Hex: FFFFFFFF				
0016081500				
4	157500	109147500	106942500	2205000
Key: 0021103000				
Hex: FFFFFFFF				
0021103000				

Figure 10-17 Stored layout for CUSTDB

We will use the stored layout. We select a new project, give it a name and a description and specify a tracking and staging KSDS (DSN, space, volser), which

we allocate by generated JCL. Back in the Conversion panel we select function 3 to display our projects and select the appropriate one (Figure 10-18).

```

.....
----- IMS HALDB Conversion Aid ----- Row 1 to 5 of 5
Command ==>

                Project List for Environment  IM0A

Sel   Project      Description
-----
-     CUST-DB      Conversion customer database to HALDB
      Type: C      DSN: JOUK01.CUST.KSDS

```

Figure 10-18 Panel displaying added project

### 10.3.3 Process a conversion project

Now we may start our project by selecting option 4. We get the Project Work Panel (Figure 10-19).

```

.....
----- IMS HALDB Conversion Aid -----
Command ==>

                Project Work Panel          Environment: IM0A
                                           Project:   CUST-DB

-   Select Function

      0 Setup Parameter
      1 Continue with Current Project
      2 Show Status of Project
      3 Recreate JCL of current phase
      4 Restart at current or prior level
      5 Restart from beginning

                                           Project Desc: Conversion customer database to HALDB
                                           Project KSDS: JOUK01.CUST.KSDS

```

Figure 10-19 Project Work Panel

With option 0 we may check all settings and eventually change them. However, we want to continue our project and select option 1, which gives us a list of DBDs to be selected. CUSTDB is the database we wish to convert, so we select it. We can select one DBD after the other. If we select the corresponding secondary index we get the message that it has already been selected because this tool automatically selects all related databases.

After the selection we will be presented with a panel that shows the DBDs indicated (Figure 10-20).

----- IMS HALDB Conversion Aid -----					----- Row 1 to 2 of 2
Command ==>					
DBD(s) in Project CUST-DB					
S	DBD	Type	Prim DBD	Log. Current Level Rel.	Current Status
-----					
	CUSTSI	X	CUSTDB	Collect DBDs	Wait for User
	CUSTDB	D		Collect DBDs	Wait for user
***** Bottom of data *****					

Figure 10-20 DBDs in project

We must now press the END key to display the continuation, which is the confirmation panel of our DBD selection. We specify Y for selection complete (N if we want to add more DBDs). When we have confirmed our selection, the following panel gives us the name of the unload data set (Figure 10-21).

----- IMS HALDB Conversion Aid -----					-----
Command ==>					
Output Dataset for Unload					
DBD CUSTDB to DBD CUSTDB					
Dsname : <u>IMSPSA.IM0A.CUSTDB.UNLOAD.CUSTDB</u>					

Figure 10-21 Panel naming unload file

Next we select whether we want to specify the high keys, a certain number of partitions, or a certain size for our partitions. In our example we simply say we want four partitions (Figure 10-22 on page 170).

```

----- IMS HALDB Conversion Aid -----
Command ==>

Select Partition Layout Method
DBD: CUSTDB

1 Select

1 Number of partitions 4
2 Partition size 9 (MB)
4 Specify high keys

```

Figure 10-22 Select partition layout

Now the setting is done and JCL for the unload is generated. We check and submit it (Example 10-1).

#### Example 10-1 Unload JCL

```

//DEL EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
DELETE IMSPSA.IMOA.CUSTDB.UNLOAD.CUSTDB
SET MAXCC=0
/*
//UNLOAD EXEC PGM=IHCALDB,COND=(4,LE),
// REGION=60M
//STEPLIB DD DISP=SHR,DSN=IHC.V2PTF2.SIHCLOAD
// DD DISP=SHR,DSN=IMSPSA.IMS0.SDFSRESL
//DFSRESLB DD DISP=SHR,DSN=IMSPSA.IMS0.SDFSRESL
//IMS DD DISP=SHR,DSN=JOUK01.DBDLIB
//RECON1 DD DISP=SHR,DSN=JOUK01.RECON1
//RECON2 DD DISP=SHR,DSN=JOUK01.RECON2
//RECON3 DD DISP=SHR,DSN=JOUK01.RECON3
//CUSTDD DD DISP=SHR,DSN=IMSPSA.IMOA.CUSTDB
//CUSTSIDD DD DISP=SHR,DSN=IMSPSA.IMOA.CUSTSI
//IHC PROJ DD DISP=SHR,DSN=IMSPSA.PROJECT.HD02
//DFSURGU1 DD DISP=(,CATLG),SPACE=(CYL,(156,50)),UNIT=SYSDA SHR,
// DSN=IMSPSA.IMOA.CUSTDB.UNLOAD.CUSTDB
//MSGPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//IHCSYSIN DD *
UNLOAD DBD(CUSTDB) -
PARTNUM(4) -
SELTP(1) -
KEYLIST(2) -

```

```

INDSEL(2) INDSIZE(1024) -
INDPART(1) INDMIN(1024) -
DBREC(5000)

```

/\*

Most generated JCLs also contain an additional step (not shown in our examples) that keeps track of the status of the project in the KSDS we defined (10.3.2, “Define a project” on page 164).

Pressing the End key leads to the Project Checkpoint panel (Figure 10-23), which shows the status of the project. We are asked if we want to start the next phase (called implementation). We enter Y for yes.

```

----- IMS HALDB Conversion Aid -----
Command ==>

Project Checkpoint

Current Environment      IM0A
Current Project         CUST

Phase completed         End Unload
Next Phase              Start Implementation

- Start next phase?
  "Y" Yes
  "N" No

```

Figure 10-23 Project Checkpoint panel

Now a series of panels with generated JCLs is shown to us, which we have to submit. The sequence is:

1. JCL for DELETE.DB
2. JCL for DBDGEN
3. JCL for saving the source (if we had specified SAVE SOURCE as default)
4. JCL for INIT.DB and INIT.PART
5. JCL for deletion and allocation of the data sets
6. JCL for saving the source (if we had specified SAVE SOURCE as default)
7. JCL for initialization
8. JCL for reload
9. JCL for image copy

Not all of the jobs are shown here. The first does a DELETE.DB of CUSTDB, which is still non-HALDB in the RECONS, so all related information is removed from the RECONS. Then the new HALDB DBD is created by DBDGEN. You may

see the source in your save file or by choosing option 7 (other utilities) of the main panel and then selecting option 1 (show DBD source). The next step registers the HALDB in the RECONS (Example 10-2).

*Example 10-2 JCL for INIT commands*

---

```
//DBRC EXEC PGM=IHCZUTIL,COND=(4,LE),REGION=50M
//STEPLIB DD DISP=SHR,DSN=IHC.V2PTF2.SIHCLOAD
// DD DISP=SHR,DSN=IMSPSA.IMS0.SDFSRESL
//RECON1 DD DISP=SHR,DSN=JOUK01.RECON1
//RECON2 DD DISP=SHR,DSN=JOUK01.RECON2
//RECON3 DD DISP=SHR,DSN=JOUK01.RECON3
//DFSRESLB DD DISP=SHR,DSN=IMSPSA.IMS0.SDFSRESL
//IMS DD DISP=SHR,DSN=JOUK01.DBDLIB
//MSGPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//IHCSYSIN DD *
  RUN PGM(IHCZDBRC)
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
  INIT.DB DBD(CUSTDB) TYPHALDB
  INIT.PART DBD(CUSTDB) PART(CUSTDB0) -
    ANCHOR(11) HIBLOCK(13725) -
    FBFF(0) FSPF(0) -
    BLOCKSIZE(8192) -
    DSNPREFIX(IMSPSA.IM0A.CUSTDB.CUSTDB) -
    KEYSTRNG(X'FOF0F0F6F0F3F1F5F0F0')
  INIT.PART DBD(CUSTDB) PART(CUSTDB1) -
    ANCHOR(11) HIBLOCK(13725) -
    FBFF(0) FSPF(0) -
    BLOCKSIZE(8192) -
    DSNPREFIX(IMSPSA.IM0A.CUSTDB.CUSTDB) -
    KEYSTRNG(X'FOF0F1F1F0F5F3F0F0F0')
  INIT.PART DBD(CUSTDB) PART(CUSTDB2) -
    ANCHOR(11) HIBLOCK(13725) -
    FBFF(0) FSPF(0) -
    BLOCKSIZE(8192) -
    DSNPREFIX(IMSPSA.IM0A.CUSTDB.CUSTDB) -
    KEYSTRNG(X'FOF0F1F6F0F8F1F5F0F0')
  INIT.PART DBD(CUSTDB) PART(CUSTDB3) -
    ANCHOR(11) HIBLOCK(13725) -
    FBFF(0) FSPF(0) -
    BLOCKSIZE(8192) -
    DSNPREFIX(IMSPSA.IM0A.CUSTDB.CUSTDB) -
    KEYSTRNG(X'FF')
  INIT.DB DBD(CUSTSI) TYPHALDB
  INIT.PART DBD(CUSTSI) PART(CUSTSI0) -
    DSNPREFIX(IMSPSA.IM0A.CUSTDB.CUSTSI) -
```

KEYSTRNG(X'FF')

---

After registering the database and partitions, the corresponding data sets must be allocated. Example 10.3 on page 155 shows the statements for the data data set and the ILDS data set of partition 1, and the allocation JCL for the secondary index data set. The allocation statements for the other partitions are similar.

*Example 10-3 JCL for allocation (shows only partition 1)*

---

```
DEFINE CLUSTER( -
    NAME(IMSPSA.IMOA.CUSTDB.CUSTDB.L00001) -
    CYLINDERS(1,1) -
    INDEXED KEYS(9,0) -
    RECSZ(50 50) -
    CISZ(4096) -
    FREESPACE(0,0) -
    VOLUMES(TOTIML) -
    REUSE )
ALLOCATE -
    DATASET('IMSPSA.IMOA.CUSTDB.CUSTDB.A00001') -
    NEW CATALOG DSORG(PS) -
    SPACE(160,16) -
    CYLINDERS
.....
.....
DEFINE CLUSTER( -
    NAME(IMSPSA.IMOA.CUSTDB.CUSTSI.A00001) -
    CYLINDERS(79,8) -
    INDEXED KEYS(42,39) -
    RECSZ(82 82) -
    CISZ(4096) -
    FREESPACE(0,0) -
    VOLUMES(TOTIML) -
    REUSE )
```

---

The next step sets the PINIT flags with a CHANGE command and initializes all partitions (Example 10-4) with the DFSUPNT0 utility.

*Example 10-4 Initialization*

---

```
//JOBHCA JOB (999,POK),'LISTINGS',NOTIFY=JOUK01,
//      CLASS=A,MSGCLASS=T,TIME=1439,
//      REGION=0M,MSGLEVEL=(1,1)
//*
//DBRC EXEC PGM=IHCZUTIL,
//      COND=(4,LE),
//      REGION=50M
//STEPLIB DD DISP=SHR,DSN=IHC.V2PTF2.SIHCLOAD
```

```

//          DD DISP=SHR,DSN=IMSPSA.IMS0.SDFSRESL
//RECON1    DD DISP=SHR,DSN=JOUK01.RECON1
//RECON2    DD DISP=SHR,DSN=JOUK01.RECON2
//RECON3    DD DISP=SHR,DSN=JOUK01.RECON3
//DFSRESLB DD DISP=SHR,DSN=IMSPSA.IMS0.SDFSRESL
//IMS       DD DISP=SHR,DSN=JOUK01.DBDLIB
//MSGPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSOUT    DD SYSOUT=*
//IHCSYSIN DD *
RUN PGM(IHCZDBRC)
//SYSPRINT DD SYSOUT=*
//SYSIN     DD *
        CHANGE.DB DBD(CUSTDB0) PINIT
        CHANGE.DB DBD(CUSTDB1) PINIT
        CHANGE.DB DBD(CUSTDB2) PINIT
        CHANGE.DB DBD(CUSTDB3) PINIT
        CHANGE.DB DBD(CUSTSIO) PINIT
//INIT EXEC PGM=DFSUPNT0,
//          COND=(4,LE),
//          REGION=50M
//STEPLIB DD DISP=SHR,DSN=IHC.V2PTF2.SIHCLOAD
//          DD DISP=SHR,DSN=IMSPSA.IMS0.SDFSRESL
//DFSRESLB DD DISP=SHR,DSN=IMSPSA.IMS0.SDFSRESL
//IMS      DD DISP=SHR,DSN=JOUK01.DBDLIB
//RECON1    DD DISP=SHR,DSN=JOUK01.RECON1
//RECON2    DD DISP=SHR,DSN=JOUK01.RECON2
//RECON3    DD DISP=SHR,DSN=JOUK01.RECON3
//DFSVSAMP DD DISP=SHR,
//          DSN=IMSPSA.IMOA.PROCLIB(DFSVMOR)
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
        CUSTDB
        CUSTSI

```

---

After this has completed successfully we may process the reload (Example 10-5), which includes CHANGE commands to turn off the *image copy needed* flags set by the initialization.

#### Example 10-5 Reload

---

```

/*=====
/*
/*          RELOAD PRIMARY DATABASE  CUSTDB
/*
/*=====
//RELD EXEC PGM=IHCHALDB,
//          COND=(4,LE),
//          REGION=60M

```



```
//STEPLIB DD DISP=SHR,DSN=IHC.V2PTF2.SIHCLOAD
//          DD DISP=SHR,DSN=IMSPSA.IMS0.SDFSRESL
//DFSRESLB DD DISP=SHR,DSN=IMSPSA.IMS0.SDFSRESL
//IMS      DD DISP=SHR,DSN=JOUK01.DBDLIB
//RECON1   DD DISP=SHR,DSN=JOUK01.RECON1
//RECON2   DD DISP=SHR,DSN=JOUK01.RECON2
//RECON3   DD DISP=SHR,DSN=JOUK01.RECON3
//DFSURGU1 DD DISP=OLD,
//          DSN=IMSPSA.IM0A.CUSTDB.UNLOAD.CUSTDB
//SYSPRINT DD SYSOUT=*
//IHCSYSIN DD *
RELOAD DBD(CUSTDB) UNLFILE(DFSURGU1)
```

---

The job output says Image copy needed for..., so the next job must be an image copy (DFSUDMP0). We check and submit it. After the image copies of the main database have been performed, we are asked whether we want to do an image copy of our secondary index. If we choose NO, we must have a tool to build the secondary index. We submit the generated image copy JCL and then again we are presented with a checkpoint panel, which asks whether we want to start the next phase: Post Processing.

After entering YES we see the following panel (Figure 10-24). This is just a reminder and lists the tasks that are to be performed by the user.

```
----- IMS HALDB Conversion Aid -----
Command ==>

The conversion project is at its end.
The following activities are still necessary:

1. If you changed the DBD Names
   Define the new DBDs to your online system.
   Change all PSBs to the new DBD name.

2. Run ACBGEnS

3. Change applications that have the PSINDEX DBD as PCB
   if the index has SX (Key increased by 4)

4. Activate everything (Online Change)

N Say Y if all has been done
   Say N if not (and you need this as a reminder)
```

*Figure 10-24 Post-processing*

If all shown tasks are performed, the HALDB database may be used and this conversion project is finished. A last panel asks if the project data set should be kept, reused, or deleted.

# 10.4 Additional functions

The HALDB Conversion and Maintenance Aid Version 2 for z/OS is a successor product to the IMS HALDB Conversion Aid Version 1. As the name implies, the new functions are primarily for the maintenance of databases after they have been converted to HALDB. They include the capability to split or consolidate HALDB partitions (choose option 5 of the main panel Figure 10-1 on page 156). Other functions include back up, recovery, unload, and reload for individual partitions, as well as INDEX pointer healing (choose option 6 of the main panel). For example, our current HALDB NORDDDB with a partition exit routine has four partitions. Now we think that perhaps six partitions would be better, but which keystings should we use? We choose option 5, make our definitions, enter the new keystings, and submit the JCL generated. Example 10-6 shows the analysis based on the new partitions. We can see that we managed to select quite a good keysting, since all the partitions are almost equal in size. Only the last partition has less segments, but this change may be done for the preparation for the new data to be added in the last partition. You are asked if you want to continue with an unload and if you enter YES, the unload JCL is generated.

Example 10-6 Output for testing new boundaries

Partitioning for DBD NORDDDB											
Part	Roots	%	All Segments	%	Prefix Bytes	%	Data Bytes	%	Total Bytes	%	
Total	193,078	100.00	193,078	100.00	3,475,404	100.00	2,703,092	100.00	6,178,496	100.00	
1	36,960 Key: 0004 FOFOFOF4	19.14	36,960	19.14	665,280	19.14	517,440	19.14	1,182,720	19.14	
2	36,582 Key: 0008 FOFOFOF8	18.94	36,582	18.94	658,476	18.94	512,148	18.94	1,170,624	18.94	
3	37,021 Key: 0012 FOFOF1F2	19.17	37,021	19.17	666,378	19.17	518,294	19.17	1,184,672	19.17	
4	36,901 Key: 0016 FOFOF1F6	19.11	36,901	19.11	664,218	19.11	516,614	19.11	1,180,832	19.11	
5	36,614 Key: 0020 FOFOF2F0	18.96	36,614	18.96	659,052	18.96	512,596	18.96	1,171,648	18.96	
6	9,000 Key: yyy FFFFFFF	4.66	9,000	4.66	162,000	4.66	126,000	4.66	288,000	4.66	

Then a job with CHANGE commands for the old partitions and INIT commands for the new partitions is created (Example 10-7 on page 177).

### Example 10-7 Defining the new partitions

---

```
//DBRC EXEC PGM=IHCZUTIL,
// COND=(4,LE),
// REGION=50M
//STEPLIB DD DISP=SHR,DSN=IHC.V2PTF2.SIHCLOAD
// DD DISP=SHR,DSN=IMSPSA.IMS0.SDFSRESL
//RECON1 DD DISP=SHR,DSN=JOUK01.RECON1
//RECON2 DD DISP=SHR,DSN=JOUK01.RECON2
//RECON3 DD DISP=SHR,DSN=JOUK01.RECON3
//DFSRESLB DD DISP=SHR,DSN=IMSPSA.IMS0.SDFSRESL
//IMS DD DISP=SHR,DSN=JOUK01.DBDLIB1
//MSGPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//IHCSYSIN DD *
RUN PGM(IHCZDBRC)
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
CHANGE.PART DBD(NORDDB) PART(NORDDB0) -
KEYSTRNG(X'F0F0F15EF0F0F45EF0F0F4') -
DSNPREFX(IMSPSA.IM0A.NORDDB.NORDDB)
CHANGE.PART DBD(NORDDB) PART(NORDDB1) -
KEYSTRNG(X'F0F0F15EF0F0F45EF0F0F8') -
DSNPREFX(IMSPSA.IM0A.NORDDB.NORDDB)
CHANGE.PART DBD(NORDDB) PART(NORDDB2) -
KEYSTRNG(X'F0F0F15EF0F0F45EF0F0F1F2') -
DSNPREFX(IMSPSA.IM0A.NORDDB.NORDDB)
CHANGE.PART DBD(NORDDB) PART(NORDDB3) -
KEYSTRNG(X'F0F0F15EF0F0F45EF0F0F1F6') -
DSNPREFX(IMSPSA.IM0A.NORDDB.NORDDB)
CHANGE.PART DBD(NORDDB) PART(NORDDB4) -
KEYSTRNG(X'F0F0F15EF0F0F45EF0F0F2F0') -
DSNPREFX(IMSPSA.IM0A.NORDDB.NORDDB)
CHANGE.PART DBD(NORDDB) PART(NORDDB5) -
KEYSTRNG(X'F0F0F15EF0F0F45EFF') -
DSNPREFX(IMSPSA.IM0A.NORDDB.NORDDB)
```

---

As already shown in the conversion example, jobs follow that allocate the data sets, initialize the partitions, perform the reload, and all image copies needed.

Option 7, other utilities, provides such functions as the Partition Selection Exit Test utility, OSAM Multi-volume, and the Check "BA" Status Code Table. Most are self-explanatory, so only a few words about the last function are added here. You may enable BA status codes for applications that do not issue the INIT STATUS call. The tool provides an exit. You determine for which programs this call should be created and you specify the databases for which the programs should wait if

they get a BA status code. For the mechanism to retry the call after receiving a BA, two parameters may be specified, wait time and retry count. The retry is made after wait time has exceeded, and the count tells how many times a retry should be attempted. This function only applies to BMP and MPP regions and for GU or GHU calls.

Option 8 (Figure 10-25) allows you to perform the following DBRC functions, where the first three are LIST commands.

```

----- IMS HALDB Conversion Aid -----
Command ==>

                                DBRC Support

    _ Select Function                                Current Settings
                                                    Environment: im0a

    1 Show Partition Definition
    2 Show Partitions with Authorization Exceptions
    3 Show all Subsystems

    4 Resolve Authorization Exceptions
    5 Clone Partition Definition
    6 Copy Partitions to a different RECON

```

Figure 10-25 DBRC Support

If you want the definitions of your HALDB partitions copied into another RECON environment, choose Clone Partition Definition. This does not transfer the content of your database. Specifying your DBDLIB and the new RECONS on the given panels leads you to the following panel (Figure 10-26).

```

----- IMS HALDB Conversion Aid -----
Command ==>

Options for DBRC cloning of DBD  NORDDDB

Enter new dataset name prefix
imspsa.test

Add DBD name to dataset name prefix
1 1 Yes
2 No

Include secondary indexes
1 1 Yes
2 No

```

Figure 10-26 Clone partition definitions

Pressing Enter will generate the needed JCL (Example 10-8 on page 179).

### Example 10-8 Clone partition definitions

```
/*-----  
/*          CREATE DBRC COMMANDS  
/*-----  
//CRE      EXEC PGM=IHCHALDB,  
//          REGION=60M  
//STEPLIB DD DISP=SHR,DSN=IHC.V2PTF2.SIHCLOAD  
//          DD DISP=SHR,DSN=IMSPSA.IMS0.SDFSRESL  
//DFSRESLB DD DISP=SHR,DSN=IMSPSA.IMS0.SDFSRESL  
//IMS      DD DISP=SHR,DSN=JOUK01.DBDLIB1  
//RECON1   DD DISP=SHR,DSN=JOUK01.RECON1  
//RECON2   DD DISP=SHR,DSN=JOUK01.RECON2  
//RECON3   DD DISP=SHR,DSN=JOUK01.RECON3  
//MSGPRINT DD SYSOUT=*  
//SYSUDUMP DD SYSOUT=*  
//SYSOUT   DD SYSOUT=*  
//IHCSYSIN DD *  
          COPYDBRC DBD(NORDDDB) -  
          DSNPREF(IMSPSA.TEST) -  
          DSNDBD(YES) -  
          INCLIND(YES) -  
          TODD(DBRCOUT)  
/*  
//DBRCOUT DD DISP=(,PASS),SPACE=(TRK,(1,1)),UNIT=SYSALLDA  
/*-----  
/*          APPLY TO OTHER DBRC  
/*-----  
//DBRC      EXEC PGM=DSPURX00,REGION=50M,COND=(4,LE)  
//STEPLIB DD DISP=SHR,DSN=IMSPSA.IMS0.SDFSRESL  
//RECON1   DD DISP=SHR,DSN=JOUK01.NEW.RECON1  
//RECON2   DD DISP=SHR,DSN=JOUK01.NEW.RECON2  
//RECON3   DD DISP=SHR,DSN=JOUK01.NEW.RECON3  
//DFSRESLB DD DISP=SHR,DSN=IMSPSA.IMS0.SDFSRESL  
//IMS      DD DISP=SHR,DSN=JOUK01.DBDLIB1  
//SYSPRINT DD SYSOUT=*  
//SYSIN    DD DISP=(OLD,DELETE),DSN=*.CRE.DBRCOUT
```

The HALDB Conversion and Maintenance Aid helps you implement the database in another RECON environment if you choose Copy Partitions to a different RECON. This will create cloned partition definitions in another RECON, and also transport full image copies to these RECON data sets. It will resolve the problems with partition IDs as they are described in 11.7.2, “Using image copies” on page 207. Then you have to perform a timestamp recovery with GENJCL.RECOV and test your database.

Archived



## Part 3

# Application considerations

In this part we discuss potential application changes and opportunities with HALDB. We describe some application programming changes that you may need to make, and other changes that you may want to implement.

We discuss the differences that HALDB creates for test environments and the copying of databases.

Archived



## Application considerations

You may not need to change any of your application programs when you migrate databases to HALDB, but there may be exceptions. In this chapter we explain the exceptions and how to handle them. Exceptions include the initial load of logically related databases and the processing of secondary indexes as databases.

We also explain why you might want to change applications to take advantage of some HALDB capabilities. These capabilities include processing partitions in parallel, processing individual partitions, and handling unavailable partitions.

We explain how to create test environments for HALDB and how to copy HALDB databases between IMS environments.

## 11.1 Initial loads

Application programs that initially load databases generally do not have to be changed when the databases are migrated to HALDB. On the other hand, you may find situations in which it is required or advantageous to modify existing initial load programs.

The only required change is for application programs that initially load logical children in logically related databases. This is not allowed with HALDB. If you have an initial load application program that inserts logical child segments, it must be modified. We describe these modifications in “Initial loads of databases with logical relationships” on page 188.

### 11.1.1 PHDAM considerations

Initially loads of PHDAM databases may insert root segments in any order. This is also true for HDAM databases. Any program that was used for initial loads of HDAM databases may be used without modification to load PHDAM databases.

Many installations load HDAM databases in RAP sequence. This loads the records in physical sequence in the database. This is done by sorting the input records into the order that the randomization routine creates. To do the same thing with PHDAM, one would have to sort the records within each partition. With PHDAM, randomization is done within a partition, not across partitions. That is, the partition is selected first and then randomization is done. Each partition could have different randomization parameters. In some cases, the techniques used with HDAM might not work for PHDAM. For example, the use of different randomization routines for different partitions would prevent a general sort for the entire database. The IBM High Performance Load (HP Load) product includes a Physical Sequential Sort for Reload (PSSR) program. PSSR may be used to sort records in RAP sequence. It includes HALDB support to sort records for a partition, a set of partitions, or all partitions in a database.

### 11.1.2 PHIDAM considerations

Initial loads of PHIDAM databases must insert root segments in key sequence within a partition. They do not have to be in key sequence when crossing partition boundaries. Figure 11-1 on page 185 illustrates this. The figure shows a PHIDAM database that uses a partition selection exit routine. Roots are in key sequence within the partitions. They are not in key sequence when crossing partition boundaries. Two possible load sequences are shown. The first shows loading the records in key sequence within each partition. The second shows loading the records in key sequence across the entire database. Other

sequences would also be valid as long as they load records in a partition in key sequence.

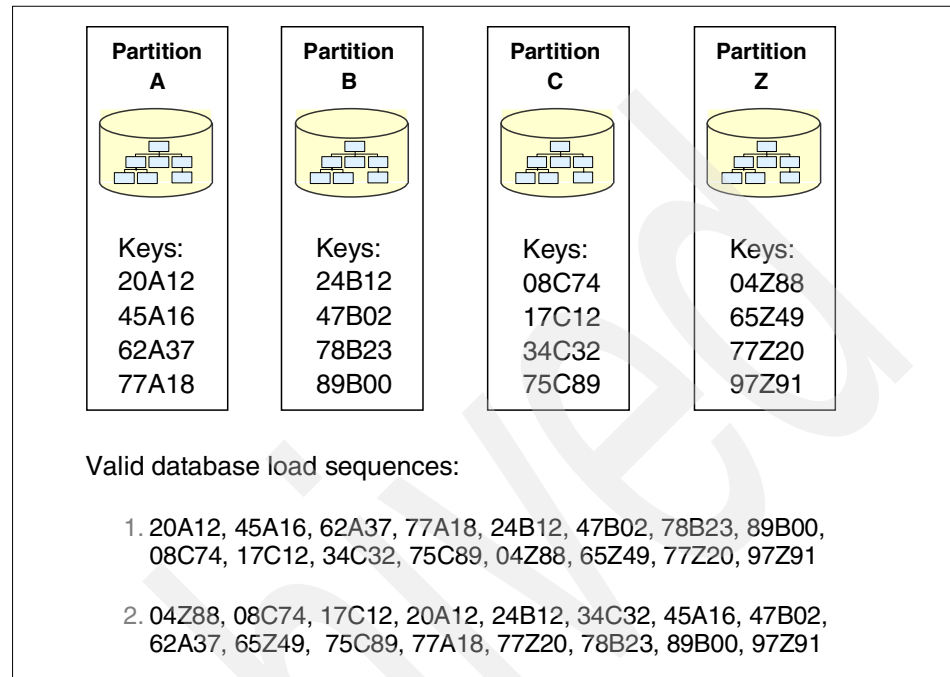


Figure 11-1 Loading sequences for a PHIDAM database

### 11.1.3 Loading partitions in parallel

You may want to load partitions in parallel. This should reduce the time to load a HALDB. You may use separate concurrently executing jobs to do this.

You may be able to load partitions in parallel by making only minor changes to the load process you use with a non-HALDB database. Figure 11-2 on page 186 illustrates a situation where the original load program does not have to be modified. Its input file is split into multiple files. There is a separate file for each partition's key range. The original program is executed multiple times. Each execution reads the input file for its partition's key range.

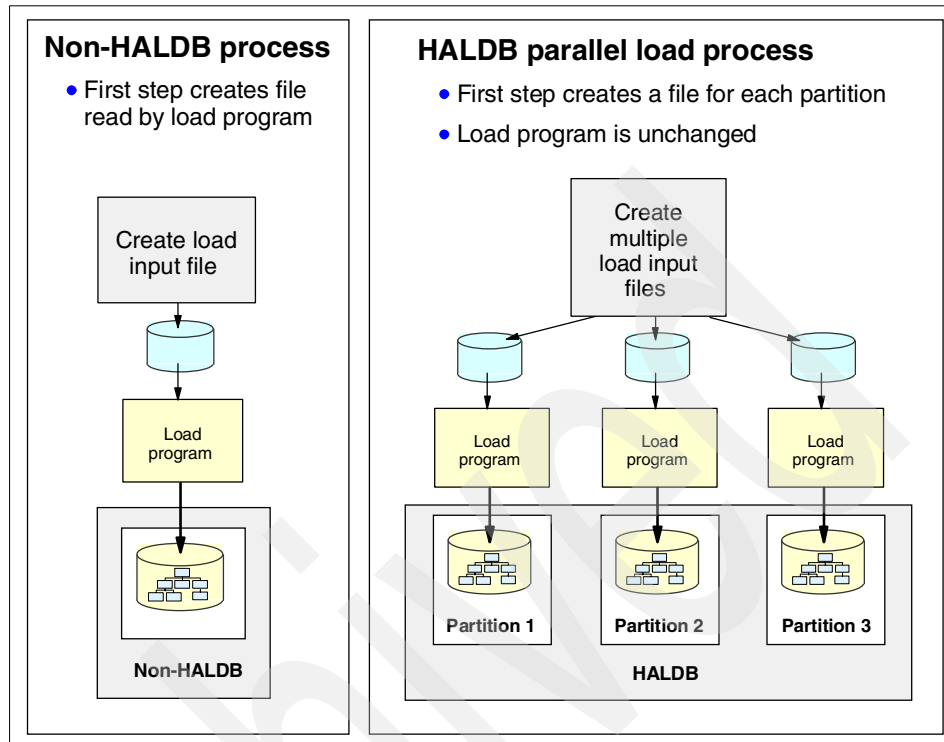


Figure 11-2 Converting a load process to parallel loads

Each partition may be loaded only once. When you load partitions in parallel, two jobs cannot load records in the same partition. An attempt to load more records in a partitions that has already been loaded will result in the original records being lost.

#### 11.1.4 Initial loads of databases with secondary indexes

The initial load of a database with secondary indexes also creates the secondary indexes. We explain an exception to this in the next section, “BLDSNDX parameter” on page 187. A load of a non-HALDB database with secondary indexes requires utility steps to create the indexes. These steps are not required with HALDB databases.

When source segments are loaded in HALDB databases, their secondary index entries are inserted in the secondary index partitions. The buffer pools used by the load program must include buffers for the secondary indexes. The secondary index partitions must have been initialized previously.

**Restriction:** Partition initialization turns the *image copy needed* flags on for data sets in the partitions.

When we wrote this book, initial loads of indexed databases failed if the image copy needed flags were on for secondary index partition data sets. IBM plans to change this with maintenance to IMS Version 8. The maintenance will allow authorizations for initial loads when the image copy needed flags are on for secondary index data sets.

Initial loads do not create entries in the ILDS. They are not needed because any secondary index entries will have accurate RBA pointers in their extended pointer sets (EPSs).

### **BLDSNDX parameter**

APARs PQ55840 for IMS Version 7 and PQ56463 for IMS Version 8 added a capability to eliminate the building of secondary indexes by initial load programs. This is invoked by including a BLDSNDX parameter on the OPTIONS statement in the DFSVSAMP data set. Example shows a use of this statement.

#### *Example 11-1 BLDSNDX statement in DFSVSAMP*

---

```
DFSVSAMP DD *  
OPTIONS, BLDSNDX=NO  
VSRBF=4096,500  
IOBF=(8192,200)  
/*
```

---

If BLDSNDX=YES is specified or if a BLDSNDX parameter is not included, secondary index entries are created by the initial load.

If you include a BLDSNDX=NO parameter, you must build your secondary index by other means. You can use the IMS Index Builder tool to do this.

When you use BLDSNDX=NO, the secondary index partitions are not authorized and their data sets are not allocated.

The use of BLDSNDX=NO and the IMS Index Builder may shorten the load and index creation process. When BLDSNDX=NO is not used, inserts in secondary indexes are random. This could lengthen the load process considerably. IMS Index Builder reads the loaded database, creates index entries without writing them to the secondary indexes, sorts the entries in secondary index key sequence, and then writes them to the secondary indexes in a sequential process. When you have many secondary index entries this process may require less elapsed time.

### 11.1.5 Initial loads of databases with logical relationships

The process for initially loading databases with logical relationships differs with HALDB. When you load logical children in a non-HALDB database, your load programs create work files. These work files are processed by the Prefix Resolution and Prefix Update utilities. They create the appropriate pointers and counters in your logically related databases. HALDB does not use work files or the Prefix Resolution and Prefix Update utilities. Logical relationships cannot be created as part of the load process. This means that load programs cannot insert logical child segments.

If you have a database that you initially load as part of your normal operations and it contains logical children, you will have to modify your load process. The logical children must be inserted in the database in a different job or step. This program must use PROCPT=I or A, not PROCPT=L.

Figure 11-3 illustrates a logical relationship. Logical child LC2 in database DBX points to logical parent LP2 in database DBY. Logical child LC1 in database DBY points to logical parent LP1 in database DBX. LC2 and LC1 are paired logical children.

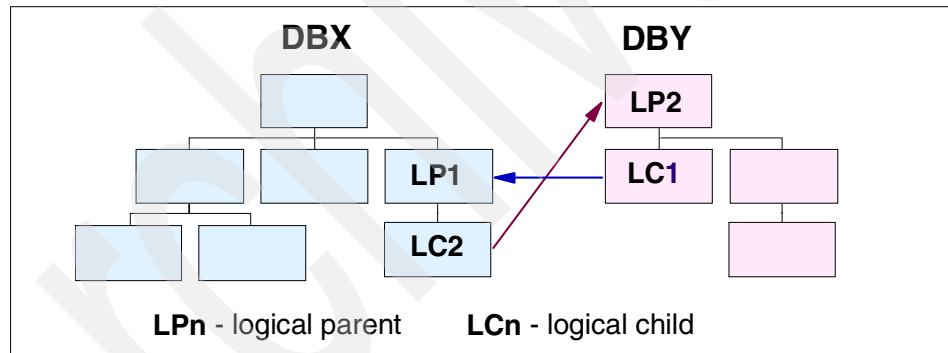


Figure 11-3 Logical children and logical parents

#### Load steps for non-HALDB logically related databases

The steps for loading the databases in Figure 11-3, if they are non-HALDB, are:

1. Execute the Prereorganization utility.
2. Execute the initial load application program for DBX. This includes the loading of logical children LC2. The program creates a work file.
3. Execute the initial load application program for DBY. This includes the loading of logical children LC1. The program creates a work file.
4. Execute the Prefix Resolution utility. This utility sorts the work file records.

5. Execute the Prefix Update utility. This utility reads the output of the Prefix Resolution utility and updates the pointers and counters for the logical relationships.

### **Load steps for HALDB logically related databases**

The steps for loading the databases in Figure 11-3 on page 188 (if they are HALDBs) do not include the Prefix Resolution and Prefix Update utility steps. The Prereorganization utility is used, but it has a different purpose. It is used to initialize the partitions in the databases. Since logical children cannot be loaded, they must be added by an update (PROCOPT=I or A) program. The steps are:

1. Execute the Prereorganization utility to initialize the partitions in DBX.
2. Execute the Prereorganization utility to initialize the partitions in DBY.
3. Execute the initial load application program for DBX. Do not load logical children LC2. The program does not create a work file.
4. Execute the initial load application program for DBY. Do not load logical children LC1. The program does not create a work file.
5. Execute an update application program for a logical database. Insert either logical children LC1 or LC2, but not both. The insertion of either of the paired segments will create its paired logical child.

Steps 3 and 4 can use modified versions of the application programs used in steps 2 and 3 for the non-HALDB process. The modifications would eliminate the inserts of the logical children. The modifications for one of these programs should also save the information about the logical children. This information is needed by the update program used in step 5. For example, the modification of the program that loads DBX could write a file with a record for each LC2 segment. The record would include the concatenated key and the data for the segment. Then the program in step 5 could read this file and insert the LC2 segments using a PCB with PROCOPT=I.

## **11.2 Processing partitions in parallel**

You may find it advantageous to have different instances of your programs process the partitions of a database in parallel. They could reduce the elapsed time required for the processing. The following sections discuss how this may be done with HALDB.

### **11.2.1 DBRC authorization of partitions**

DBRC authorization applies to HALDB partitions, not the entire database. Different partitions of the same database may be authorized to different

subsystems concurrently. For example, two batch jobs may be updating different partitions at the same time without the use of the IRLM. This is not possible with non-HALDB full-function databases. Authorization for non-HALDB full function applies to the entire database. HALDB authorization of partitions is similar to the authorization of areas for Fast Path DEDBs. Both HALDB partitions and DEDB areas are authorized independently.

### **11.2.2 Parallel processing within an online system**

You can easily process different partitions in parallel within an online system. Multiple dependent regions may process records in the same or in different partitions. Data sharing is not required.

### **11.2.3 Parallel processing with block level data sharing**

If you use block level data sharing, you can easily process different partitions in parallel with multiple subsystems. The subsystems may be online systems or batch jobs. Block level data sharing requires the IRLM and the registration of databases at SHARELVL=2 or 3.

### **11.2.4 Parallel processing without block level data sharing**

You may be able to process different partitions in parallel from different subsystems without using block level data sharing, but there are some restrictions. A partition may be authorized concurrently to multiple subsystems, such as multiple batch jobs, if the authorizations are only for reads. If any subsystem is authorized to update a partition, no other subsystem can have update or read with integrity authorization. Other subsystems could have read without integrity (ACCESS=RO or PROCOPT=GO) authorizations. This restriction may not be serious if each program updates only one partition. Different programs could be processing different partitions simultaneously. On the other hand, if each program must also update another database, it is less likely that you can restrict each program to a different partition of the other database.

Figure 11-4 on page 191 illustrates a situation where parallel processing cannot be done. Programs that update different partitions of the target database update all of the partitions of the secondary index. For example, an insert of a target database record with a key of CML might create a secondary index with a key of 632. An insert of a record with a key of RZL might create a secondary index with a key of 980. The target database records are in different partitions but their secondary index entries are in the same partition of the secondary index. Similar situations exist with logical relationships.



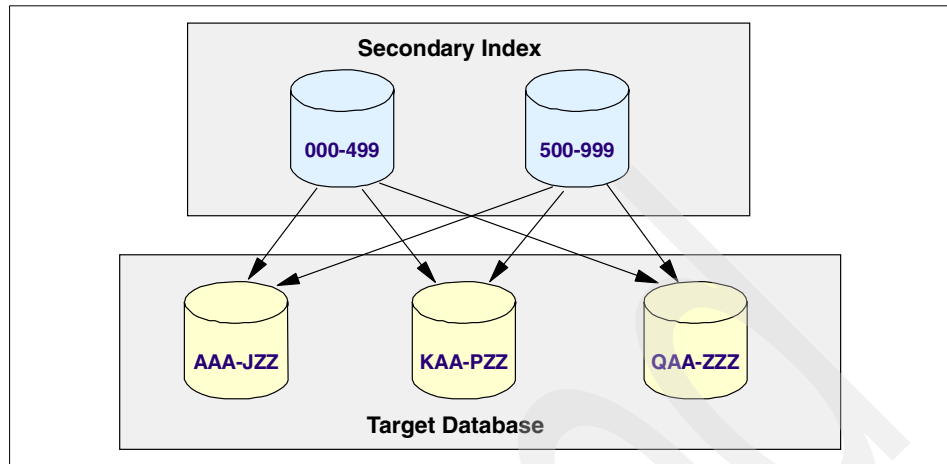


Figure 11-4 Database partitions and secondary index partitions

In some cases the use of partition selection exit routines can make parallel processing of partitions with secondary indexes or logical relationships feasible. Figure 11-5 illustrates an example of this. The keys of the target database and the secondary index have some common data. The third byte of the target database key contains a one-byte code that also appears in the second byte of the secondary index key. The valid values for code are A, B, and C. You can write partition selection exit routines that assign records to partitions based only on this code. This will cause all secondary index entries pointing to the partition of the target database to reside in the corresponding partition of the secondary index.

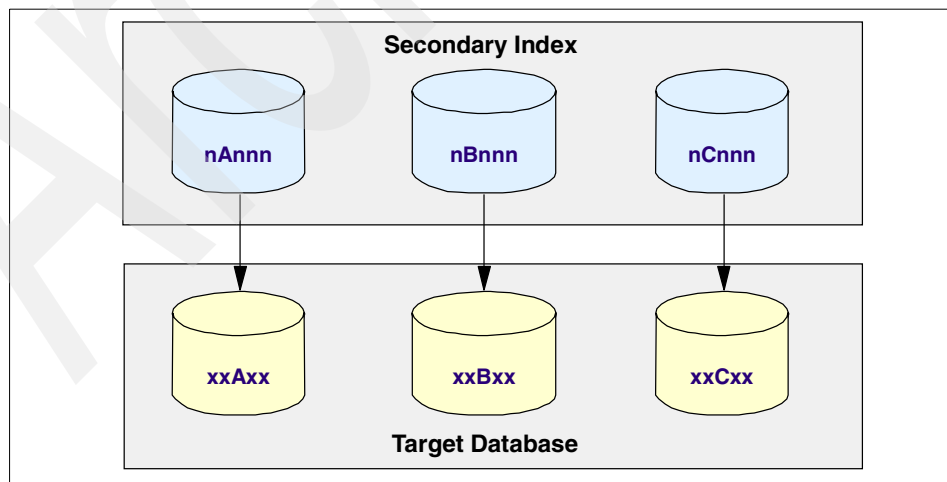


Figure 11-5 Alignment of secondary index and target database partitions

## 11.2.5 Program modifications for parallel processing

If you want your application programs to process partitions in parallel, you will probably need to modify the existing programs. The following changes may be required:

- ▶ Restriction of the program to one or a set of partitions
- ▶ Splitting of the inputs to feed multiple programs
- ▶ Consolidation of the outputs of multiple programs
- ▶ Handling the inability to access a partition

You will need to restrict your program to one partition or a set of partitions. This can be done easily with the HALDB control statement. This statement limits a PCB to one partition of a database. The specification of this statement and its use are described in 11.3, “Restricting a PCB to one partition” on page 193. If you wish to use a PCB to access multiple partitions, you cannot use the HALDB control statement. Instead, your program must understand which records reside in these partitions and access only those records. Information about which records are in which partitions is not available from control blocks or calls that the application may use. You must devise a scheme to provide this information to your programs. We will not discuss this further. It is much easier to design systems so that application programs either have access to all partitions or are limited to only one partition.

Your current program may read an input file. If you wish to modify it so that multiple instances of the program process different partitions, you need to split the input file. Each instance should read a different file. Similarly, if your current program reads a database to get its input, different instances of the program need to read different parts of the database.

Your current program may produce output files or reports. If you create multiple instances of the program, you will need to consolidate output files or reports from the separate programs.

You may need to modify your program so that it can handle situations when a partition is not available. An example of this requirement is a program that typically reads records from a partition, but occasionally must access a record out of this range. This is explained in Appendix 11.4, “Handling unavailable partitions” on page 197.

Application programs that process a database sequentially are typically the easiest to restrict to one partition. With key range partitioning these programs can begin with the lowest key in the partition and terminate when they reach the end of the partition. This is easily done with the HALDB control statement, which is explained in 11.3, “Restricting a PCB to one partition” on page 193. If the HALDB control statement is not used, the program must know the lowest key in

the partition so that it may begin with it. It must also know the highest possible key in the partition so that it can determine when to terminate. If the highest possible key does not exist, the program will attempt to cross to the next partition. The program may not have authorization to process this partition. This could occur with a batch program not using block level data sharing. Another program could have authorization for the next partition. This problem is avoided when the HALDB control statement is used.

Application programs that process a database randomly may be more difficult to restrict to one partition. Often these programs decide which records to access based on data they read from other sources. You may need to modify how these programs get their data from the other sources. For example, you may need to read the other sources first and then create separate input files. Each file would contain data for one partition and would be read by one application program.

### 11.3 Restricting a PCB to one partition

The HALDB control statement may be used to restrict a batch program (DLI, DBB, BMP, or JBP) to one partition of a HALDB database. This capability was introduced by APARs PQ57313 for IMS Version 7 and PQ58600 for IMS Version 8. The control statement may be used with DLI, DBB, BMP, and JBP regions. It may not be used with MPP, IFP, or JMP regions, or with DBCTL or ODBA threads.

The control statement is used with the DFSHALDB DD statement. It restricts a PCB to one partition. It specifies the PCB and the partition.

```
//DFSHALDB DD *  
HALDB PCB=(nnn|dddddddd,ppppppp)
```

nnn - DBPCB number  
dddddddd - DBPCB label or name  
ppppppp - partition name

*Figure 11-6 HALDB control statement*

The format of the statement is shown in Figure 11-6. You may specify either the PCB label, name, or number. The capability to specify the PCB label or name was added by APARs PQ65486 for IMS Version 7 and PQ65489 for IMS Version

8. The PCB label is the label on the PCB statement in the PSB. The PCB name is the name of the database specified on the PCB statement. The PCB number is the relative PCB number among the database PCBs in the PSB. Our examples show the use of the PCB number, not the PCB label or name. You may have multiple HALDB control statements. Each statement would restrict a different PCB. Up to ten statements may be specified.

The HALDB control statement may be used with PHDAM, PHIDAM, and PSINDEX databases. It restricts the PCB to the partition of the database or secondary index specified on the PCB statement in the PSB. If a PROCSEQ parameter is specified on the PCB, the statement restricts the PCB to a partition of the secondary index. Otherwise, it restricts the PCB to a partition of the database specified on the DBD parameter.

When you include a DFSHALDB DD statement you should also include a SYSHALDB DD statement. SYSHALDB is used for information and error messages about the HALDB control statement. You may use the SYSOUT parameter on the SYSHALDB DD statement.

### **11.3.1 Processing a partition sequentially**

You can easily process a partition sequentially by specifying the HALDB control statement. When the control statement is used, a request for the first segment in a database returns the first segment in the partition. For example, an unqualified GN with no previous position would return the first segment in the partition. A get next call that reaches the end of the partition returns a GB status code. Without the control statement, the GB status code indicates that the end of the database has been reached. These characteristics make it easy to use a program that otherwise processes the entire database to process only a partition. Other modifications may be required. For example, the program may produce a report or output file. Multiple instances of the program that processed different partitions would produce multiple reports or files. You would need to consolidate them.

### **11.3.2 Requesting segments outside of the partition**

A call that requests a segment in another partition returns an FM status code. For example, a GU call for key=2,546,773 with a control statement restricting the PCB to a partition with keys between 1,000,000 and 1,999,999 would return the FM status code. Your existing programs probably do not expect an FM status code. If a program might request a record outside of the partition, it should be modified to handle the FM status code before you use the HALDB control statement.

### 11.3.3 Secondary index and logical relationship considerations

You should be careful when using the HALDB control statement with secondary indexes and logical relationships. The control statement specifies a partition of the database or secondary index that is used to enter the database. When you specify the PROCSEQ parameter on the PCB, you must specify a partition of the secondary index. When you do not specify PROCSEQ, you must specify a partition of the indexed database.

Figure 11-7 shows the use of the control statement with PROCSEQ on the PCB. The control statement restricts the PCB to one partition of the database specified in the control statement. It does not restrict the use of partitions in the indexed database. In Figure 11-7 the statement restricts the third PCB to partition AAA in the secondary index. It does not restrict the partitions accessed in the indexed database. Partition AAA has index entries pointing to each of the partitions in the indexed database.

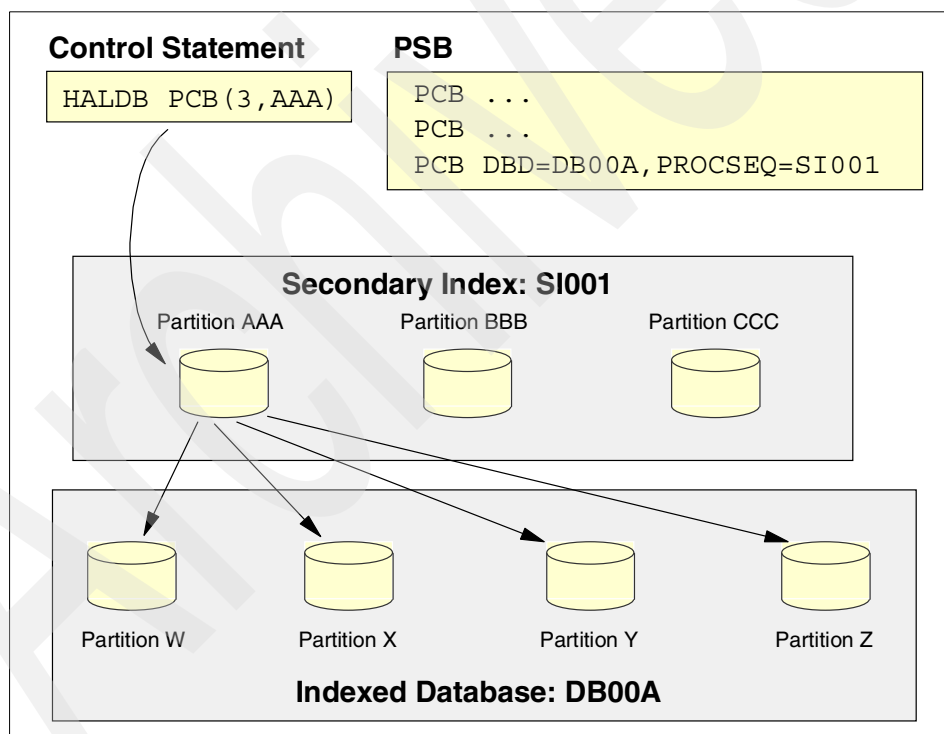


Figure 11-7 Using a HALDB control statement with PROCSEQ on the PCB

Figure 11-8 on page 196 shows the use of the control statement without a PROCSEQ on the PCB. The control statement restricts the PCB to one partition of the indexed database. It does not restrict the use of partitions in the secondary

index. In the figure the statement restricts the second PCB to partition X of the indexed database. Partition X has index entries in all of the partitions of the secondary index. An update of a source segment in partition X of database DB00A could cause an update in any partition of the secondary index. Similarly, a call qualified on a secondary index field could use any of the partitions of the secondary index.

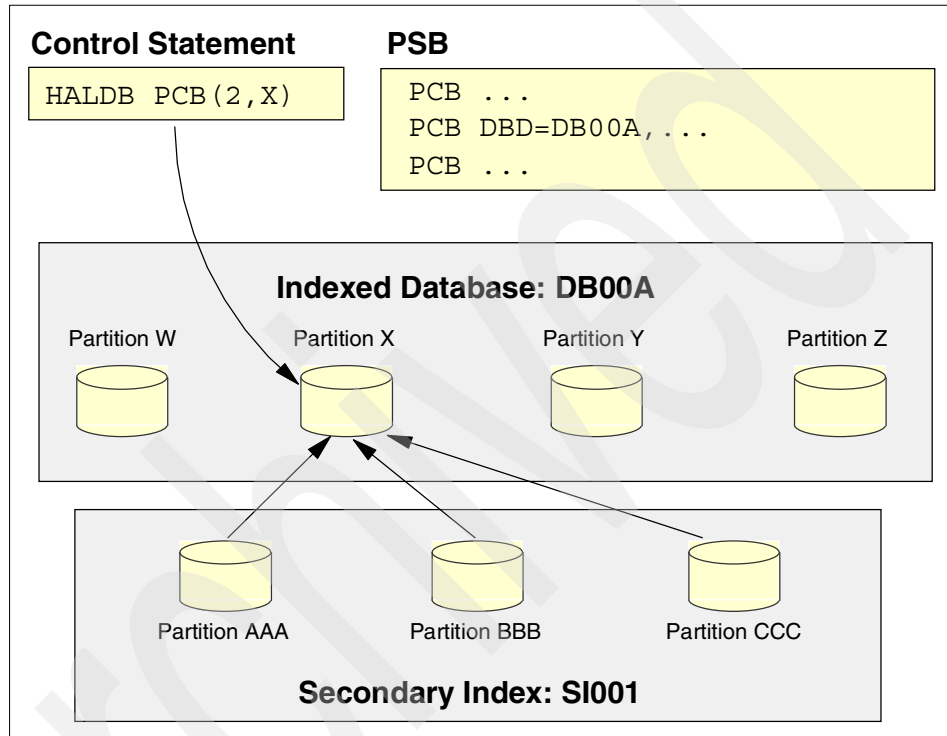


Figure 11-8 Using a HALDB control statement without PROCSEQ on the PCB

Figure 11-9 on page 197 shows the use of the control statement with logical relationships. The control statement restricts the PCB to one partition of the physical database containing the root segment in logical database LDBABC. This physical database is DB0L1. It does not restrict the use of partitions in the physical database DB0L2. Partition XAA in database DB0L1 has logical relationships with records in all of the partitions of database DB0L2.

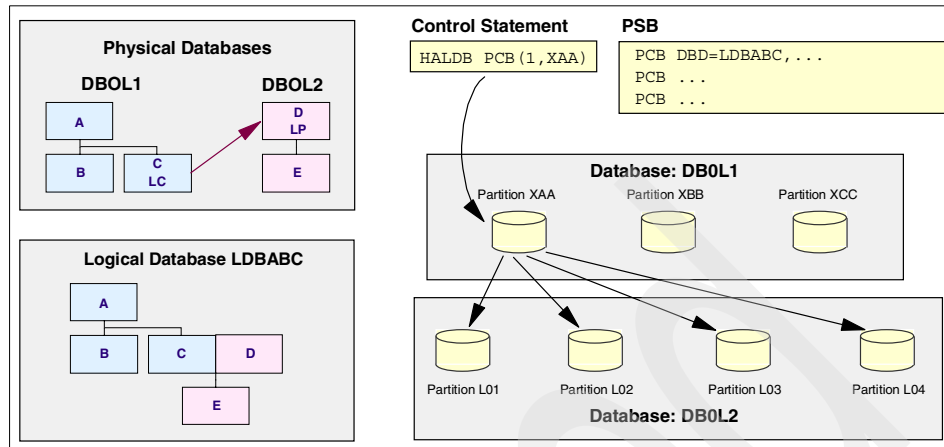


Figure 11-9 Using a HALDB control statement with logical relationships

## 11.4 Handling unavailable partitions

A partition may not be available to an application program. This is likely to happen if you deallocate individual partitions with /DBR commands or if you are using multiple batch jobs without block level data sharing. A partition is unavailable if a /DBR, /STOP, or /LOCK command has been issued for it. DBRC flags such as recovery needed or prohibit further authorization make a partition unavailable. A partition is unavailable for update if a /DBD command has been issued for it or if a /STA DB command with ACCESS=RD or RO has been issued for it. The DBRC *image copy needed* flag also makes a partition unavailable for update. The use of batch jobs without block level data sharing may make partitions unavailable. For example, your program may attempt to process records in another partition. As explained in 11.2.4, "Parallel processing without block level data sharing" on page 190, DBRC authorization processing may prevent access to the other partition. If a partition may be unavailable or unavailable for update and your program might attempt to access the partition, your program must handle the unavailable partition condition.

Attempts to access a partition that is not available result in an unavailable data condition. If your program has issued an INIT STATUS GROUPx call or EXEC DLI ACCEPT STATUSGROUP command, an attempt to access an unavailable partition returns a BA status code. If you have not issued this INIT call or this ACCEPT command, your program will abend. A U3303 abend code is produced. Obviously, you should include the call or command in your program if it might attempt to access an unavailable partition.

If your program receives a BA status code due to an unavailable partition, a GN call using the same PCB will move to the next available partition. If there are no more available partitions, the GN call will return a GB status code indicating that the end of the database has been reached. Programs that process databases sequentially and that may encounter unavailable partitions, may need to issue the GN call after receiving a BA. They also should be designed to understand that they have not processed all of the data in the database.

There are two reasons for the unavailable data condition. One is an unavailable partition. The other is a lock reject. A lock reject can occur when a database record is not available due the failure of a data-sharing subsystem holding a lock on the record. It can also occur due to the failure of a CICS® system or ODBA thread with an in-doubt unit of work holding a lock on the database record. ODBA threads include DB2-stored procedures. If your program receives a BA status code, it could be due to either an unavailable partition or a lock reject. It does not receive information indicating which is the cause.

### **11.4.1 Database PCB status code priming**

When a program is scheduled the database PCB status code field indicates the status of the database. This is the same for HALDB and non-HALDB databases. The PCB does not include information about the status of partitions in the database. A “blank blank” status code indicates that the database is available. NA indicates that the database is not available. NU indicates that the database is available, but not available for update. The INIT DBQUERY call may be used to get the same information in the status code fields. That is, it restores these fields to the settings that they had when the application was scheduled.

Since there is no information about the status of individual partitions, a “blank blank” could indicate that the database is available, but some or all of its partitions might not be available. The program is not given a way to determine the status of partitions.

## **11.5 Processing secondary indexes as databases**

If you have an application program that processes a secondary index as a database, you may have to adjust the program and its PSB. Adjustments are required when the secondary index uses a /SX field to create unique keys. Adjustments may be required when symbolic pointing is used.



### 11.5.1 Secondary indexes with /SX fields

The /SX field is four bytes for non-HALDB databases but eight bytes with HALDB. This increases the size of the secondary index's key field and changes the offset to any following fields. The three changes that may be required are:

- ▶ The KEYLEN parameter on the PCB statement for the secondary index in the PSB by must be at least the size of the VSAM key of the secondary index. It should be increased by four bytes.
- ▶ The I/O area used by the secondary index segment must be increased by four bytes.
- ▶ The offset to any duplicate data fields or user fields that the application program processes must be adjusted by four bytes.

Figure 11-10 shows the change to the program I/O area when a secondary index with a /SX field is migrated to HALDB.

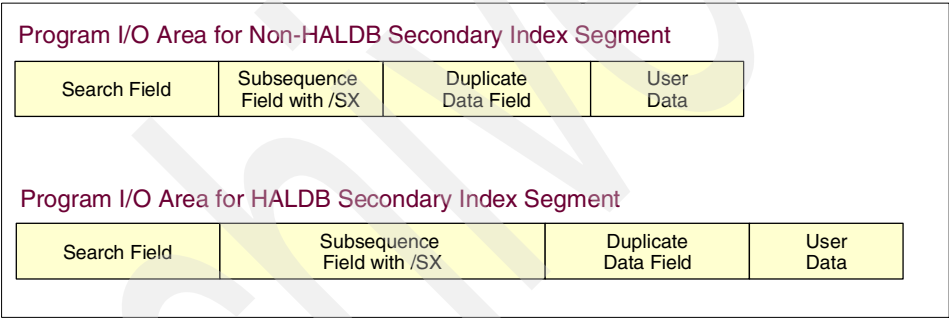


Figure 11-10 Program I/O area for secondary index with /SX field

#### Determining which programs are affected by /SX fields

You should examine your PSBs to determine if you need to make adjustments. You may use the following steps.

1. See if you have a PSB with a PCB statement on which the DBDNAME field specifies a secondary index database name. This indicates that the secondary index is being processed as a database.
2. Find the INDEX parameter on the LCHILD statement in the DBD for the secondary index. This parameter specifies the name of an indexed field in the indexed database.
3. Find the XDFLD statement in the DBD for the indexed database whose NAME parameter has the same value as the INDEX parameter found in step 2.

4. Determine if there is a SUBSEQ parameter on this XDFLD statement with a value beginning with /SX. If not, you do not have a /SX field and you do not need to make any adjustments. If there is a SUBSEQ value that begins with /SX, a /SX field is being used. The KEYLEN parameter on the PCB statement and the program's I/O area length should be incremented by four.
5. Determine if there is also a DDATA parameter on the XDFLD statement. If there is, you have one or more duplicate data fields following the /SX field in your secondary index segment. The offsets in your application program must be adjusted by four bytes.
6. Determine if there is also user data. These are fields that are not maintained by IMS. User application programs are required to create, delete, or replace user data fields. When a non-HALDB database is reorganized, user data is lost. It is not recreated when the secondary index is recreated. For these reasons most installations do not include user data in their secondary indexes. User data fields are created by making the segment larger than required to hold the sequence, subsequence, and duplicate data fields. Compare the value of the BYTES parameter in the SEGM statement of the secondary index DBD with the combined sizes of the SRCH, SUBSEQ, and DDATA fields defined in the XDFLD statement in the DBD for the indexed database. If the BYTES value is larger, you have space for user data.

These conditions are illustrated by Example 11-2, which shows the PSB; Example 11-3, which shows the secondary index DBD; Example 11-4 on page 201, which shows the indexed database DBD; and Figure 11-11 on page 201, which shows the secondary index segment.

*Example 11-2 PSB using secondary index as a database*

---

```
PCB TYPE=DB,DBDNAME=PEOSKSI,PROCOPT=G,KEYLEN=25
SENSEG NAME=SKILLCD,PARENT=0
PSBGEN LANG=COBOL,PSBNAME=SKLQRY
END
```

---

*Example 11-3 DBD for secondary index with /SX and duplicate data fields*

---

```
DBD NAME=PEOSKSI,ACCESS=INDEX
DATASET DD1=PESKSI,DEVICE=3390,SIZE=8192
SEGM NAME=SKILLCD,BYTES=31,PARENT=0
FIELD NAME=(SKTYPE,SEQ,U),BYTES=25,START=1,TYPE=C
LCHILD NAME=(SKILL,PEOPLE),INDEX=SKILIDX
DBDGEN
FINISH
END
```

---

Example 11-4 DBD for indexed database with /SX and duplicate data fields

DBD	NAME=PEOPLE,ACCESS=(HDAM,OSAM),	X
	RMNAME=(DFSHDC40,1,500,824)	
DATASET	DD1=SKILHDAM,BLOCK=1648,SCAN=0	
SEGM	NAME=NAMEMAST,BYTES=150,PTR=TWIN	
FIELD	NAME=(EMPLOYEE,SEQ,U),BYTES=60,START=1,TYPE=C	
SEGM	NAME=SKILL,PARENT=NAMEMAST,BYTES=80,PTR=TWIN	
FIELD	NAME=(SKILLTYP,SEQ,U),BYTES=21,START=1,TYPE=C	
FIELD	NAME=RATING,BYTES=6,START=22,TYPE=C	
FIELD	NAME=/SXA	
LCHILD	NAME=(SKILLCD,PEOSKSI),PTR=INDX	
XDFLD	NAME=SKILIDX,SRCH=SKILLTYP,SUBSEQ=/SXA,DDATA=RATING	
DBDGEN		
FINISH		
END		

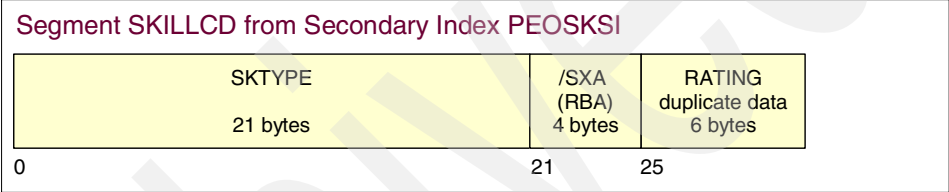


Figure 11-11 Secondary index segment with /SX and duplicate data fields

Adjusting programs and PSBs that are affected by /SX fields

The changes required when migrating to HALDB are in the examples and figures that follow. Example 11-5 shows the PSB with the KEYLEN parameter increased from 25 to 29 bytes. Example 11-6 shows the DBD for the secondary index with the BYTES parameter on the SEGM statement increased from 31 to 35 bytes and the ACCESS parameter on the DBD statement changed to PSINDEX. Example 11-7 on page 202 shows the DBD for the indexed database with the ACCESS parameter on the DBD statement changed to PHDAM. Figure 11-12 on page 202 shows the secondary index segment with the larger /SX field and resulting change in the offset to the duplicate data field.

Example 11-5 PSB using HALDB secondary index as a database

PCB	TYPE=DB,DBDNAME=PEOSKSI,PROCOPT=G,KEYLEN=29
SENSEG	NAME=SKILLCD,PARENT=0
PSBGEN	LANG=COBOL,PSBNAME=SKLQRY
END	

Example 11-6 DBD for HALDB secondary index with /SX and duplicate data field

DBD	NAME=PEOSKSI,ACCESS=PSINDEX
SEGM	NAME=SKILLCD,BYTES=35,PARENT=0

```
FIELD NAME=(SKTYPE,SEQ,U),BYTES=29,START=1,TYPE=C
LCHILD NAME=(SKILL,PEOPLE),INDEX=SKILIDX,RKSIZE=21
DBDGEN
FINISH
END
```

Example 11-7 DBD for indexed HALDB database with /SX and duplicate data fields

```
DBD NAME=PEOPLE,ACCESS=(PHDAM,OSAM), X
RMNAME=(DFSHDC40,1,100,824)
SEGM NAME=NAMEMAST,BYTES=150,PTR=TWIN
FIELD NAME=(EMPLOYEE,SEQ,U),BYTES=60,START=1,TYPE=C
SEGM NAME=SKILL,PARENT=NAMEMAST,BYTES=80,PTR=TWIN
FIELD NAME=(SKILLTYP,SEQ,U),BYTES=21,START=1,TYPE=C
FIELD NAME=RATING,BYTES=6,START=22,TYPE=C
FIELD NAME=/SXA
LCHILD NAME=(SKILLCD,PEOSKSI),PTR=INDX
XDFLD NAME=SKILIDX,SRCH=SKILLTYP,SUBSEQ=/SXA,DDATA=RATING
DBDGEN
FINISH
END
```

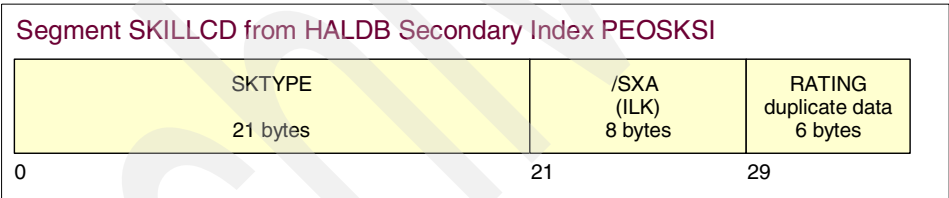


Figure 11-12 HALDB Secondary index segment with /SX and duplicate data fields

### 11.5.2 Secondary indexes with symbolic pointers

If you have an application program that processes a secondary index as a database and the secondary index uses symbolic pointing, you may need to make adjustments to your program. These adjustments can be avoided if the HALDB secondary index definition accounts for the differences in pointing with HALDB.

HALDB does not use symbolic pointers. All HALDB secondary indexes use the extended pointer set. Symbolic pointing for non-HALDB secondary indexes places the concatenated key of the target segment in the data area of the secondary index segment. Figure 11-13 on page 203 illustrates its place.

Program I/O Area for Non-HALDB Sec. Index Segment using Symbolic Pointing

Search Field	Subsequence Field	Duplicate Data Field	Concatenated Key Field	User Data
--------------	-------------------	----------------------	------------------------	-----------

Figure 11-13 Non-HALDB secondary index segment with symbolic pointing

When you convert the secondary index to HALDB, you can place the concatenated key field in the segment at the same place by including it as a duplicate data field. If it is the only or last duplicate data field, it will occupy the same place in the data portion of the segment.

### Determining if programs are affected by symbolic pointers

You should examine your PSBs to determine if you need to make adjustments. You may use the following steps.

1. See if you have a PSB with a PCB statement on which the DBDNAME field specifies a secondary index database name. This indicates that the secondary index is being processed as a database.
2. Find the LCHILD statement in the DBD for this secondary index. If it includes a PTR=SYMB parameter, this secondary index is using symbolic pointing.

Figure 11-8, Figure 11-9, and Figure 11-10 on page 204 show the PSB and DBDs for a non-HALDB database and its secondary index, which uses symbolic pointing.

*Example 11-8 PSB using secondary index with symbolic pointing as a database*

```
PCB TYPE=DB,DBDNAME=CONTRSI,PROCOPT=G,KEYLEN=8
SENSEG NAME=CONTR,PARENT=0
PSBGEN LANG=COBOL,PSBNAME=CONQRY
END
```

*Example 11-9 DBD for secondary index with symbolic pointing*

```
DBD NAME=CONTRSI,ACCESS=INDEX
DATASET DD1=CONTSI,DEVICE=3390,SIZE=8192
SEGM NAME=CONTR,BYTES=26,PARENT=0
FIELD NAME=(CONTRNUM,SEQ,U),BYTES=8,START=1,TYPE=C
LCHILD NAME=(CONTRACT,ENGAGEM),INDEX=CONTRIDX,PTR=SYMB
DBDGEN
FINISH
END
```

*Example 11-10 DBD for indexed database with symbolic pointing*

---

```
DBD      NAME=ENGAGEM,ACCESS=HDAM,RMNAME=(DFSHDC40,1,500,824)
DATASET DD1=ENGAHDAM,BLOCK=1648,SCAN=0
SEGM     NAME=CLIENT,BYTES=100,PTR=TWIN
FIELD    NAME=(CLNUM,SEQ,U),BYTES=10,START=1,TYPE=C
SEGM     NAME=CONTRACT,PARENT=CLIENT,BYTES=60,PTR=TWIN
FIELD    NAME=(CONTRNO,SEQ,U),BYTES=8,START=1,TYPE=C
LCHILD   NAME=(CONTR,CONTRSI),PTR=SYMB
XDFLD    NAME=CONTRIDX,SRCH=CONTRNO
DBDGEN
FINISH
END
```

---

### **Adjusting DBDs that are affected by symbolic pointing**

You can include the symbolic field in the DBD for the secondary index by adding a /CK field as the last or only specification in the DDATA parameter of the XDFLD statement in the DBD for the indexed database. A FIELD statement for the /CK field also must be added. When converting the DBDs for the secondary index and the indexed database, you must also eliminate the specifications of symbolic pointing. This is done by deleting the PTR=SYMB parameter on the LCHILD statement in the DBD for the secondary index and replacing PTR=SYMB with PTR=INDX on the LCHILD statement in the DBD for the indexed database.

Example 11-11 and Example 11-12 show the DBDs for the secondary index and the indexed database after they have been converted to HALDB. The secondary index includes a concatenated key field as duplicate data. The PSB is not changed when making this conversion to HALDB.

*Example 11-11 DBD for PSINDEX with concatenated key as duplicate data field*

---

```
DBD      NAME=CONTRSI,ACCESS=PSINDEX
SEGM     NAME=CONTR,BYTES=26,PARENT=0
FIELD    NAME=(CONTRNUM,SEQ,U),BYTES=8,START=1,TYPE=C
LCHILD   NAME=(CONTRACT,ENGAGEM),INDEX=CONTRIDX,RKSIZE=10
DBDGEN
FINISH
END
```

---

*Example 11-12 DBD for indexed database with concat. key field as duplicate data*

---

```
DBD      NAME=ENGAGEM,ACCESS=PHDAM,RMNAME=(DFSHDC40,1,500,824)
SEGM     NAME=CLIENT,BYTES=100,PTR=TWIN
FIELD    NAME=(CLNUM,SEQ,U),BYTES=10,START=1,TYPE=C
SEGM     NAME=CONTRACT,PARENT=CLIENT,BYTES=60,PTR=TWIN
FIELD    NAME=(CONTRNO,SEQ,U),BYTES=8,START=1,TYPE=C
FIELD    NAME=/CK1,BYTES=18,START=1
```

```
LCHILD NAME=(CONTR,CONTRSI),PTR=INDX
XDFLD NAME=CONTRIDX,SRCH=CONTRNO,DDATA=/CK1
DBDGEN
FINISH
END
```

---

## 11.6 Handling test environments

In general, application testing with HALDB databases is similar to testing with non-HALDB databases. There may be some changes required. We discuss these considerations in the following sections.

### 11.6.1 DBRC registration

You must register all HALDB databases with DBRC. If you do not register non-HALDB test database, you will have to change your test processes when migrating to HALDB.

DBRC only allows one instance of a database to be registered in a set of RECONs. Multiple databases with the same database name are not allowed. This implies that you must register multiple test versions of the same HALDB database in different sets of RECONs. Since a single batch job or online system can use only one set of RECONs, you must define all of its HALDB databases and registered non-HALDB databases in the same set of RECONs. This could force you to change some of your test procedures. If your batch test jobs shared databases that were not updated but had separate instances of databases that were updated, you are affected. You cannot use this test configuration with HALDB. Instead, you must have separate instances of all databases. Figure 11-14 on page 206 shows a non-HALDB test environment. Batch jobs A and B update separate copies of database X. They share databases Y and Z, which they only read. The databases are not registered.

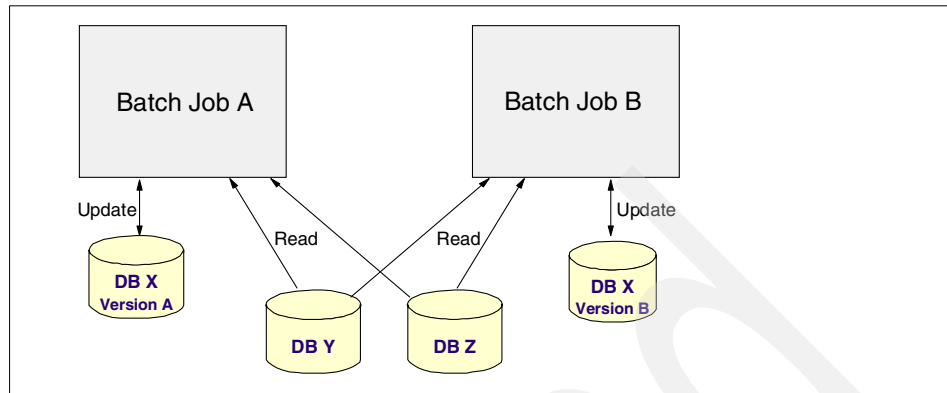


Figure 11-14 Non-HALDB test environment

Figure 11-15 shows a corresponding HALDB test environment. Batch jobs A and B each have their own copies of all databases and their own set of RECONs. All of the databases are registered in the corresponding RECONs. The registration requirement with HALDB may require you to make similar changes.

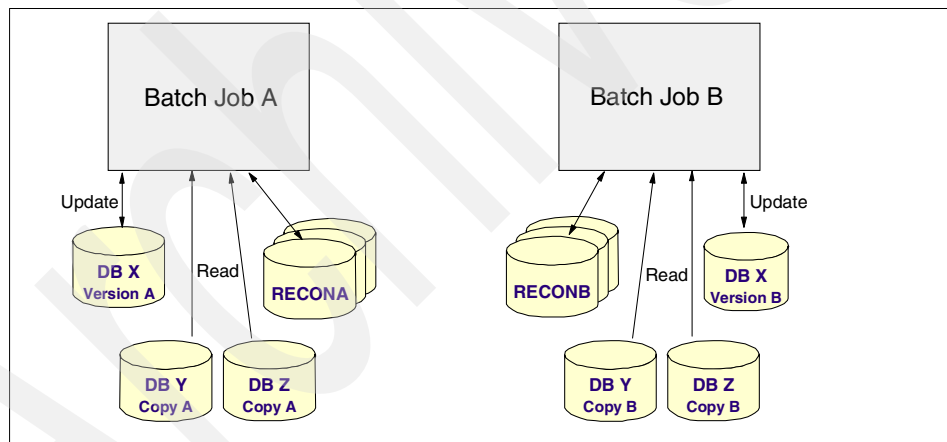


Figure 11-15 HALDB test environment

## 11.6.2 Testing with non-HALDB databases

You may want to do some testing with non-HALDB databases. Application functional testing may be done with non-HALDB databases. The incompatibilities between HALDB and non-HALDB databases are minor. They will not affect most programs. The incompatibilities apply to initial loads of logical children, processing some secondary indexes as databases, and taking advantage of HALDB exclusive capabilities. These capabilities include limiting a batch job, or a



BMP or JBP region, to a single partition; processing partitions in parallel; and making some partitions unavailable to a subsystem. Most processing will not be affected by these incompatibilities. This allows you to use non-HALDB versions of the databases for testing. Of course, you would want to do system and regression testing with HALDB databases before using them in production.

## 11.7 Copying databases to different environments

You may want to copy a database from one system to another. Some users do this to build a test database from production data. Others copy a database from an online system for processing by batch jobs such as report generators. You could have other reasons. HALDB adds additional considerations for these types of copies.

### 11.7.1 The partition ID

The partition ID is a number assigned to a partition when it is defined. Each newly defined partition is assigned a new number. The last partition ID number assigned is stored in the master database record in the RECONS. It is incremented when a new partition is defined. You cannot change this number. Partition IDs are stored in database data sets and in the RECONS. A partition cannot be opened if the ID stored in the data set does not match the ID stored in the RECONS. This limits the kinds of copies that may be done between systems. IBM plans to address this limitation. See 11.7.7, “Planned enhancement for copying partition definitions” on page 209, for a discussion of the planned enhancements.

### 11.7.2 Using image copies

Image Copy utilities copy the partition ID that is stored in the database data set. When the image copy is restored by a Recovery utility, the partition ID is restored. If the image copy is restored to a different system, the data set is unusable if the partition ID does not match the value in the RECONS. It is unlikely that they would match. Over the life of a HALDB database, partitions are likely to be added and deleted. This is done as the space requirements and root key distributions change. You should expect the partition IDs to be non-contiguous. For example, a database with five partitions could have partition IDs of 1, 4, 5, 8, and 12. It is cumbersome to define partitions for a copy of the database with matching numbers. For this reason, you may not be able to use image copies to create copies of a database for use in another system. IBM plans to address this limitation. See 11.7.7, “Planned enhancement for copying partition definitions” on page 209, for a discussion of the planned enhancements.

### 11.7.3 Using unloads

Unload and Reload utilities do not copy partition IDs. Reload uses the partition ID defined in its RECONS. This makes it easy to unload data in a database and insert it into another instance of the database using a different set of RECONS. The new copy will have new partition IDs. You can have a different number of partitions and you can have different partition boundaries when you use unload and reload to create the new instance of the database.

### 11.7.4 Copying part of the data

You may want to copy only a part of data. This is often true when creating test databases. You need only a subset of the data for testing. You may be able to unload one partition and reload only its data in the test database.

Alternatively, you may use the FABHEXTR exit in the IMS High Performance Unload tool (program number 5655-E06) to extract a subset of the database records in a database. This tool includes a PARTEXTR control statement, which causes the extract to unload a subset of records from each partition of a HALDB database. This would typically provide a more representative subset of the entire database. The output of IMS High Performance Unload may be used as input to the HD Reload utility, which creates the test database.

### 11.7.5 Copying partition definitions

The Partition Definition utility (PDU) allows you to export and import partition definitions. The export function writes partition definitions for a database to an output data set. The import function reads this data set and creates these partition definitions for the same database name in another set of RECONS. The import function allows you to easily copy your partition definitions before you move or copy a database to another set of RECONS. Of course, after the import you may modify the partition definitions. The partition definitions created by import do not maintain the partition IDs from the RECONS used for export. Instead, it assigns new partition ID numbers. IBM plans to allow you to copy the partition ID numbers with this process. These plans are explained in 11.7.7, “Planned enhancement for copying partition definitions” on page 209.

### 11.7.6 Timestamp recoveries to a copy

Some installations do a timestamp recovery to create a special instance of a database. The timestamp could be the end of a month, quarter, or year. Such copies are useful for some kinds of report processing. With non-HALDB databases the timestamp copy is created with image copies and, possibly, change accumulation and log inputs. As we mentioned in 11.7.2, “Using image

copies” on page 207, the partition ID is unlikely to allow you to use this method with HALDB databases. We explain IBM plans to eliminate this restriction in 11.7.7, “Planned enhancement for copying partition definitions” on page 209. In the meantime, you must use another method for creating your timestamp copy.

An alternative for creating a timestamp copy of a database requires the copying of your RECONS. The copy of the RECONS has the same partition IDs as the original. This solves the partition ID matching problem. It allows you to use image copies for a timestamp recovery, which creates a copy of the database. The data set names will need to be changed if the copy will be used on the system with the original data sets. These names may be changed by modifying the data set name prefixes in the RECON copy. The RECON copy is used for the timestamp recovery process. GENJCL.RECOV may be used to generate the timestamp recovery job. This technique assumes that the database was not allocated to any subsystem at the time to which you are recovering. The RECON copy must be used by the job that reads the database copy.

### **11.7.7 Planned enhancement for copying partition definitions**

IBM plans to enhance the export and import functions. The enhancement will allow you to export and import partition IDs with the partition definitions. Import will use the exported partition IDs for the partition definitions it creates. Then image copies from an exporting system may be used as inputs to recoveries in the system that imports the definitions. This will allow you to more easily copy databases from one system to another.





## Part 4

# Administration

In this part we discuss the administration of the HALDB. First we describe the IMS commands that can be used with HALDB, and then we have a chapter for the back up and recovery of HALDB. Different options of reorganization and the self-healing pointers process are discussed also in this part of the book, as well as considerations for when you need to change your existing HALDB databases. We also provide a description and examples of how you can utilize the IMS Data Management Tools for IMS administering the HALDB databases.



## HALDB online commands

In this chapter we discuss the use of IMS commands with HALDB. We explain how to use the commands for entire databases and individual partitions. We provide examples of the commands and their responses.

## 12.1 Online commands with HALDB

Commands and keywords used with HALDB are the same commands that are used with other full-function databases. There are seven of these commands:

- ▶ /DBDump
- ▶ /DBRecovery
- ▶ /DISplay
- ▶ /LOCK
- ▶ /STart
- ▶ /STOp
- ▶ /UNLock

In the command you may specify either a HALDB master database name or a partition name.

IMS keeps track of partition statuses and database statuses separately. For example, a partition may be stopped but its master database may be started. Alternatively, the partition may be started but its master database may be stopped. Before opening, authorizing, or scheduling a partition, IMS always examines the status of the partition and the database. If either one has a status that would prevent the action, IMS does not take the action.

Each partition has the access limitations of both itself and its master database. For example, if the master database has an access intent of read (RD) and one of its partitions has an access intent of update (UP), the partition cannot be updated. Alternatively, if the master database has an access intent of update (UP) and one of its partitions has an access intent of read (RD), the partition cannot be updated. Similar considerations apply to other statuses that affect access limitations, such as being stopped, locked, or DBRed.

Commands issued with a partition name act only on the partition and its statuses. Commands issued against the master database act only on the master database and its statuses. This means that a start of a master database does not start any of its partitions. If they are stopped, they remain stopped.

The following sections show how to use the commands.

## 12.2 /DISPLAY command

The /DIS DB command may be used to display the status of a HALDB database or partition.



In the TYPE column the master database is shown as PHDAM, PHIDAM, or PSINDEX. Partitions are shown as PART. Non-HALDB full-function databases are shown as DL/I.

### 12.2.1 Display of a HALDB master database

The display command for a HALDB master database shows the HALDB master and, typically, each of its partitions. When using key range partitioning, the partitions are shown in high key order. When using a partition selection exit routine, the partitions are shown in partition name sequence.

Example 12-1 shows the output of a /DISPLAY command for a master database.

*Example 12-1 /DIS DB command for master database*

**/DIS DB CUSTDB**

```
DFS4444I DISPLAY FROM ID=IM1A
```

DATABASE	TYPE	TOTAL UNUSED	TOTAL UNUSED	ACC	CONDITIONS
CUSTDB	PHDAM			UP	
CUSTDB1	PART			UP	ALLOCS
CUSTDB2	PART			UP	ALLOCS
CUSTDB3	PART			UP	ALLOCS
CUSTDB4	PART			UP	ALLOCS
*2003091/114910*					

If a /DBR command has been issued for the master database, a /DISPLAY command of the master database does not show the partitions. Figure 12-2 shows the output of the /DIS command in this situation. We explain more about the effects of /DBR commands in 12.3, “DBRECOVERY command” on page 217.

*Example 12-2 /DIS DB command for master database after /DBR command*

**/DIS DB CUSTDB**

```
DFS4444I DISPLAY FROM ID=IM1A
```

DATABASE	TYPE	TOTAL UNUSED	TOTAL UNUSED	ACC	CONDITIONS
CUSTDB	PHDAM			UP	STOPPED
*2003091/115232*					

### 12.2.2 Display of a HALDB partition

The display command for a partition shows the partition and, typically, its master database.

Example 12-3 shows the output of a /DISPLAY command for a partition.

*Example 12-3 /DIS DB command for a partition*

---

**/DIS DB CUSTDB1**

```
DFS4444I DISPLAY FROM ID=IM1A
      DATABASE  TYPE  TOTAL UNUSED  TOTAL UNUSED ACC  CONDITIONS
      CUSTDB    PHDAM
      CUSTDB1   PART
      *2003091/120027*
```

---

Example 12-4 shows an example of the use of the /DIS DB command to display multiple partitions. For each partition, the master database and the partition are displayed.

*Example 12-4 /DIS DB command for multiple partitions*

---

**/DIS DB CUSTDB1 CUSTDB2 CUSTDB3 CUSTDB4**

```
DFS4444I DISPLAY FROM ID=IM1A 284
      DATABASE  TYPE  TOTAL UNUSED  TOTAL UNUSED ACC  CONDITIONS
      CUSTDB    PHDAM
      CUSTDB1   PART
      CUSTDB    PHDAM
      CUSTDB2   PART
      CUSTDB    PHDAM
      CUSTDB3   PART
      CUSTDB    PHDAM
      CUSTDB4   PART
      *2003091/161000*
```

---

If a /DBR command has been issued for a master database, a /DISPLAY command for one of its partitions does not show the master database. Example 12-5 shows the output of the /DIS command in this situation.

*Example 12-5 /DIS DB command for a partition after /DBR command for master*

---

**/DIS DB CUSTDB1**

```
DFS4444I DISPLAY FROM ID=IM1A
      DATABASE  TYPE  TOTAL UNUSED  TOTAL UNUSED ACC  CONDITIONS
      CUSTDB1   PART
      *2003091/121151*
```

---

## 12.3 /DBRECOVERY command

The /DBR DB command may be used to deallocate and stop a HALDB database or partition.

### 12.3.1 DBR of a HALDB master database

The /DBR DB command for a HALDB master database closes and deallocates all partition data sets and unauthorizes all partitions in the database. It also sets flags in the control block for the master database. It does not set flags in the control blocks for the partitions.

The /DBR DB command for a master database is used in the way that a /DBR DB command is used for non-HALDB databases. It applies to the entire database.

Example 12-6 shows an example of the use of the /DBR command for a master database.

*Example 12-6 /DBR DB command for master database*

---

**/DBR DB CUSTDB**

```
DFS058I 11:52:23 DBRECOVERY COMMAND IN PROGRESS IM1A
DFS0488I DBR COMMAND COMPLETED. DBN= CUSTDB RC= 0 IM1A
```

---

If you issue a /DBR command for the HALDB master database, the HALDB partitions are not shown when a display command for the master database is issued. A display command for the master database only shows the master database with its STOPPED status.

### 12.3.2 DBR of a HALDB partition

The /DBR DB command for a partition closes and deallocates all data sets in the partition and unauthorizes the partition. It also sets flags in the control block for the partition. It does not set flags in the control block for the master database.

The /DBR DB command for a partition should only be used if you wish to DBR a subset of the partitions in a HALDB database. If you wish to DBR the entire database, issue the command for the master database, not for the partitions.

Example 12-7 shows an example of the use of the /DBR DB command for a partition.

*Example 12-7 /DBR DB command for a partition*

---

**/DBR DB CUSTDB1**

DFS058I 12:35:02 DBRECOVERY COMMAND IN PROGRESS IM1A  
DFS0488I DBR COMMAND COMPLETED. DBN= CUSTDB1 RC= 0 IM1A

---

Example 12-8 shows a /DIS DB command for the master database after the /DBR command for a partition.

*Example 12-8 /DIS of master database after a /DBR for a partition*

---

**/DIS DB CUSTDB**

DFS4444I DISPLAY FROM ID=IM1A

DATABASE	TYPE	TOTAL UNUSED	TOTAL UNUSED ACC	CONDITIONS
CUSTDB	PHDAM		UP	
CUSTDB1	PART		UP	STOPPED, NOTOPEN
CUSTDB2	PART		UP	ALLOCS
CUSTDB3	PART		UP	ALLOCS
CUSTDB4	PART		UP	ALLOCS

\*2003091/123659\*

---

If you issue a /DBR DB command for a partition, you must explicitly issue a /STA DB command for the partition before it may be accessed. A /STA DB command for the master database will not allow you to access the partition. When you issue the /DBR DB command for the partition, flags are set in the partition's control block indicating its status. These flags are not affected by a /STA DB command for the master database.

## 12.4 /DBDUMP command

The /DBD DB command may be used to prevent transactions or programs from updating HALDB databases and partitions. If the data sets of the database or partition are open for update, they are closed and reopened for read. These are the same actions that occur with non-HALDB full-function databases when the command is issued for them. The command may be used to ensure that image copies taken while the command is in effect are not fuzzy.

### 12.4.1 DBD of a HALDB master database

The /DBD DB command for a HALDB master database closes and reopens all partition data sets and reauthorizes all partitions in the database. It also sets flags in the control block for the master database. It does not set flags in the control blocks for the partitions.

The /DBD DB command for a master database is used in the way that a /DBD DB command is used for non-HALDB databases. It applies to the entire database.

## 12.4.2 DBD of a HALDB partition

The /DBD DB command for a partition closes and reopens all data sets in the partition and reauthorizes the partition. It also sets flags in the control block for the partition. It does not set flags in the control block for the master database.

The /DBD DB command for a partition should only be used if you wish to DBD a subset of the partitions in a HALDB database. If you wish to DBD the entire database, issue the command for the master database, not for the partition.

## 12.5 /START command

The /START DB command may be used to allow transactions or programs to update HALDB databases and partitions or to change their access intents. The command has the same functions that it has with non-HALDB databases.

### 12.5.1 Start of a HALDB master database

The /START DB command for a HALDB master database resets flags for the master database that were set by previous /DBR, /DBD, or /STOP commands issued for the master database. It does not reset flags in the control blocks for the partitions. If the ACCESS parameter is used, the access intent setting for the master database is set, but the access intent setting for the partitions in the database are not affected. The OPEN parameter is not supported when the command is issued for a HALDB master database.

### 12.5.2 Start of a HALDB partition

The /START DB command for a partition resets flags for the partition that were set by previous /DBR, /DBD, or /STOP commands issued for the partition. It does not reset flags in the control block for the master database. The command may also be used to set the access intent for a partition.

The /START DB command for a partition should only be used in three circumstances. First, previous /DBR, /DBD, or /STOP commands have been issued for the partition. Second, you wish to change the access intent for a single partition or subset of partitions. Third, you wish to use the OPEN parameter to open the partition data sets. In the third case, you may issue the command after you have issued the /START DB command for the master database.

## 12.6 /STOP command

The /STOP DB command may be used to stop the use of HALDB databases or partitions. The command has the same functions that it has with non-HALDB databases.

### 12.6.1 Stop of a HALDB master database

The /STOP DB command for a HALDB master database closes the partition data sets and unauthorizes the partitions in the database. It sets a flag in the control block for the master database. It does not set flags in the control blocks for the partitions.

### 12.6.2 Stop of a HALDB partition

The /STOP DB command for a HALDB partition closes the partition data sets and unauthorizes the partition. It sets a flag in the control block for the partition. It does not set flags in the control block of its master database.

The /STOP DB command for a partition should only be used if you wish to stop a partition or a subset of the partitions in a HALDB database. If you wish to stop the entire database, issue the command for the master database, not for the partition.

## 12.7 /LOCK command

The /LOCK DB command may be used to prevent the use of HALDB master databases or partitions. The command has the same function that it has with non-HALDB databases. It prevents subsequently scheduled programs from accessing the database or partition. It does not close the data sets or affect currently scheduled programs.

### 12.7.1 Lock of a HALDB master database

The /LOCK DB command for a HALDB master database sets a flag in the control block for the master database. It does not set flags in the control blocks for the partitions. The flag for the master database is turned off by an /UNLOCK DB command for the master database.

## 12.7.2 Lock of a HALDB partition

The /LOCK DB command for a HALDB partition sets a flag in the control block for the partition. It does not set the flag in the control blocks for the master database. The flag for the partition is turned off by an /UNLOCK DB command for the partition.

## 12.8 /UNLOCK command

The /UNLOCK DB command may be used to undo the actions of a /LOCK DB command. It allows the use of the HALDB master database or partition to be resumed. The command has the same function that it has with non-HALDB databases.

### 12.8.1 Unlock of a HALDB master database

The /UNLOCK DB command for a HALDB master database resets the lock flag in the control block for the master database. It does not reset flags in the control blocks for the partitions.

### 12.8.2 Unlock of a HALDB partition

The /UNLOCK DB command for a HALDB partition resets the lock flag in the control block for the partition. It does not reset the flag in the control block for the master database.

## 12.9 Using the ALL keyword with commands

The ALL keyword may be used with IMS database commands. The ALL keyword causes the command to act on all HALDB master databases, but not on HALDB partitions. For example, a /START DB ALL command would start all HALDB master databases but would not start any partitions.

If you DBR, DBD, or STOP a partition, the /START DB ALL command will not reset the condition. You must specify the partition explicitly in the /START DB command.

## 12.10 Commands while databases are in use

/DBR, /DBD, and /START commands may be rejected. For non-HALDB databases this occurs when a PSB for a currently scheduled BMP or JBP has a

PCB that references the database. The same is true with HALDB master databases. If the PSB for a currently scheduled BMP or JBP references a HALDB database, a /DBR, /DBD, or /START command for the master database is rejected. Message DFS0565I is issued in these cases.

When no currently scheduled BMP or JBP references the database, but a currently scheduled MPP, IFP, JMP, DBCTL thread, or ODBA thread references the master database, the command waits for the program to terminate. Then the command is processed.

Commands for HALDB partitions are treated differently. If the command references a partition, it is rejected only if a currently scheduled BMP or JBP has accessed the partition in its current scheduling. The DFS0565I message is issued.

When no currently scheduled BMP or JBP has accessed the partition, but a currently scheduled MPP, JMP, IFP, DBCTL thread, or ODBA thread has accessed the partition, the command waits for the program to terminate.

## 12.11 Command examples

This section shows some combinations of commands that illustrate the use of commands for master databases and partitions.

Example 12-9 shows the results of a /STOP command for a master database. The partition attributes that are shown in the CONDITIONS column are not changed by the /STOP command for the master database. The partitions cannot be accessed because the master database, CUSTSI, is stopped.

*Example 12-9 /STOP command for a master database*

**/DIS DB CUSTSI**

```
DFS4444I DISPLAY FROM ID=IM1A
      DATABASE  TYPE  TOTAL UNUSED  TOTAL UNUSED  ACC  CONDITIONS
      CUSTSI    PSINDEX                                UP
      CUSTSI1   PART                                UP  ALLOCS
      CUSTSI2   PART                                UP  ALLOCS
      CUSTSI3   PART                                UP  ALLOCS
      *2003091/171903*
```

**/STOP DB CUSTSI**

```
DFS058I 17:24:27 STOP COMMAND IN PROGRESS IM1A
DFS0488I STO COMMAND COMPLETED. DBN= CUSTSI  RC= 0 IM1A
```



### **/DIS DB CUSTSI**

```
DFS4444I DISPLAY FROM ID=IM1A
  DATABASE TYPE TOTAL UNUSED TOTAL UNUSED ACC CONDITIONS
  CUSTSI PSINDEX UP STOPPED
  CUSTSI1 PART UP ALLOCS
  CUSTSI2 PART UP ALLOCS
  CUSTSI3 PART UP ALLOCS
*2003091/172536*
```

---

Example 12-10 shows the results of a /DBR command for a partition followed by a /START command for its master database. The partition remains unavailable after the /START command because it does not reset the flags for the partition. /DISPLAY commands are included to show the results of the /DBR and /START commands.

*Example 12-10 /DBR for a partition and /START for its master database*

### **/DIS DB CUSTDB**

```
DFS4444I DISPLAY FROM ID=IM1A 312
  DATABASE TYPE TOTAL UNUSED TOTAL UNUSED ACC CONDITIONS
  CUSTDB PHDAM UP
  CUSTDB1 PART UP ALLOCS
  CUSTDB2 PART UP ALLOCS
  CUSTDB3 PART UP ALLOCS
  CUSTDB4 PART UP ALLOCS
*2003091/164943*
```

### **/DBR DB CUSTDB1**

```
DFS058I 16:51:23 DBRECOVERY COMMAND IN PROGRESS IM1A
DFS0488I DBR COMMAND COMPLETED. DBN= CUSTDB1 RC= 0 IM1A
```

### **/DIS DB CUSTDB**

```
DFS4444I DISPLAY FROM ID=IM1A
  DATABASE TYPE TOTAL UNUSED TOTAL UNUSED ACC CONDITIONS
  CUSTDB PHDAM UP
  CUSTDB1 PART UP STOPPED, NOTOPEN
  CUSTDB2 PART UP ALLOCS
  CUSTDB3 PART UP ALLOCS
  CUSTDB4 PART UP ALLOCS
*2003091/165254*
```

### **/STA DB CUSTDB**

```
DFS058I 16:54:33 START COMMAND IN PROGRESS IM1A
```

DFS0488I STA COMMAND COMPLETED. DBN= CUSTDB RC= 0 IM1A

**/DIS DB CUSTDB**

DFS4444I DISPLAY FROM ID=IM1A

DATABASE	TYPE	TOTAL UNUSED	TOTAL UNUSED	ACC	CONDITIONS
CUSTDB	PHDAM			UP	
CUSTDB1	PART			UP	STOPPED, NOTOPEN
CUSTDB2	PART			UP	ALLOCS
CUSTDB3	PART			UP	ALLOCS
CUSTDB4	PART			UP	ALLOCS

\*2003091/165608\*

---

Example 12-11 shows the results of a /STOP command for a partition followed by a /START DB ALL command. After the /STOP command, the /DISPLAY command response shows that the master database conditions have not changed. After the /START command, the partition remains unavailable because the ALL keyword does not cause the command to act on partitions. It only acts on the master database. /DISPLAY commands are included to show the results of the /STOP and /START commands.

*Example 12-11 /STOP for a partition and /START DB ALL*

---

**/DIS DB CUSTDB**

DFS4444I DISPLAY FROM ID=IM1A

DATABASE	TYPE	TOTAL UNUSED	TOTAL UNUSED	ACC	CONDITIONS
CUSTDB	PHDAM			UP	
CUSTDB1	PART			UP	NOTOPEN, ALLOCS
CUSTDB2	PART			UP	NOTOPEN, ALLOCS
CUSTDB3	PART			UP	NOTOPEN, ALLOCS
CUSTDB4	PART			UP	NOTOPEN, ALLOCS

\*2003091/170647\*

**/STOP DB CUSTDB2**

DFS058I 17:07:37 STOP COMMAND IN PROGRESS IM1A

DFS0488I STO COMMAND COMPLETED. DBN= CUSTDB2 RC= 0 IM1A

**/DIS DB CUSTDB**

DFS4444I DISPLAY FROM ID=IM1A

DATABASE	TYPE	TOTAL UNUSED	TOTAL UNUSED	ACC	CONDITIONS
CUSTDB	PHDAM			UP	
CUSTDB1	PART			UP	NOTOPEN, ALLOCS
+ CUSTDB2	PART			UP	STOPPED, NOTOPEN,
ALLOCS					
CUSTDB3	PART			UP	NOTOPEN, ALLOCS

```

CUSTDB4  PART                                UP  NOTOPEN, ALLOCS
*2003091/170838*

/START DB ALL

DFS058I 17:09:31 START COMMAND IN PROGRESS IM1A
DFS0488I STA COMMAND COMPLETED. KEYWORD ALL IM1A

/DIS DB CUSTDB

DFS4444I DISPLAY FROM ID=IM1A
      DATABASE TYPE TOTAL UNUSED TOTAL UNUSED ACC CONDITIONS
      CUSTDB   PHDAM                                UP
      CUSTDB1  PART                                UP  NOTOPEN, ALLOCS
+  CUSTDB2    PART                                UP  STOPPED, NOTOPEN,
  ALLOCS
      CUSTDB3  PART                                UP  NOTOPEN, ALLOCS
      CUSTDB4  PART                                UP  NOTOPEN, ALLOCS
      *2003091/171025*

```

---

**Tip:** If you have explicitly stopped, deallocated, locked, or reset the access intent for a partition, you cannot reset the condition with a command for its master database or by using the ALL keyword. You must issue the /START or /UNLOCK command using the partition name.



## Backup and recovery

In this chapter we describe the backup and recovery processes for HALDB databases. We explain the different techniques used with different types of data sets. We provide examples of the JCL used for these utilities. We show DBRC GENJCL commands to generate JCL.

## 13.1 Backup and recovery overview

The backup and recovery processes used with HALDB databases vary depending on the data set types.

Most data sets are backed up with image copies and recovered with the Database Recovery utility (DFSURDB0). These are the same processes that are used for non-HALDB databases. These processes are used for PHDAM and PHIDAM data data sets and PSINDEX data sets. These data sets have A–J suffixes for their DD names.

PHIDAM primary index data sets and ILDSs do not use these processes. They are not backed up with image copies. Database change log records are not written for them. The Index/ILDS Rebuild utility (DFSPREC0) is used to rebuild these data sets.

## 13.2 Image copies for HALDB data sets

Image copies are used to back up the HALDB database data sets that have A–J suffixes. These are PHDAM and PHIDAM data data sets and PSINDEX data sets. Image copies are not used for PHIDAM primary indexes or ILDSs. IMS provides three image copy utilities:

- ▶ The Database Image Copy utility (DFSUDMP0)
- ▶ The Database Image Copy 2 utility (DFSUDMT0)
- ▶ The Online Database Image Copy utility (DFSUICP0)

All of these utilities may be used for HALDB data sets. As with all HALDB processing, DBRC is required.

Both fuzzy and clean image copies may be made for HALDB database data sets.

The IMS Image Copy Extensions is an alternative to the use of the Image Copy utilities provided by IMS. It has full support for HALDB. This tool is explained in 16.4, “IMS Image Copy Extensions” on page 276.

You can also use various utilities supplied by the operating system to make your backup copies. These are called non-standard or user image copies. As with non-HALDB databases, these copies do not interact with DBRC. You must notify DBRC of these user image copies. Use the NOTIFY.UIC command to do this.

### 13.2.1 Database Image Copy utility (DFSUDMP0) example

Example 13-1 shows the Database Image Copy utility (DFSUDMP0) JCL for data sets in database NORDDDB. This database has one data set group and four partitions, so each partition has one data set to copy. The job creates one image copy data set for each partition. The control statements specify the master database name and the DD name for each data set. Partition names are not used on control statements. Dynamic allocation is used for the database data sets.

*Example 13-1 JCL for Image Copy utility (DFSUDMP0)*

---

```
//IC      EXEC  PGM=DFSUDMP0,PARM='DBRC=Y'
//STEPLIB DD  DISP=SHR,DSN=IMSPSA.IMS0.SDFSRESL
//        DD  DISP=SHR,DSN=IMSPSA.IM0A.MDALIB
//IMS     DD  DISP=SHR,DSN=JOUK04.DBDLIB
//SYSPRINT DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
//DBIMAGE1 DD  DISP=(,CATLG),DSN=IMSPSA.NORDDDB1.IMCOPY,
//              UNIT=3390,VOL=SER=TOTIMK,SPACE=(CYL,(1,1))
//DBIMAGE2 DD  DISP=(,CATLG),DSN=IMSPSA.NORDDDB2.IMCOPY,
//              UNIT=3390,VOL=SER=TOTIMK,SPACE=(CYL,(1,1))
//DBIMAGE3 DD  DISP=(,CATLG),DSN=IMSPSA.NORDDDB3.IMCOPY,
//              UNIT=3390,VOL=SER=TOTIMK,SPACE=(CYL,(1,1))
//DBIMAGE4 DD  DISP=(,CATLG),DSN=IMSPSA.NORDDDB4.IMCOPY,
//              UNIT=3390,VOL=SER=TOTIMK,SPACE=(CYL,(1,1))
//SYSIN   DD  *
D1 NORDDDB  NORDDDB1A DBIMAGE1
D1 NORDDDB  NORDDDB2A DBIMAGE2
D1 NORDDDB  NORDDDB3A DBIMAGE3
D1 NORDDDB  NORDDDB4A DBIMAGE4
/*
```

---

### 13.2.2 Database Image Copy 2 utility (DFSUDMT0) example

The Database Image Copy 2 utility (DFSUDMT0) is an alternative to the Image Copy utility (DFSUDMP0). It allows you to create clean image copies with minimum down time for the database. It also may be used to create fuzzy image copies. It uses the DUMP function of DFSMSdss™ to create the copy of the data sets. Copies are made in two phases, logical copy and physical copy. The logical copy is very quick. Updates may be resumed while DFSMSdss is creating the physical copy. These processes are the same with HALDB and non-HALDB database data sets.

Example 13-2 on page 230 shows the Database Image Copy 2 utility JCL to copy four database data sets to one output data set. This capability was added to Database Image Copy 2 in IMS Version 8. The control statements specify the

master database name and the DD names for the data sets. Partition names are not used on control statements. Dynamic allocation is used for the database data set.

*Example 13-2 JCL for Image Copy 2 utility (DFSUDMT0)*

---

```
//IC      EXEC  PGM=DFSRR00,PARM=(ULU,DFSUDMT0,,,,,,,,,Y)
//STEPLIB DD   DISP=SHR,DSN=IMSPSA.IMSO.SDFSRESL
//        DD   DISP=SHR,DSN=IMSPSA.IMOA.MDALIB
//IMS     DD   DISP=SHR,DSN=JOUK04.DBDLIB
//SYSPRINT DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
//DBIMAGE1 DD  DISP=(,CATLG),DSN=IMSPSA.IMCOPY2.NORDDB,
//            UNIT=3390,VOL=SER=TOTIMK,SPACE=(CYL,(5,5))
//SYSIN   DD   *
1 NORDDB  NORDDB1A DBIMAGE1          XL
S NORDDB  NORDDB2A                  XL
S NORDDB  NORDDB3A                  XL
S NORDDB  NORDDB4A                  XL
/*
```

---

### 13.2.3 Using GENJCL to create image copy JCL

The following GENJCL command may be used to create image copy JCL. The *dbname* in the command is either a partition name or a master database name:

```
GENJCL.IC DBD(dbname)
```

GENJCL.IC may be used to create either the Database Image Copy (DFSUDMP0) or the Database Image Copy 2 (DFSUDMT0) JCL. GENJCL.OIC may be used to create the Online Database Image Copy (DFSUICP0) JCL.

If the master database name is specified on a GENJCL command, JCL is generated for each partition in the database. JCL is generated for all A–J data sets in the partitions.

If a partition name is specified on a GENJCL command, JCL is generated only for that partition. JCL is generated for all A–J data sets in the partition.

## 13.3 Recoveries of HALDB data sets

HALDB database data sets that have A–J suffixes are recovered with the same processes that are used for non-HALDB database data sets. These HALDB data sets are PHDAM and PHIDAM data data sets and PSINDEX data sets. The recovery process may use the Database Change Accumulation utility



(DFSUCUM0) and the Database Recovery utility (DFSURDB0). Alternatively, tools such as the IMS Online Recovery Service (ORS) may be used.

### 13.3.1 Change Accumulation utility with HALDB

The Change Accumulation utility (DFSUCUM0) may be used for HALDB database data sets that have A–J suffixes. These data sets are PHDAM and PHIDAM data data sets and PSINDEX data sets. The utility has the same function with HALDB data sets that it has with non-HALDB database data sets. It accumulates database changes from log records and writes them to a change accumulation data set.

To include a HALDB data set in Change Accumulation utility processing, specify the master database name and the DD name on the change accumulation DB0 control statement.

HALDB database data sets may be included in change accumulation groups defined to DBRC. Different data sets from the same partition or different partitions in the same database may be in different CA groups. Data sets from different partitions, databases, or areas may be in the same CA group. The only rule is that a data set cannot be in two CA groups. GENJCL.CA will produce the appropriate control statements when HALDB database data sets are included in a change accumulation group.

### 13.3.2 Database Recovery utility and ORS with HALDB

The Database Recovery utility (DFSURDB0) may be used to recover HALDB database data sets that have A–J suffixes. These data sets are PHDAM and PHIDAM data data sets and PSINDEX data sets. The utility has the same function with HALDB data sets that it has with non-HALDB database data sets. Its inputs are a combination of an image copy, a change accumulation data set, and logs. It recovers one database data set in one execution.

To recover a HALDB data set, specify the master database name in the PARM field of the EXEC statement of the Database Recovery utility, and specify the master database name and the DD name on the control statement.

Since all PHDAM and PHIDAM VSAM data data sets and all PSINDEX data sets must be defined with the REUSE attribute, they may be reused. They do not have to be deleted and redefined before they are recovered.

The IMS Online Recovery Service (ORS) tool (product number 5655-E50) may be used to recover IMS database data sets. Like the Database Recovery utility, it may be used to recover HALDB database data sets that have A–J suffixes. These data sets are PHDAM and PHIDAM data data sets and PSINDEX data

sets. It does not recover ILDSs and PHIDAM primary index data sets. The partitions being recovered cannot be allocated to any online system, including the one doing the recovery. A /DBR command for the partitions should be used to deallocate them. The master database must be started on the system doing the recovery. Do not issue a /DBR command for the master database.

Since HALDB databases must be registered in RECONs, it is not possible to recover a HALDB database data set without using DBRC. Techniques for creating copies of HALDB database are explained in 11.7, “Copying databases to different environments” on page 207.

## **Timestamp recoveries**

Timestamp recoveries restore database data sets to their state at a previous time. They are typically done to recover from an application or operational error that incorrectly updated a database.

When a timestamp recovery is done, typically all data sets in the database are recovered to the same time. This avoids potential data integrity problems. Logically related databases and secondary indexes for the database are also recovered to the same time. If there are databases that are related by application logic, they are also recovered to this time. These considerations apply to HALDB databases. All partitions and all related databases should be recovered to the same time to avoid potential data integrity problems.

When a timestamp recovery is done for a PHIDAM database, the PHIDAM indexes must be rebuilt. The Index/ILDS Rebuild utility (DFSPREC0) is used to do this. We explain the use of this utility in 13.4, “Rebuilding ILDS and PHIDAM index data sets” on page 236.

If a timestamp recovery restores a database to a time preceding the last reorganization and the database has secondary indexes or logical relationships, the ILDSs must be rebuilt. The Index/ILDS Rebuild utility (DFSPREC0) is used to do this. We explain the use of this utility in 13.4, “Rebuilding ILDS and PHIDAM index data sets” on page 236.

It is possible to do a timestamp recovery of a partition without recovering all partitions in the database to the same time. This is a dangerous practice. If the database has a secondary index or logically related database, all of the partitions of the index or logically related database will probably have pointers to the recovered partition and other partitions. This would force you to recover all partitions of the database, its secondary indexes, and logically related databases to the same time.

## GENJCL.RECOV examples

Example 13-3 shows the GENJCL.RECOV step used to generate the recovery JCL for all partitions of the NORDDDB database.

*Example 13-3 GENJCL.RECOV for partitions of NORDDDB*

---

```
//DBRC      EXEC  PGM=DSPURX00
//STEPLIB   DD  DISP=SHR,DSN=IMSPSA.IMS0.SDFSRESL
//          DD  DISP=SHR,DSN=IMSPSA.IM0A.MDALIB
//SYSPRINT  DD  SYSOUT=*
//SYSUDUMP  DD  SYSOUT=*
//IMS       DD  DISP=SHR,DSN=JOUK04.DBDLIB
//JCLPDS    DD  DISP=SHR,DSN=IMSPSA.IM0A.PROCLIB
//JCLOUT    DD  DISP=SHR,DSN=JOUK04.HALDB.CNTL(RECOVOUT)
//JCLOUTS   DD  SYSOUT=*
//SYSIN     DD  *
            GENJCL.RECOV NOJOB DBD(NORDDDB) MEMBER(RECOVJCL)
/*
```

---

Example 13-4 shows the GENJCL.RECOV step used to generate the recovery JCL for only partition NORDDDB1.

*Example 13-4 GENJCL.RECOV for partition NORDDDB1*

---

```
//DBRC      EXEC  PGM=DSPURX00
//STEPLIB   DD  DISP=SHR,DSN=IMSPSA.IMS0.SDFSRESL
//          DD  DISP=SHR,DSN=IMSPSA.IM0A.MDALIB
//SYSPRINT  DD  SYSOUT=*
//SYSUDUMP  DD  SYSOUT=*
//IMS       DD  DISP=SHR,DSN=JOUK04.DBDLIB
//JCLPDS    DD  DISP=SHR,DSN=IMSPSA.IM0A.PROCLIB
//JCLOUT    DD  DISP=SHR,DSN=JOUK04.HALDB.CNTL(RECOVOUT)
//JCLOUTS   DD  SYSOUT=*
//SYSIN     DD  *
            GENJCL.RECOV NOJOB DBD(NORDDDB1) MEMBER(RECOVJCL)
/*
```

---

## Database recovery example

Example 13-5 shows the JCL that was generated by the GENJCL.RECOV command in Example 13-3. Since the database has four partitions and one data set group, four recovery steps are generated.

*Example 13-5 Generated recovery JCL for database NORDDDB*

---

```
//RCV1 EXEC  PGM=DFSRRCO0,
//          PARM='UDR,DFSURDB0,NORDDB,,,,,,,,,Y,,,,,,,,,'
//STEPLIB   DD  DISP=SHR,DSN=IMS810C.SDFSRESL
//SYSPRINT  DD  SYSOUT=*
```

```

//SYSUDUMP DD SYSOUT=*
//IMS DD DISP=SHR,DSN=IMS810C.DBDLIB
//NORDDDB1A DD DSN=IMSPSA.IMOA.NORDDDB.A00001,
// DISP=OLD,
// DCB=BUFNO=10
//DFSUDUMP DD DSN=JOUK03.IMCOPY2.NORDDDB.C01,
// DISP=OLD,DCB=BUFNO=10
//DFSVDUMP DD DUMMY
//DFSUCUM DD DUMMY
//DFSULOG DD DUMMY
//DFSVSAMP DD DISP=SHR,
// DSN=IMS810C.PROCLIB(DFSVS MDB)
//SYSIN DD *
S NORDDDB NORDDDB1A
/*
//RCV2 EXEC PGM=DFSRR00,
// PARM='UDR,DFSURDB0,NORDDDB,,,,,,,,,Y,,,,,,,,,'
//STEPLIB DD DISP=SHR,DSN=IMS810C.SDFSRESL
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//IMS DD DISP=SHR,DSN=IMS810C.DBDLIB
//NORDDDB2A DD DSN=IMSPSA.IMOA.NORDDDB.A00002,
// DISP=OLD,
// DCB=BUFNO=10
//DFSUDUMP DD DSN=IMSPSA.IMCOPY.NORDDDB2,
// DISP=OLD,DCB=BUFNO=10
//DFSVDUMP DD DUMMY
//DFSUCUM DD DUMMY
//DFSULOG DD DUMMY
//DFSVSAMP DD DISP=SHR,
// DSN=IMS810C.PROCLIB(DFSVS MDB)
//SYSIN DD *
S NORDDDB NORDDDB2A
/*
//RCV3 EXEC PGM=DFSRR00,
// PARM='UDR,DFSURDB0,NORDDDB,,,,,,,,,Y,,,,,,,,,'
//STEPLIB DD DISP=SHR,DSN=IMS810C.SDFSRESL
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//IMS DD DISP=SHR,DSN=IMS810C.DBDLIB
//NORDDDB3A DD DSN=IMSPSA.IMOA.NORDDDB.A00003,
// DISP=OLD,
// DCB=BUFNO=10
//DFSUDUMP DD DSN=IMSPSA.IMCOPY.NORDDDB3,
// DISP=OLD,DCB=BUFNO=10
//DFSVDUMP DD DUMMY
//DFSUCUM DD DUMMY
//DFSULOG DD DUMMY
//DFSVSAMP DD DISP=SHR,

```

```

//          DSN=IMS810C.PROCLIB(DFSVMDB)
//SYSIN     DD *
S  NORDB   NORDB3A
/*
//RCV4 EXEC PGM=DFSRR00,
//          PARM='UDR,DFSURDB0,NORDB,,,,,,,,Y,,,,,,,,',
//STEPLIB  DD DISP=SHR,DSN=IMS810C.SDFSRESL
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//IMS      DD DISP=SHR,DSN=IMS810C.DBDLIB
//NORDB4A  DD DSN=IMSPSA.IMOA.NORDB.A00004,
//          DISP=OLD,
//          DCB=BUFNO=10
//DFSUDUMP DD DSN=IMSPSA.IMCOPY.NORDB4,
//          DISP=OLD,DCB=BUFNO=10
//DFSVDUMP DD DUMMY
//DFSUCUM  DD DUMMY
//DFSULOG  DD DUMMY
//DFSVSAMP DD DISP=SHR,
//          DSN=IMS810C.PROCLIB(DFSVMDB)
//SYSIN     DD *
S  NORDB   NORDB4A
/*

```

---

Example 13-6 shows the JCL that was generated by the GENJCL.RECOV command in Example 13-4 on page 233. The GENJCL.RECOV command specified a partition name. The database has only one data set group, so only one data set is recovered for the partition.

*Example 13-6 Generated recovery JCL for partition NORDB1*

---

```

//RCV1 EXEC PGM=DFSRR00,
//          PARM='UDR,DFSURDB0,NORDB,,,,,,,,Y,,,,,,,,',
//*
//STEPLIB  DD DISP=SHR,DSN=IMSPSA.IMS0.SDFSRESL
//          DD DISP=SHR,DSN=IMSPSA.IMOA.MDALIB
//IMS      DD DISP=SHR,DSN=JOUK04.DBDLIB
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//NORDB1A  DD DSN=IMSPSA.IMOA.NORDB.A00001,
//          DISP=OLD,
//          DCB=BUFNO=10
//DFSUDUMP DD DSN=IMSPSA.NORDB1.IMCOPY,
//          DISP=OLD,DCB=BUFNO=10
//DFSVDUMP DD DUMMY
//DFSUCUM  DD DUMMY
//DFSULOG  DD DUMMY
//DFSVSAMP DD DSN=IMSPSA.IMOA.PROCLIB(DFSVMOS),DISP=SHR

```

```
//SYSIN      DD *
S  NORDDDB  NORDDDB1A
/*
```

---

## 13.4 Rebuilding ILDS and PHIDAM index data sets

PHIDAM primary index data sets and ILDSs do not use the same backup and recovery processes that are used for other HALDB data sets. They are not backed up with image copies. Database change log records are not written for them. They are not recovered with the Database Recovery utility. In a sense, they are never recovered. They are rebuilt. The Index/ILDS Rebuild utility (DFSPPREC0) is used to rebuild these data sets.

The Index/ILDS Rebuild utility scans the HALDB partition and recreates the PHIDAM primary index, the ILDS, or both. A single execution of the utility will recover only one partition.

The master database name is specified in the PARM field of the EXEC statement. The SYSIN statement specifies the partition name and the recovery type. RECOVTYP can be either INDEX, ILE, or BOTH.

INDEX rebuilds only the PHIDAM index data set. ILE rebuilds only the ILDS. BOTH rebuilds the PHIDAM index data set and the ILDS. When BOTH is used, the utility rebuilds the index and then rebuilds the ILDS. The ILDS cannot be rebuilt if the index does not exist.

Only one SYSIN control statement is allowed.

The DFSVSAMP data set provides buffer pool definitions for the data sets being read and written in the process.

Example 13-7 shows the JCL to rebuild the ILDS and the PHIDAM index data set for partition NORDDDB1 in database NORDDDB.

*Example 13-7 JCL for rebuilding ILDS and index data set*

---

```
//RCV1 EXEC PGM=DFSPPREC0,
//          PARM='ULU,DFSPPREC0,NORDDDB,,,,,,,,,Y,N'
//*
//STEPLIB DD DISP=SHR,DSN=IMSPSA.IMSO.SDFSRESL
//          DD DISP=SHR,DSN=IMSPSA.IMOA.MDALIB
//IMS      DD DISP=SHR,DSN=JOUK04.DBDLIB
//SYSPRINT DD SYSOUT=*
//DFSVSAMP DD DSN=IMSPSA.IMOA.PROCLIB(DFSVMOS),DISP=SHR
//SYSIN    DD *
PARTITION=NORDDDB1,RECOVTYP=BOTH
```

If the database has no secondary index or logical relationships, there will be no records in ILDS. The rebuild job will complete with RC=04 when BOTH or ILDS is specified for the RECOVTYP.

The PHIDAM index data set or ILDS must be deleted and redefined before it is rebuilt. The VSAM REUSE attribute cannot be used for these data sets.

### 13.4.1 Performance of the Index/ILDS Rebuild utility

The Index/ILDS Rebuild utility scans the partition to rebuild the PHIDAM index or ILDS. When rebuilding both data sets, it rebuilds the index before rebuilding the ILDS.

When rebuilding the PHIDAM index, the utility reads all of the root segments in the partition. It only needs to read the first data data set. This is the one with the A prefix.

If the roots have twin pointers, the utility reads blocks until it finds a root segment. When it finds the a root segment, it follows the twin backward pointers until it finds the first root in the partition. It then follows twin forward pointers to find all of the roots and rebuild their index entries.

If the roots do not have twin pointers, the utility reads all of the blocks in the data set. It builds index entries for each root that it finds.

When rebuilding the ILDS, the utility does unqualified GN calls for all segments that are targets of Extended Pointer Sets (EPSs). These are secondary index targets, logical parents for unidirectional logical relationships, and paired logical children for bidirectional logical relationships.

The utility uses database buffer pools. You must supply a DFSVSAMP statement with buffer pools defined for the data sets being read and written. As with other IMS database processing, performance is often optimized with a large number of buffers.

If the database uses OSAM, you should use OSAM sequential buffering. The read process is almost always a sequential process. In fact, when recovering the PHIDAM index without twin pointers for the roots, the read is strictly sequential. OSAM sequential buffering will eliminate waits for reads.

## 13.4.2 Using GENJCL.USER to generate Index/ILDS Rebuild JCL

You can use GENJCL.USER to generate the JCL for an Index/ILDS Rebuild job. IMS supplies a sample DBRC skeletal JCL member for this purpose. The member is DSPUPJCL. It is in the SDFSISRC data set. This is the data set that contains IMS supplied DBRC skeletal JCL members as well as sample application and miscellaneous source modules.

Example 13-8 shows the syntax of the GENJCL.USER command with member DSPUPJCL. User key %MDBNAME specifies the master database name. User key %DBNAME specifies the partition name. User key %RECTYP specifies the type of recovery. It may have values of INDEX, ILE, or BOTH.

*Example 13-8 Syntax of GENJCL.USER to generate DFSPREC0 JCL*

---

```
GENJCL.USER MEMBER(DSPUPJCL)
      USERKEYS((%MDBNAME,'master dbname'),
                (%DBNAME,'partition name'),
                (%RCVTYP,'recovery type'))
```

---

Example 13-9 shows the GENJCL.USER job used to generate the rebuild JCL for both the ILDS and the PHIDAM index data set of partition NORDB1 in database NORDDDB.

*Example 13-9 GENJCL.USER for DFSPREC0*

---

```
//JOUK03S JOB (999,POK),JOUK03,CLASS=A,NOTIFY=&SYSUID,
//          MSGLEVEL=(1,1),MSGCLASS=X,REGION=OM
/*JOBPARM SYSAFF=SC42
//DBRC     EXEC PGM=DSPURX00
//STEPLIB DD DISP=SHR,DSN=IMSPSA.IMS0.SDFSRESL
//          DD DISP=SHR,DSN=IMSPSA.IM0A.MDALIB
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//IMS      DD DISP=SHR,DSN=JOUK04.DBDLIB
//JCLPDS   DD DISP=SHR,DSN=IMSPSA.IMS0.SDFSISRC
//JCLOUT   DD DISP=SHR,DSN=JOUK03.HALDB.CNTL(PREC0)
//JCLOUTS DD SYSOUT=*
//SYSIN    DD *
      GENJCL.USER NOJOB MEMBER(DSPUPJCL)
      USERKEYS((%MDBNAME,'NORDDDB'),(%DBNAME,'NORDB1'),(%RCVTYP,'BOTH'))
/*
```

---

Example 13-10 shows the JCL generated by the job in Example 13-9.

*Example 13-10 Generated DFSPREC0 JCL*

---

```
//UPREC1 EXEC PGM=DFSRR00,
```



```
//          PARM='ULU,DFSPRECO,NORDDB,,,,,,,,,Y,N'
//SYSPRINT DD SYSOUT=A
//IMS      DD DSN=JOUK04.DBDLIB,DISP=SHR
//DFSVSAMP DD DSN=IMS.VSAM.PARM(OPTIONS),DISP=SHR
//SYSIN     DD *
PARTITION=NORDDB1,RECOVTYP=BOTH
/*
```

---

## 13.5 Recovering HALDB secondary index data sets

HALDB secondary index database data sets may be recovered with the same processes used to recover PHDAM and PHIDAM database data sets. You may use GENJCL.RECOV to generate the Database Recovery utility JCL for PSINDEX data sets. The secondary index database name is specified in the GENJCL.RECOV command. Example 13-11 shows a job used to generate the recovery steps for the data sets of all the partitions of the CUSTSI secondary index.

*Example 13-11 Job to generate recovery of HALDB secondary index*

```
//DBRC      EXEC PGM=DSPURX00
//STEPLIB   DD DISP=SHR,DSN=IMSPSA.IMS0.SDFSRESL
//          DD DISP=SHR,DSN=IMSPSA.IMOA.MDALIB
//SYSPRINT  DD SYSOUT=*
//SYSUDUMP  DD SYSOUT=*
//IMS       DD DISP=SHR,DSN=JOUK04.DBDLIB
//JCLPDS    DD DISP=SHR,DSN=IMSPSA.IMOA.PROCLIB
//JCLOUT    DD DISP=SHR,DSN=JOUK04.HALDB.CNTL(RECOVOU3)
//JCLOUTS   DD SYSOUT=*
//SYSIN     DD *
GENJCL.RECOV DBD(CUSTSI) ONEJOB LIST MEMBER(RECOVJCL)
```

---

Example 13-12 shows the JCL generated by the job in Example 13-11. The secondary index has three partitions. Each is recovered.

*Example 13-12 Generated recovery JCL for HALDB secondary index*

```
//IVPGNJCL JOB (999,P0K),
// 'JJ',
// CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1),
// REGION=64M
/* JOBPARM SYSAFF=SC42
//RCV1 EXEC PGM=DFSRRCOO,
//          PARM='UDR,DFSURDBO,CUSTSI,,,,,,,,,Y,,,,,,,,,'
//STEPLIB   DD DISP=SHR,DSN=IMS810C.SDFSRESL
//SYSPRINT  DD SYSOUT=*
```

```

//SYSUDUMP DD SYSOUT=*
//IMS DD DISP=SHR,DSN=IMS810C.DBDLIB
//CUSTSI1A DD DSN=IMSPSA.IMOA.CUSTSI.A00001,
// DISP=OLD
//DFSUDUMP DD DSN=IMSPSA.IC1.CUSTSI1.CUSTSI1A.D03073.T143654,
// DISP=OLD,DCB=BUFNO=10
//DFSVDUMP DD DUMMY
//DFSUCUM DD DUMMY
//DFSULOG DD DUMMY
//DFSVSAMP DD DISP=SHR,
// DSN=IMS810C.PROCLIB(DFSVMDB)
//SYSIN DD *
S CUSTSI CUSTSI1A
/*
//RCV2 EXEC PGM=DFSRRCOO,
// PARM='UDR,DFSURDB0,CUSTSI,,,,,,,,,Y,,,,,,,,,'
//STEPLIB DD DISP=SHR,DSN=IMS810C.SDFSRESL
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//IMS DD DISP=SHR,DSN=IMS810C.DBDLIB
//CUSTSI2A DD DSN=IMSPSA.IMOA.CUSTSI.A00002,
// DISP=OLD
//DFSUDUMP DD DSN=IMSPSA.IC1.CUSTSI2.CUSTSI2A.D03073.T143654,
// DISP=OLD,DCB=BUFNO=10
//DFSVDUMP DD DUMMY
//DFSUCUM DD DUMMY
//DFSULOG DD DUMMY
//DFSVSAMP DD DISP=SHR,
// DSN=IMS810C.PROCLIB(DFSVMDB)
//SYSIN DD *
S CUSTSI CUSTSI2A
/*
//RCV3 EXEC PGM=DFSRRCOO,
// PARM='UDR,DFSURDB0,CUSTSI,,,,,,,,,Y,,,,,,,,,'
//STEPLIB DD DISP=SHR,DSN=IMS810C.SDFSRESL
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//IMS DD DISP=SHR,DSN=IMS810C.DBDLIB
//CUSTSI3A DD DSN=IMSPSA.IMOA.CUSTSI.A00003,
// DISP=OLD
//DFSUDUMP DD DSN=IMSPSA.IC1.CUSTSI3.CUSTSI3A.D03073.T143654,
// DISP=OLD,DCB=BUFNO=10
//DFSVDUMP DD DUMMY
//DFSUCUM DD DUMMY
//DFSULOG DD DUMMY
//DFSVSAMP DD DISP=SHR,
// DSN=IMS810C.PROCLIB(DFSVMDB)
//SYSIN DD *
S CUSTSI CUSTSI3A

```

### 13.5.1 Using IMS Index Builder for recoveries

The IMS Index Builder is an alternative to using the Database Recovery utility or IMS Online Recovery Service (ORS) to recover secondary indexes. Instead of recovering the secondary index, you can rebuild it from the data in the indexed database. This tool is explained in 16.3, “IMS Index Builder” on page 275.

## 13.6 Batch Backout utility with HALDB

The Batch Backout utility (DFSBB000) provides the same function with HALDB that it provides for other IMS databases. It backs out updates to databases that were made by a transaction or batch program. Batch Backout reads logs from batch jobs or online systems. These logs include undo records for all HALDB databases, including PHIDAM primary index updates. No changes are required to the use of the Batch Backout utility when HALDB databases are updated. Dynamic allocation of HALDB database data sets may be used.



## HALDB reorganization

One of the primary advantages of HALDB is its simplified and shortened reorganization process. In this chapter we contrast the processes used with non-HALDB and HALDB databases. We show the unload and reload jobs used for reorganizing HALDB databases. We explain your options for the HALDB process.

Reorganizations of databases with logical relationships and secondary indexes do not require the execution of utilities to update these pointers. Instead, HALDB uses a self-healing pointer process to correct these pointers when they are used. We explain how this process works.

## 14.1 The reorganization process

The reorganization of an IMS full-function database is an unload and reload of the database. Other steps may be required for non-HALDB databases.

### 14.1.1 Non-HALDB reorganization overview

The reorganization of a non-HALDB database requires an unload of the entire database by one program execution followed by a reload of the entire database by another.

If the database has secondary indexes or logical relationships, additional steps are required. In these cases, the reload produces a work file. The work file is processed by the Prefix Resolution utility.

For secondary indexes, the output of Prefix Resolution is processed by the HISAM Unload utility. Its output is processed by the HISAM Reload utility. Alternatively, you may use a tool such as IMS Index Builder to rebuild secondary indexes. This tool recreates secondary indexes by reading and processing either the work file from the reload or the source and target segments from the reorganized database.

For logical relationships, the output of the Prefix Resolution utility is processed by the Prefix Update utility. Prefix Update updates the logical pointers.

Figure 14-1 shows the processes used to reorganize a non-HALDB database with logical relationships and secondary indexes. A lot of time is used to update pointers in the logically related database and rebuild the secondary indexes.

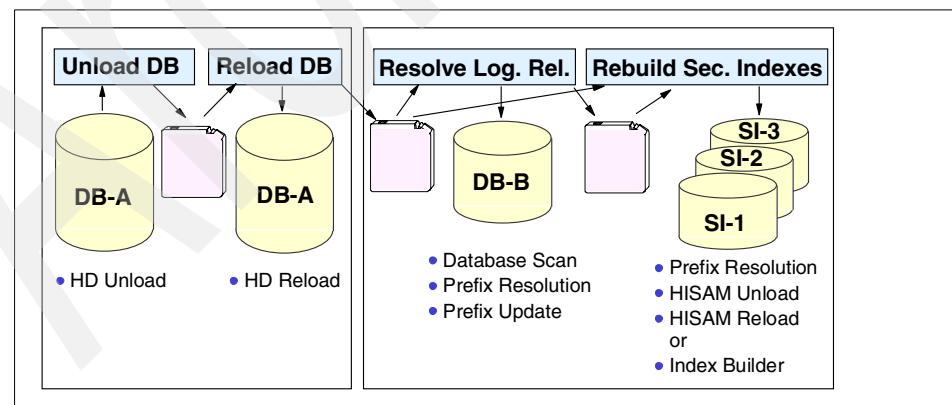


Figure 14-1 Reorganization of non-HALDB database

## 14.1.2 HALDB reorganization overview

A reorganization of a HALDB database may be done with one or more parallel processes. These processes unload one or more partitions and reload them. If the database has secondary indexes or logical relationships, additional steps are not required. The HALDB self-healing process makes updates of pointers during the reorganization unnecessary. The amount of time required for a reorganization depends on the sizes of the partitions. Smaller partitions reduce the time. You may reduce your reorganization time by creating more partitions and reorganizing them in parallel.

Figure 14-2 shows the processes used to reorganize a HALDB database with logical relationships and secondary indexes. In this case, the partitions are reorganized by parallel processes. Each partition can be unloaded and reloaded in less time than unloading and reloading the entire database. This is much faster than the process for a non-HALDB database. Additionally, no time is required for updating pointers in the logically related database or rebuilding secondary indexes. This further shortens the process.

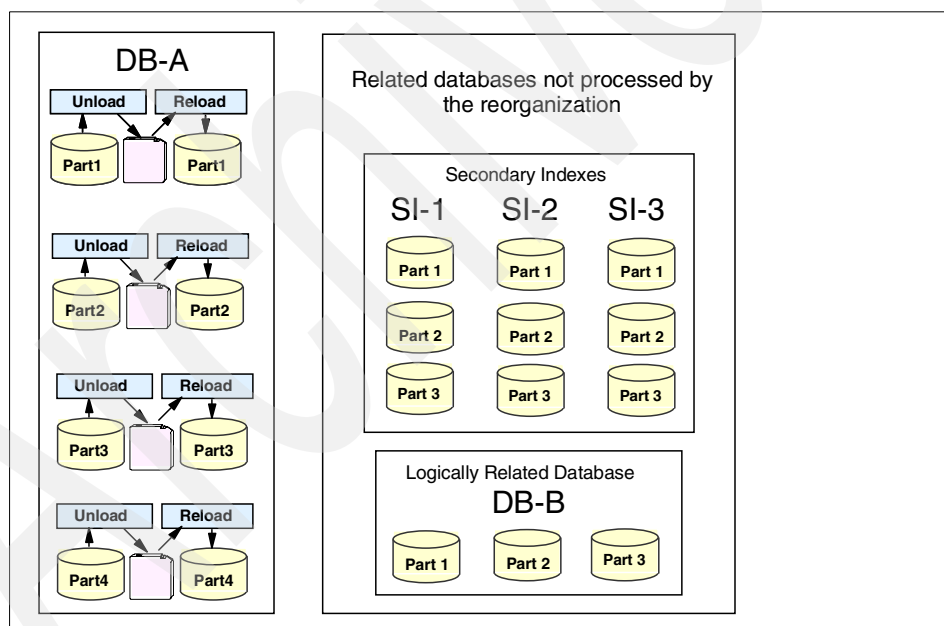


Figure 14-2 Reorganization of a HALDB database

### 14.1.3 Options for reorganization

You have several options when reorganizing a HALDB database.

- ▶ You may reorganize all of your partitions or only a subset. If only one partition is disorganized, you may unload and reload it without processing other partitions.
- ▶ You may reorganize partitions in parallel or you may reorganize the database with one process. The degree of parallelism is determined by the number of reorganization jobs you run. Each job may process one or multiple partitions. To increase the parallelism you may increase the number of reorganization jobs and decrease the number of partitions each job processes.
- ▶ You may reuse existing database data sets or you may delete them after they are unloaded and allocate new data sets for the reload.
- ▶ You may add partitions, delete partitions, or change partition boundaries. We discuss these changes in 15.2, “Changing partition definitions” on page 259.

### 14.1.4 Unloading HALDB partitions and databases

HALDB partitions or databases may be unloaded with the HD Unload utility (DFSURGU0). You may unload an entire HALDB database by not including a SYSIN DD statement. To limit the unload to a partition or set of partitions, you must include a control statement in your SYSIN data set. The control statement identifies the name of the first partition to unload. If you wish to unload more than one partition, the control statement must also include the number of partitions to unload. Consecutive partitions are unloaded. For key range partitioning, this is determined by the high keys. When using a partition selection exit routine, this is determined by the order assigned by the exit routine.

Example 14-1 shows a control statement to unload one partition.

*Example 14-1 HD Unload control statement to unload one partition*

---

```
SYSIN DD *  
PARTITION=PE002
```

---

Example 14-2 shows a control statement to unload three partitions.

*Example 14-2 HD Unload control statement to unload multiple partitions*

---

```
SYSIN DD *  
PARTITION=PE004,NUMBER=3
```

---



Do not include DD statements for the HALDB database data sets. HD Unload uses dynamic allocation for HALDB data sets. This is not true for non-HALDB databases.

You should enable OSAM sequential buffering for databases using OSAM.

You must supply buffer pools for all data sets in the partitions that are unloaded. This includes the ILDSs.

Example 14-3 shows a sample job that unloads a HALDB partition.

*Example 14-3 Sample unload JCL*

---

```
//JOUK03C JOB (999,P0K),JOUK03,CLASS=A,NOTIFY=&SYSUID,
//          MSGLEVEL=(1,1),MSGCLASS=X,REGION=OM
//JOBLIB DD DSN=IMSPSA.IMSO.SDFSRESL,DISP=SHR
//          DD DSN=IMSPSA.IMOA.MDALIB,DISP=SHR
//*****
//* HD UNLOAD FOR THE PARTITION PE002 OF PEOPLE DATABASE
//*****
//UNLOAD EXEC PGM=DFSRRCOO,REGION=1024K,
//          PARM='ULU,DFSURGUO,PEOPLE,,,,,,,,,Y,N'
//DFSRESLB DD DSN=IMSPSA.IMSO.SDFSRESL,DISP=SHR
//IMS DD DISP=SHR,DSN=JOUK03.HALDB.DBDLIB
//DFSURGU1 DD DSN=JOUK03.UNLOAD.PE002,UNIT=3390,VOL=SER=TOTIMN,
//          SPACE=(CYL,(10,5),RLSE),DISP=(NEW,CATLG)
//DFSVSAMP DD *
IOBF=(4096,50)
VSRBF=8192,50
/*
//SYSPRINT DD SYSOUT=*
//DFSCTL DD *
SBPARM ACTIV=COND
/*
//SYSIN DD *
PARTITION=PE002
/*
```

---

The High Performance Unload tool is an alternative to the use of HD Unload. You may use HP Unload to unload any number of partitions or the entire database.

### 14.1.5 Reallocating HALDB database data sets

You do not have to delete and redefine HALDB database data sets before you reload them. This applies to both OSAM and VSAM data sets. VSAM data sets, other than ILDSs, must be specified with the REUSE option. HALDB honors this option.

If you delete and redefine database data sets, you do not have to initialize the data sets for partitions that are reloaded. You must initialize any partition whose data sets are redefined if the partition is not reloaded. The requirements for partition initialization are explained in 6.1, “Partition initialization function” on page 74.

If you delete and redefine VSAM data sets, you will receive an IEC161I 152-061 message when reloading a partition. This is not an error message. It indicates that a VSAM data set was empty when it was opened. Example 14-4 shows the message for an ILDS.

*Example 14-4 IEC161I message during reload*

---

```
IEC161I 152-061,JOUK03D,RELOAD,PE001L,,,  
IEC161I JOUK03.HALDB.DB.PEOPLE.L00001,  
IEC161I JOUK03.HALDB.DB.PEOPLE.L00001.DATA,CATALOG.TOTICF2.VTOTCAT
```

---

## **Concurrent partition initialization jobs**

Partition initialization steps in concurrently executing reorganization jobs for the same database may cause job failures. The failures occur because of a timing problem. The following example illustrates the problem. A database has two partitions, A and B. We have two reorganization jobs, one for each partition. Job A reorganizes partition A. Job B reorganizes partition B. The steps of the jobs are:

1. Unload the partition.
2. Delete the partition data sets.
3. Define the partition data sets.
4. Set the *partition init needed* (PINIT) flag for the partition.
5. Execute the Prereorganization utility.
6. Reload the partition.

Step 5 initializes all partitions for which the PINIT flag is set. Step 5 from the two jobs may execute at approximately the same time. Step 5 of job A may attempt to initialize both partitions. First, it gets DBRC authorization for partition A. Before it completes, step 5 of job B executes. The PINIT flag is still on for both partitions. Step 5 of job B attempts to initialize both partitions. It requests DBRC authorization for partition A. This fails. Job B abends due to the authorization failure. It does not reload partition B. The solution is not to set the PINIT flag. Do not include steps 4 and 5. This technique assumes that there is data in both partitions. If a partition does not have data and you delete and redefine its data sets, the partition must be initialized. Empty partitions should be rare. HD Unload will complete with RC=4 if it unloads an empty partition. You may use this to alert yourself to this condition.

## 14.1.6 Reloading HALDB partitions and databases

HALDB partitions and databases may be reloaded with HD Reload (DFSURGL0). HD Reload reads the output file from HD Unload. You do not specify the partitions to be reloaded. They are determined by the records in the input file to HD Reload.

Do not include DD statements for the HALDB database data sets. HD Reload uses dynamic allocation for HALDB data sets. This is not true for non-HALDB databases.

You must supply buffer pools for all data sets in the partitions that are reloaded. This includes the ILDSs.

HD Reload sets the *image copy needed* flag for data sets in partitions that it loads. You should image copy them as you would any database data sets after they have been reloaded.

Example 14-5 shows a sample job that reloads HALDB partitions. The partitions it reloads depend on the records in the input file.

*Example 14-5 Sample reload JCL*

---

```
//JOUK03D JOB (999,P0K),JOUK03,CLASS=A,NOTIFY=&SYSUID,
//          MSGLEVEL=(1,1),MSGCLASS=X,REGION=0M
//JOB LIB DD DSN=IMSPSA.IMSO.SDFSRESL,DISP=SHR
//          DD DSN=IMSPSA.IMOA.MDALIB,DISP=SHR
//*****
//* HD RELOAD FOR THE PEOPLE DATABASE
//*****
//RELOAD EXEC PGM=DFSRR00,REGION=1024K,
//          PARM='ULU,DFSURGL0,PEOPLE,,,,,,,Y,N'
//DFSRESLB DD DSN=IMSPSA.IMSO.SDFSRESL,DISP=SHR
//IMS DD DISP=SHR,DSN=JOUK03.HALDB.DBDLIB
//DFSUINPT DD DSN=JOUK03.UNLOAD.PEOPLE,DISP=OLD
//DFSVSAMP DD *
VSRBF=8192,50
IOBF=(4096,50)
/*
//SYSPRINT DD SYSOUT=*
//DFSSTAT DD SYSOUT=*
```

---

## **ILDS updates**

HD Reload updates the ILDS for partitions that contain targets of logical relationships or secondary indexes. It has three options for updating ILDSs. They are:

- ▶ No control statement
- ▶ NOILDS control statement
- ▶ ILDSMULTI control statement

### ***No control statement***

When you do not specify a control statement in the SYSIN data for HD Reload, an ILDS entry is updated or created when a target of a secondary index or logical relationship is inserted in the partition. An entry exists if a previous reorganization loaded the target segment in the partition. The updates to the ILDS are done in VSAM update mode. When a CI or CA is filled, it must be split by VSAM. Free space in the ILDS may help avoid these splits. Updates may be random or sequential. This depends on the order in which these segments are inserted and their ILKs. The ILDS keys are based on the ILK that is based on the location of the target segment when it was created.

You may create free space in an ILDS by copying it using REPRO. REPRO honors the free space parameters in the VSAM DEFINE.

You may delete and redefine the ILDS before reloading. You may want to do this to eliminate entries in the ILDS for target segments that are no longer in the partition. HD Reload never deletes an entry. The only way to delete these entries is to delete and redefine the ILDS. On the other hand, an empty ILDS will contain no free space. A reload with a large number of target segments may require a large number of CI and CA splits.

### ***NOILDS control statement***

The NOILDS control statement was added to IMS Version 7 by APAR PQ36991 and to IMS Version 8 by PQ54227. When you specify a NOILDS control statement in the SYSIN data, HD Reload does not update or create entries in the ILDSs. They must be created by a separate process. The HALDB Index/ILDS Rebuild utility (DFSPREC0) is used to do this. Separate executions of DFSPREC0 are required for each partition. These executions may be done in parallel and on different machines. Obviously, the use of NOILDS improves the performance of HD Reload since it does not have to write the ILDS entries. On the other hand, DFSPREC0 must read the partitions to create the ILDSs. We do not recommend the use of NOILDS unless you have specific needs for it.

### ***ILDSMULTI control statement***

The ILDSMULTI control statement was added to IMS Version 7 by APAR PQ36991 and to IMS Version 8 by PQ54227. The ILDSMULTI option only applies

to migration reloads. We explain its use in 8.4.2, “HD Reload with an ILDSMULTI control statement” on page 100.

### **14.1.7 Reorganizing HALDB secondary indexes**

You may need to reorganize your HALDB secondary indexes. Many installations never reorganize non-HALDB secondary indexes. Since they are recreated with every reorganization of their indexed databases, they do not become highly disorganized. Reorganizations of HALDB databases do not require the recreation of their secondary indexes. As entries are added to these secondary indexes they may become disorganized.

HD Unload and HD Reload may be used to reorganize HALDB secondary indexes. The restrictions and recommendations for reorganizing other HALDB databases also apply to secondary indexes with one exception. HALDB secondary indexes have no ILDSs. The HD Reload control statements should not be used with secondary indexes.

## **14.2 The self-healing pointer process**

Reorganizations of databases with logical relationships and secondary indexes do not require the execution of utilities to update pointers. Instead, HALDB uses a self-healing pointer process to correct logical relationship and secondary index pointers. This process is implemented by placing a target key and an extended pointer set (EPS) in the secondary index or logically related database and by using an indirect list data set (ILDS) in each partition of PHDAM and PHIDAM databases.

### **14.2.1 The elements of the self-healing process**

Figure 14-3 on page 252 shows the elements used to implement the self-healing pointer process.

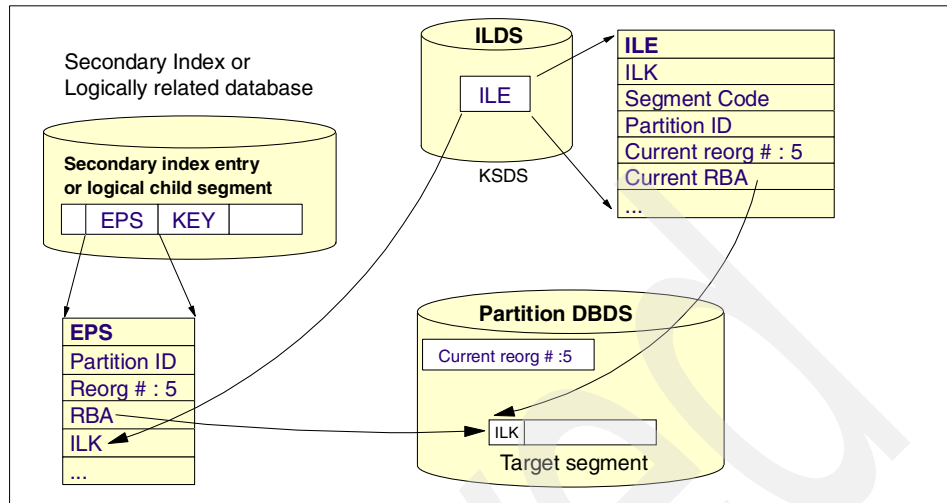


Figure 14-3 HALDB pointer before a reorganization

Each secondary index entry and each logical child segment contains the key of its target record. For secondary indexes the key of the target's root segment is included in the prefix. For logical child segments, the concatenated key of the logical parent is included in the segment data.

Each segment in a PHDAM or PHIDAM database has an Indirect List Key (ILK). The ILK is unique for the segment type across the entire database. It is composed of the RBA, partition ID, and partition reorganization number of the segment when it was first created. The ILK for a segment never changes. It is maintained across reorganizations.

Each secondary index entry or logical child segment has an extended pointer set (EPS). The extended pointer set includes the ILK of its target segment. It also contains the RBA, partition ID, and partition reorganization number for the target segment. These parts of the EPS may not be accurate. That is, they may not reflect the current location of the target segment or the current reorganization number of the target segment's partition. In Figure 14-3 they are accurate.

The target segment has an Indirect List Entry (ILE) in the Indirect List Data Set (ILDS) for a partition. The ILE contains accurate information about the target segment. This includes its current RBA, the correct partition ID, and the current reorganization number for the partition. The key of the ILE is composed of the ILK and the segment code of the target segment.

The reorganization number for a partition is physically stored in the partition's first database data set. This number is initialized by partition initialization or load, and incremented with each reorganization that reloads segments in the partition.

## 14.2.2 Finding target segments

When IMS accesses the target segment from the secondary index entry or logical child segment, it must first determine the partition in which the target resides. It uses the key in the secondary index or logical child to determine the partition. Next it must determine the location in the target partition database data set. It compares the partition ID and reorganization number of the target partition with the partition ID and reorganization number stored in the EPS. If they match, IMS uses the RBA in the EPS to locate the target segment. If they do not match, the RBA in the EPS cannot be used.

When the RBA in the EPS cannot be used, IMS uses the information in the ILE to locate the target segment. The ILE key is found by using the ILK from the EPS and the target's segment code. The ILE is read from the ILDS of the partition determined from the target's key.

Figure 14-4 illustrates a situation where the RBA in the EPS cannot be used. In the figure, the target partition has been reorganized three times since the EPS was accurate. This has moved the target segment and updated the reorganization number in the partition data set. The EPS still contains a reorganization number of 5, but the reorganization number in the partition data set is now 8. The information in the ILE has been updated by HD Reload. IMS uses the ILK from the EPS to find the ILE and uses the RBA in the ILE to find the target segment.

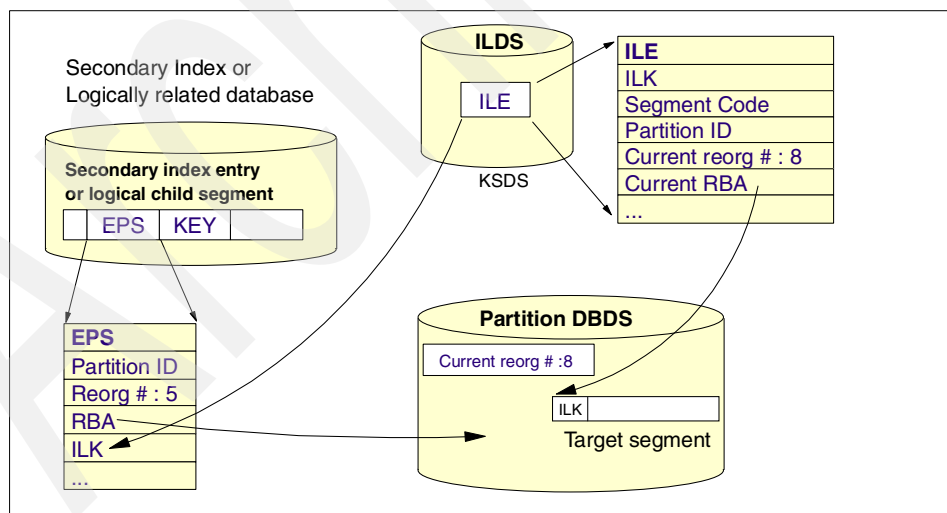


Figure 14-4 HALDB pointer after a reorganization

We will see that even though the retrieval is indirect, often the CI containing the ILE will already be in IMS's buffer pool. Nevertheless, it would be good to avoid

the indirect process if possible. We would prefer to get the target segment location from the EPS without reading the ILE. The self-healing process allows IMS to limit the use of ILEs.

### 14.2.3 Healing pointers

The self-healing process updates or corrects the information in EPSs. When the ILE is used, the information about the current location of the segment in the ILE is moved to the EPS. This allows IMS to avoid the indirect process if the EPS is used for a later retrieval. This correction to the EPS in the database buffer pool is always done. Due to locking considerations, the update may not be written to the database on DASD. The buffer containing the entry or segment with the updated EPS is marked as altered if the application program is allowed to update the database. The call must be done with a PCB allowing updates, and the subsystem must have an access intent for the partition that allows updates. If updates are not allowed, the buffer is not marked as altered. When the application reaches a sync point, it does not write buffers to DASD if they are not marked as altered. If the updated EPS is not written to DASD, the next time it is retrieved from DASD and used to find its target, IMS must use the indirect process. That is, IMS must read the ILE again.

Figure 14-5 shows the EPS after it has been healed. The RBA points to the current location. The partition ID is correct. The partition reorganization number matches the number stored in the partition database data set.

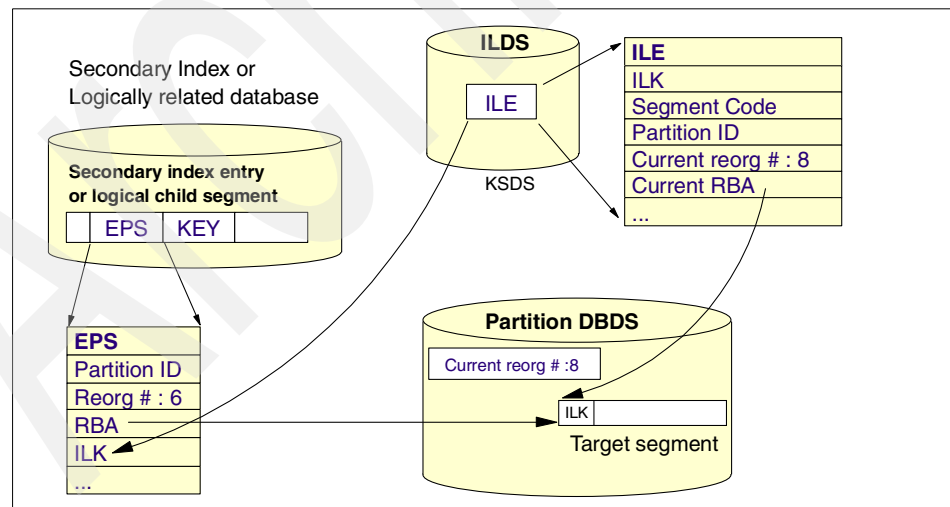


Figure 14-5 HALDB pointer after the self-healing process



## Performance of the self-healing process

The performance of the self-healing process may be much more efficient than you would anticipate.

Many pointers may be healed with a small number of ILDS reads. This is due to the use of IMS database buffering. ILDSs are database data sets. They use database buffer pools in the same way that other database data sets use them. If a CI is already in its buffer pool, it does not have to be read from DASD. Each ILE is 50 bytes. You specify the CI sizes for your ILDSs. An 8 K ILDS CI holds up to 163 ILEs. A 16 K CI holds up to 327 ILEs. So a single CI may hold many ILEs. After a reorganization IMS may need to heal many pointers to the reorganized partitions. When there are frequent uses of the CIs in an ILDS, they tend to remain in their buffer pool. One read of an ILDS CI may be sufficient to heal hundreds of pointers. As with most IMS database tuning, having a large number of buffers for frequently used data sets may be highly beneficial.

See 2.1.7, “Buffer pools” on page 25, for an explanation of how to assign HALDB data sets, including ILDSs, to buffer pools.

Another benefit of the self-healing process is that it does not waste resources healing pointers that are not used. In many secondary indexes, only a small number of entries are actually used. With a non-HALDB database, the entire index is rebuilt every time the indexed database is reorganized. With HALDB, the index is not rebuilt and only a small number of referenced index entries are updated. HALDB does not use resources to update pointers that are never used.

As we stated in 14.2.3, “Healing pointers” on page 254, when an EPS is updated, an application only marks buffers altered if the application is allowed to make updates. If updates are allowed and a block level data sharing environment is being used, a block lock is requested for the altered block. Block level data sharing environments exist when the IRLM is used and the share level for the database is either 2 or 3. The block locks are held until the application program commits its unit of work. This could cause a performance problem.

### ***Potential self-healing performance problem***

Usually update programs commit frequently. This is good programming practice. Occasionally, a program that is allowed to do updates does not actually do them. For example, a program with a PCB specifying PROCOPT=A may only read. In this case, it may not commit frequently. Since it only reads, it never holds many locks. This could change with the implementation of HALDB. If the program runs in a block level data sharing environment and invokes the healing process, it will hold block locks until they are committed. This could cause two problems. First, it might hold the locks for a long time and cause other programs to wait before they may update the blocks. Second, it could hold many locks. This could cause a storage shortage in the IRLM or a lock structure.

If you have this potential problem, you have four alternatives to address it. First, if the program does not make updates, you could use PROCOPT=G. Second, your program could commit frequently. Third, you could invoke the pointer healing process before this application is run. Another program or utility could be run before this program. The HALDB Conversion and Maintenance Aid tool supplies a pointer healing utility. Fourth, you could rebuild a secondary index with IMS Index Builder. Index Builder creates EPSs with accurate RBAs.

We believe this problem is rare. We expect that most users will be able to let the pointer healing process occur without taking any special precautions.

**Tip:** Do not rebuild your secondary indexes after a reorganization. Let the self-healing process of HALDB correct the pointers. This will shorten the outage for reorganizations and will tend to minimize the use of resources.

## Changing existing HALDB databases

After you have migrated a database to HALDB or created a new HALDB database, you may want to change it. Changes include modifications to the database definition and changes to the partitions. Partitions may be changed to accommodate more records, balance the amount of data across partitions, or delete records. In this chapter we explain how these changes are made.

## 15.1 Changing database definitions

You may change the database definition of an existing HALDB database. This is similar to the changes that you could make to a non-HALDB database. Some changes require that the database be unloaded and reloaded. Some do not. We discuss these changes and techniques for adding secondary indexes in the following sections. Partitions are not part of the database definition. We discuss changes to them in 15.2, “Changing partition definitions” on page 259.

### 15.1.1 Changes not requiring unload and reload of the database

Some database changes do not require you to unload and reload the database. These are the changes that can be made to non-HALDB databases without the unload and reload. These changes include the following:

- ▶ Adding a field definition that does not increase the size of a segment
- ▶ Deleting a field definition that does not decrease the size of a segment and is not used with a secondary index
- ▶ Adding, deleting, or changing a data capture exit routine
- ▶ Adding, deleting, or changing the data conversion user exit routine (DFSDBUX1)

These changes are made by updating the DBD while the database is not in use.

### 15.1.2 Changes requiring unload and reload of the database

Some database changes require you to unload the database, change your definitions, and reload the database. These are the same types of changes that require an unload and reload with non-HALDB databases. These changes include:

- ▶ Adding or deleting a segment type
- ▶ Changing the definition of a segment size
- ▶ Changing the definition of a sequence field
- ▶ Changing the location of a segment in the hierarchy
- ▶ Changing pointer options
- ▶ Adding, deleting, or changing a segment edit/compression exit routine

### 15.1.3 Using online change with HALDB

DBD changes for HALDB databases may be implemented in online systems with online change. The database cannot be in use during the online change process. You should issue a /DBR DB command for the database before making the change. You may issue a /START DB command for the database after the

change is committed. DBD changes affect every partition, so the master database name should be used for these commands. For more information on using online commands with HALDB, see Chapter 12, “HALDB online commands” on page 213.

### 15.1.4 Adding a secondary index

IMS does not provide a utility to add a secondary index to an existing HALDB database. The easiest way to add one is with a tool, such as the IMS Index Builder. This tool reads an existing HALDB database and creates one or more secondary indexes for it. It requires new definitions in the indexed database DBD, but does not require changes in the database data sets. It does not need to add any entries to the ILDSs because the pointers in the newly created secondary index are accurate. Later, when the database or any of its partitions are reorganized, entries for target segments will be added to the ILDSs.

If you do not have a tool to add a secondary index, you may use the following steps:

1. Create an unload file for the indexed database. You may use HD Unload or an application program that you write.
2. Add the secondary index definitions to the indexed database DBD.
3. Create the DBD for the secondary index.
4. Define the partitions for the secondary index.
5. Allocate the data sets for the secondary index.
6. Initialize the secondary index partitions.
7. Load the indexed database. You must provide the program to do this. It reads the file created in step 1. When the indexed database is loaded, secondary index entries are created. This is a random process so it may significantly elongate the load time.

If you use HD Unload in step 1, the output file contains a header record, one record for each segment, and a trailer record. The segment record includes the segment name and the segment data. Your load program for step 7 may map the records in this file by using the DSECT in the IMS DFSURGUP macro in SDFSMAAC.

## 15.2 Changing partition definitions

You can make changes that only affect one partition. These include:

- Changing the data set name prefix

- ▶ Changing the randomizing module or randomization parameters for a partition
- ▶ Changing the free space parameters for a database data set
- ▶ Changing the OSAM block size or VSAM CI size for a database data set

These changes cannot be made while the partition is authorized, but other partitions in the database may be authorized. All of these parameters, other than the VSAM CI size, are kept in the RECONs. They are specified in the Partition Definition utility or in DBRC CHANGE commands. If you change them, the *partition init needed* (PINIT) flag is set for the partition.

You can make changes that affect multiple partitions. These include adding partitions, deleting partitions, changing partition boundaries, and changing a partition selection exit routine. These changes cannot be made while the affected partitions are authorized, but other partitions in the database may be authorized.

## 15.2.1 Changing high key definitions

You may change the distribution of records across partitions when using key range partitioning. This is done by adding partitions, deleting partitions, or changing the high key specification for partitions. These changes generally require that you unload affected partitions, change your definitions, and reload the unloaded segments. Affected partitions are those whose range of keys have changed. The change either moves records into or out of affected partitions.

### Using the HALDB Migration Aid utility (DFSMAID0)

You may use the HALDB Migration Aid utility (DFSMAID0) to analyze existing HALDB databases before changing high key definitions. The use of DFSMAID0 is explained in 8.2, “HALDB Migration Aid utility (DFSMAID0)” on page 93. When the input to the utility is a HALDB database, the prefix-incr and length-incr column values will always be zeros since segment sizes will not change. An entire HALDB database is analyzed by DFSMAID0. It cannot be restricted to a single partition or subset of partitions.

### Adding a partition

You may want to add a partition. This is typically done when a partition grows too large. Adding a partition causes some of the records in an existing partition to be moved to the new partition. Figure 15-1 on page 261 shows the addition of a partition. In the figure partition D is added to the database. Its high key is 3M. Previously defined partitions A, B, and C had high keys of 2M, 4M, and high values. The addition of the new partition will require the movement of records with keys above 2M and up to 3M from partition B to partition D. This means that partition B is affected by the change. When partition D is defined with a high key of 3M, IMS sets the PINIT flag for partitions B and D. Partition initialization will

initialize these two partitions. Partitions A and C are not affected by the change. The steps in the process are:

1. Unload partition B.
2. Define the new partition with the Partition Definition utility or DBRC commands and allocate the new partition's data sets.
3. Initialize partitions B and D.
4. Reload using the output of step 1. This loads partitions B and D. The *image copy needed* flag is set for the data sets in B and D.

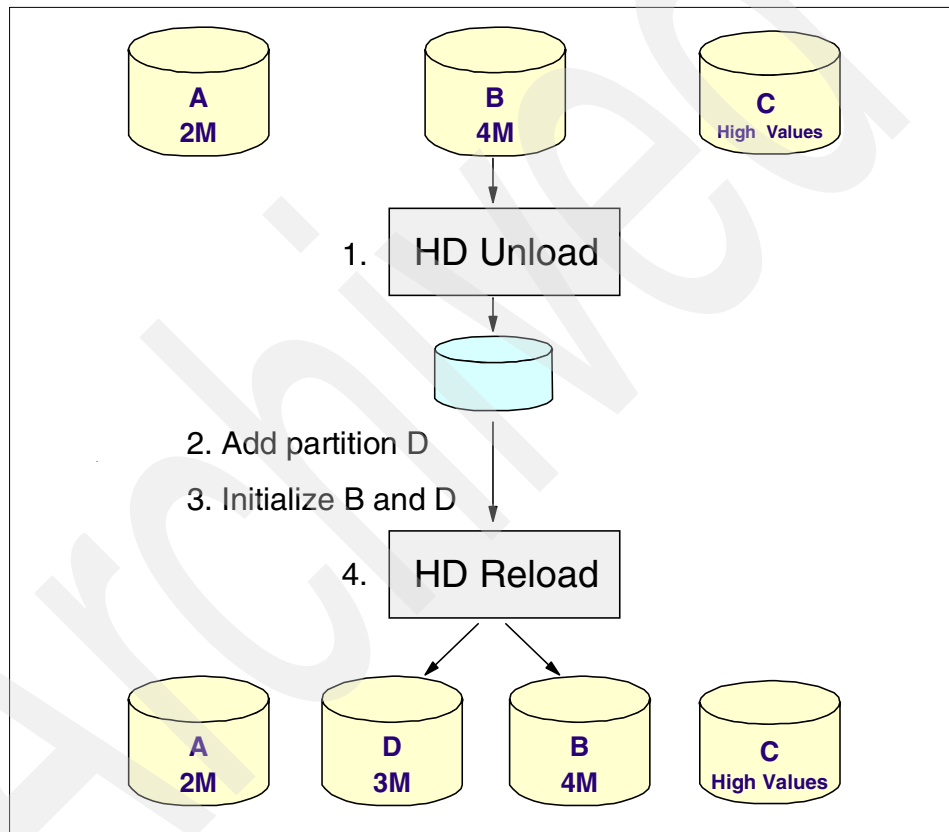


Figure 15-1 Adding a partition with key range partitioning

## Deleting a partition

You may want to delete a partition. This is typically done when a partition has little data. Deleting a partition causes its records to be moved to another partition. Figure 15-2 on page 262 shows the deletion of partition B. Its high key is 4M. The deletion of partition B will require the movement of its records to partition C. This

means that partition C is affected. When the definition of partition B is deleted, IMS sets the PINIT flag for partition C. Partition initialization will initialize partition C. Partitions A and D are not affected by the change. The steps in the process are:

1. Unload partitions B and C.
2. Delete the definition for partition B with the Partition Definition utility or DBRC commands.
3. Initialize partition C.
4. Reload using the output of step 1. This loads partition C. The *image copy needed* flag is set for the data sets in C.

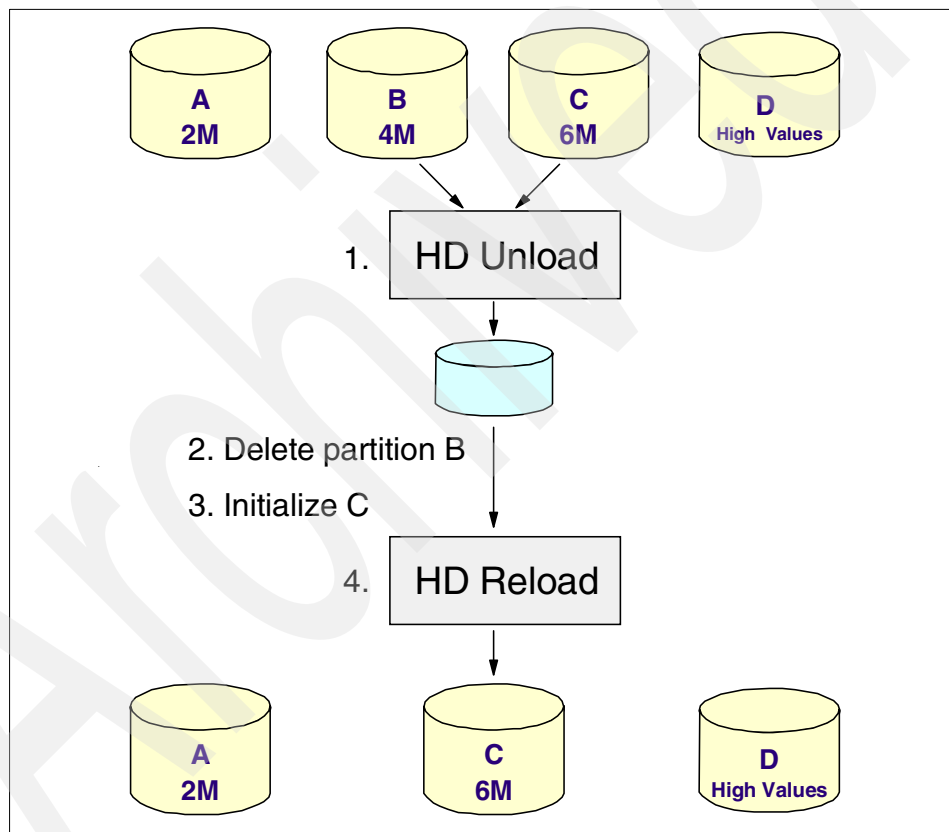


Figure 15-2 Deleting a partition with key range partitioning

### Changing the high key of a partition

You may want to change the high key of a partition. This is typically done when the amount of data in partitions is not balanced. Changing a high key causes



some records to be moved from one partition to another. Figure 15-3 on page 263 shows the change of the high key for partition B. Its high key is changed from 4M to 5M. The change will require the movement of records with keys above 4M and up to 5M from partition C to partition B. This means partitions B and C are affected. When the definition of partition B is changed, IMS sets the PINIT flag for partitions B and C. Partition initialization will initialize these partitions. Partitions A and D are not affected by the change. The steps in the process are:

1. Unload partitions B and C.
2. Change the definition for partition B with the Partition Definition utility or DBRC commands.
3. Initialize partitions B and C.
4. Reload using the output of step 1. This loads partitions B and C. The *image copy needed* flag is set for the data sets in B and C.

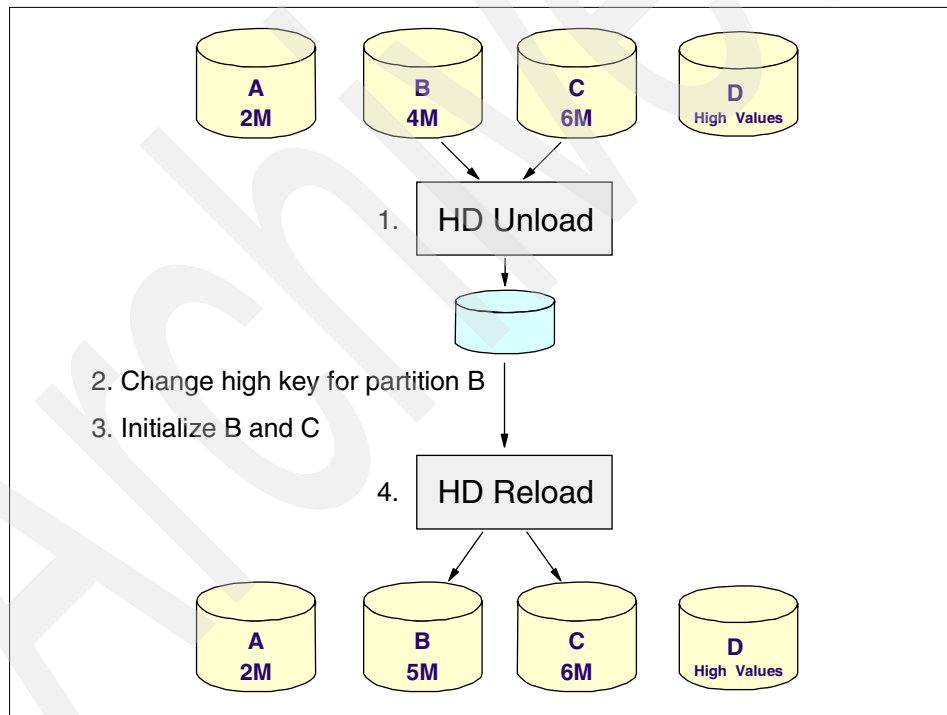


Figure 15-3 Changing a high key with key range partitioning

### Adding a higher key partition

You may add a partition that has a higher key than any existing partition. This is likely to happen if the keys are based on time or an ascending value. As the key

values increase, you can add new partitions to hold new records. Figure 15-4 on page 264 illustrates this situation. The key for the database is based on time. The high order part of the key contains the year. There is a partition for each year. A new partition is required for each new year. There are no records with keys above 2003zzz. Before they are added to the database, partition D with a high key of 2004zzz is added. Since no records are moved from an existing partition, they are not affected. The steps in the process are:

1. Define partition D.
2. Initialize partition D. The *image copy needed* flag is set for the data sets in D.

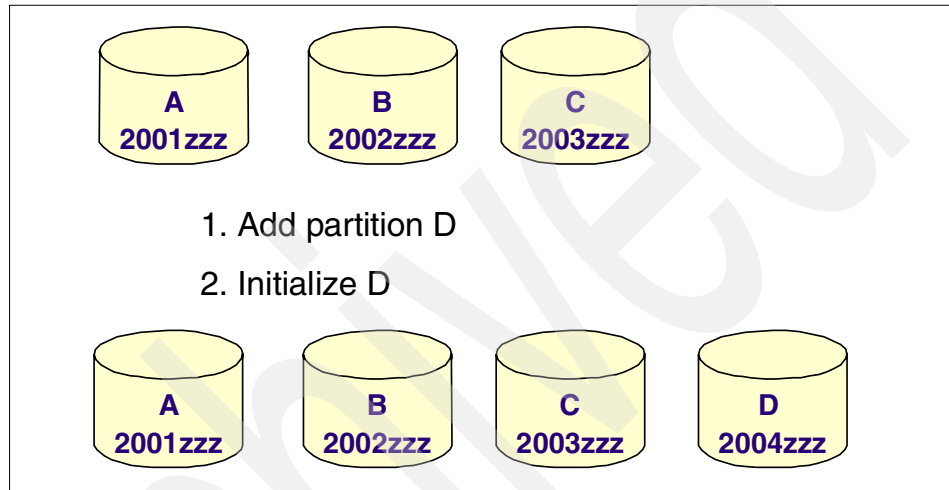


Figure 15-4 Adding a higher key partition with key range partitioning

### Deleting the lowest key partition

You may delete the partition that has the lowest key range. This is likely to happen if the keys are based on time or an ascending value. As the key values increase, you can delete a partition that has become empty or whose records are no longer of interest. Figure 15-5 on page 265 illustrates this situation. The key for the database is based on time. The high order part of the key contains the year. There is a partition for each year. In this example, partition A with a high key of 2001zzz is no longer needed. Either all of its records have been deleted or they are no longer needed. In fact, this is an efficient way to delete all the records in partition A. Since no records are moved from an existing partition, they are not affected. No partitions need to be initialized. There is only one step in the process. It is to delete partition A.

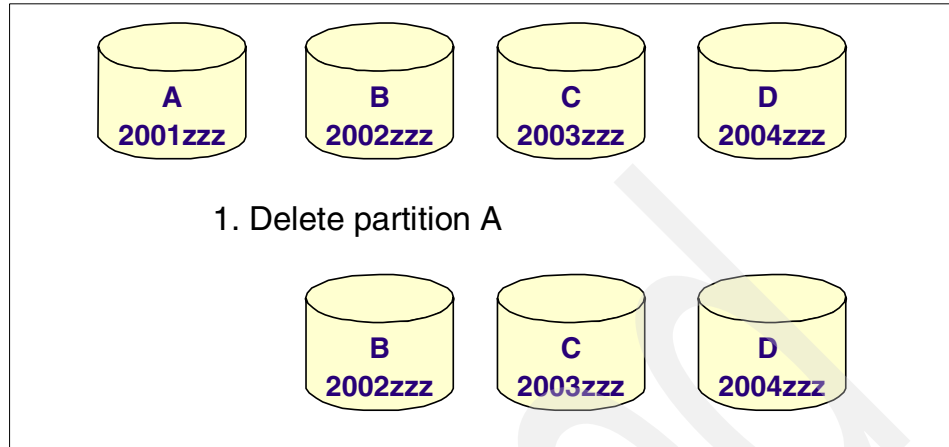


Figure 15-5 Deleting the lowest key partition with key range partitioning

If there is a secondary index, you must be careful when deleting a partition. If the secondary index has entries pointing to the deleted partition, the secondary index becomes invalid. You must rebuild it with a tool such as the IMS Index Builder. If the secondary index does not have entries pointing to the deleted partition, it is not affected and does not have to be rebuilt. If there are no records in the deleted partition, the secondary index does not have to be rebuilt. It is unaffected by the partition deletion.

## 15.2.2 Making changes with a partition selection exit routine

If you are using a partition selection exit routine, you must understand which partitions are affected by any changes you make. When a change causes records to be moved from one partition to another, HD Unload must be run before the changes are made. HD Reload must be run after the changes are made. You must unload any existing partition that is affected by the change. All affected partitions must be initialized before they are loaded by HD Reload.

Typical changes are discussed in the following sections.

### Adding a partition

You may want to add a partition. This is typically done when a partition grows too large. Adding a partition causes some of the records in one or more existing partitions to be moved to the new partition. This is similar to adding a partition to a database that uses key range partitioning. We explained this in “Adding a partition” on page 260. You must unload the partition or partitions from which records will be moved. After the unload, define the new partition and make any necessary changes to your exit routine. You may also need to change the

partition strings for some partitions. When you use a partition selection exit routine, IMS cannot know which existing partitions are affected by the addition of the partition. IMS does not set the PINIT flag for existing partitions. You must set it for them. IMS sets the flag for the new partition. Once you have set the PINIT flag in the appropriate partitions, you initialize them. Then you run HD Reload. It loads data in partitions according to the assignments made by your exit routine. The *image copy needed* flag is set for the data sets in the partitions that are initialized or loaded by HD Reload.

## Deleting a partition

You may want to delete a partition. This is typically done when a partition has little data. Deleting a partition causes its records to be moved to one or more other partitions. This is similar to deleting a partition when using key range partitioning. We explain this in “Deleting a partition” on page 261. You must unload the partition to be deleted and any other affected partitions. These are the partitions into which the records from the deleted partition will be moved. After the unload, delete the definition for the partition to be deleted and make any necessary changes to your exit routine. You may also need to change the partition strings for some partitions. When you use a partition selection exit routine, IMS cannot know which partitions are affected by the deletion of the partition. IMS does not set the PINIT flag for existing partitions unless their definitions, such as their partition strings, are changed. You must set the PINIT flag for partitions when they are affected but you make no changes to their definitions. Once you have set the PINIT flag in the appropriate partitions, you initialize them. Then you run HD Reload. It loads data in partitions according to the assignments made by your exit routine. The image copy needed flag is set for the data sets in the partitions that are initialized or loaded by HD Reload.

## Changing the exit routine

You may want to change your exit routine so that it assigns records to different existing partitions. This is typically done when the amount of data in the partitions is not balanced. You can make this change by switching exit routines or modifying the existing exit routine. As with other changes, you must unload any partitions that are affected by the change. These are the partitions from which or into which records will be moved. After the unload, you may need to set the PINIT flag in the affected partitions or reset it in unaffected partitions. If you change the name of your exit routine, the PINIT flag is set in all of the partitions. If some are not affected by the change, you should turn off their PINIT flags. You may do this with the DBRC CHANGE.DB command. If you do not change the name of your exit routine or change any partition definitions, you must set the PINIT flag for all of the affected partitions. You may need to change some partition definitions, such as their partition strings or randomizing parameters. If you change the definition of a partition, IMS will set the PINIT flags for it. Once the PINIT flags are set correctly, you initialize the affected partitions. After making the changes to the

exit routine, you run HD Reload. It loads data in partitions according to the assignments made by changes. The *image copy needed* flag is set for the data sets in the partitions that are initialized or loaded by HD Reload.

### 15.2.3 Changing partition names

You cannot change the name of a partition without deleting it and redefining it. This will cause the new partition to have a new partition ID. We explain the implications of changing partition ID numbers in the next section.

To change a partition name you must unload the partition, delete its definition, redefine the partition with the new name, initialize the partition, and reload it. The deletion of the existing definition also deletes image copy and ALLOC record information about the partition from the RECONS.

### 15.2.4 Partition ID numbers

A partition ID number is assigned to a partition when it is defined. The last number assigned is stored in the RECONS. Each new partition in a HALDB database is assigned a new higher number. These numbers are not reused. If you delete the definition of a partition and redefine it, it will have a new higher partition ID. Since partition IDs are physically stored in partition data sets, data sets created for a partition cannot be used with other partitions or with the same partition if it is deleted and redefined.

The assignment of a new partition ID has important implications for deleting and redefining a partition definition and the possibility of backing out our change. This is illustrated by the example in Figure 15-6 on page 268. In this example, we have a database with three partitions. This is shown in step 1. We decide to delete partition C. In step 2 we unload partitions B and C, delete partition C, initialize B, and run reload. This changes the database to two partitions. If we change our minds and try to back out our changes, we have a potential problem. In step 3 we redefine partition C. It has a new partition ID. Even though we may have image copied our partition C data sets before deleting the partition in step 2, these image copies are useless. They contain partition ID 3. Our new definition requires partition ID 4. There is an alternative. We could use our HD Unload to reload partitions B and C after redefining C. A reload in step 3 would be successful. Reload does not attempt to overwrite the new partition ID. After the reload, the image copy needed flags would be set for the data sets in partitions B and C.

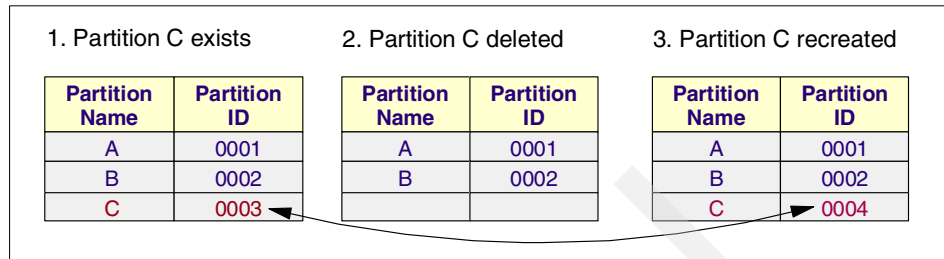


Figure 15-6 Partition ID change when deleting and recreating a partition

IBM plans to provide another alternative for this situation. We describe this enhancement to the export and import functions of the Partition Definition utility in 11.7.7, “Planned enhancement for copying partition definitions” on page 209. Using this capability you could export the partition definitions along with their partition IDs before deleting partition C. If you decided to back out your change, you could import the definitions in step 3. Since the definitions for partitions A and B already exist, they would not be imported. Only the definition of partition C would be imported. It would have a partition ID of 3. This would allow you to restore the image copies of the partition C data sets that you made before deleting the partition. Importing only partition C would require that your execution of import specified “Try all partitions” for its processing option.

## 15.3 Changing secondary indexes

Changes to secondary indexes are like changes to other HALDB databases. You may add, delete, or modify their partitions. Affected partitions must be unloaded, initialized, and reloaded. The indexed database is unaffected by changes to secondary index partitions. The HALDB Migration Aid utility (DFSMAID0) may be used to analyze changes in the high keys for secondary index partitions.

If changes to the secondary index require changes to the definition of the indexed database, you may have to unload and reload the indexed database. The rules governing these changes for all databases apply. We explained them in 15.1.1, “Changes not requiring unload and reload of the database” on page 258, and 15.1.2, “Changes requiring unload and reload of the database” on page 258. An example of a change to a secondary index is a change in a subsequence field. If you wish to add or modify a subsequence field, the indexed DBD would have to be changed. If the field in the indexed database already exists or does not require other changes to the DBD, the indexed database does not have to be unloaded and reloaded. Of course, you will need to recreate the secondary index using the new definitions. Since reorganizations of the indexed database do not cause its secondary indexes to be recreated, you must use

other means for this. You may use a tool such as the IMS Index Builder. Alternatively, you may use a technique similar to one used to add a secondary index. We describe this in 15.1.4, “Adding a secondary index” on page 259.

When you make changes to an indexed database that do not require changes to secondary index definitions, you do not need to make any changes to the secondary indexes. They do not need to be unloaded, reloaded, or rebuilt. The self-healing pointer scheme of HALDB provides this capability. The reload process for the indexed database updates the ILDSs. This is all that is required to ensure that the secondary index pointers may be used to find the moved target segments.

## **15.4 Changing logically related databases**

You may change the partitions of a logically related database without making any changes to other logically related databases. When you add, delete, or modify partitions in a database, you do not need to unload and reload any logically related database. The self-healing pointer scheme of HALDB provides this capability.





## Using IMS tools for HALDB

This chapter provides information on the following IMS tools:

- ▶ IMS High Performance Unload
- ▶ IMS High Performance Load
- ▶ IMS Index Builder
- ▶ IMS Image Copy Extensions

This chapter also provides some examples of how to use the tools with HALDB databases.

## 16.1 IMS High Performance Unload

IMS High Performance Unload provides two unloaded utilities, FABHURG1 and FABHFSU, both of which run as HSR application programs. Both unload utilities are usually run in the ULU region. In that ULU region, all database segment types are unload automatically.

The unload data set produced by these unload utilities can be used as input to the IMS HD Reorganization Reload utility and to other reload utilities that are compatible with it.

The functions and benefits of both FABHURG1 and FABHFSU include:

- ▶ Fast segment retrieval from databases  
High Performance Unload's segment retrieval engine, HSSR engine, provides the facility to retrieve segments much faster than DL/1. Also, batch DL/1 application programs that use GN calls to read a database can use the facility. Programs that read large portions of a database sequentially may get significant reductions in elapsed time and CPU use. You do not have to recompile or relink-edit the application program to use HSSR engine.
- ▶ Unload of a database without decompressing compressed segments  
Both FABHURG1 and FABHFSU can unload a database that uses the Segment Edit/Compression Exit without decompressing the segments. This can decrease the CPU time and elapsed time.
- ▶ Unload of a selected partition or a selected sequence of partitions of a HALDB  
Both FABHURG1 and FABHFSU provide a function for unloading a particular partition or a sequence of partitions from a HALDB. Unloading partition data sets in parallel, using the multiple FABHURG1 or FABHFSU jobs, reduces the elapsed time required for unloading a HALDB.
- ▶ User exit facility to allow additional processing for retrieved segments  
Each of the Unload utilities provides the User Exit facility, which provides more control on unload process than the IMS HD Reorganization Unload utility.
- ▶ Ability to continue processing after sequence errors  
High Performance Unload optionally does sequence-key checks for twin chains.

► Read of a corrupted database

High Performance Unload options give you greater control when you are unloading a database that has an incorrect pointer.

- The SKERROR option can be used to bypass pointer errors.
- The BYINDEX option can be used to force the roots of the HIDAM or PHIDAM database to be accessed from the primary index.

Example 16-1 shows the JCL to unload the entire customer (CUSTDB) database. The unloaded data set is defined by the SYSUT2 DD statement.

The HALDB Partition Definition report is produced if PARTINFO DEF is specified.

The partition-wide Segment Statistics report is produced if SEGSTAT PART is specified.

*Example 16-1 JCL to unload entire database*

---

```
//UNLOAD EXEC FABHULU,MBR=FABHURG1,DBD=CUSTDB,DBRC=Y
//STEPLIB DD DISP=SHR,DSN=IMSPSA.IMOA.MDALIB
// DD DISP=SHR,DSN=HPS.SHPSLMDO
// DD DISP=SHR,DSN=IMSPSA.IMS0.SDFSRESL
//DFSRESLB DD DSN=IMSPSA.IMS0.SDFSRESL,DISP=SHR
//IMS DD DISP=SHR,DSN=JOUK04.DBDLIB
//HSSROPT DD *
/*
//SYSIN DD *
SEGSTAT PART
//SYSPRINT DD SYSOUT=*
//SYSUT2 DD DISP=(NEW,CATLG),DSN=JOUK04.CUSTDB.HPUNLOAD,UNIT=SYSDA,
// SPACE=(CYL,(750,250),RLSE)
```

---

For more information on IMS High Performance Unload, refer to the *IMS High Performance Unload for OS/390, V1R1, User's Guide*, SC27-0936.

## 16.2 IMS High Performance Load

IMS High Performance Load provides high performance reload processing for databases. It is a performance replacement of the basic HD Reorganization Reload utility (DFSURGL0). It analyzes database performance and prepares reports containing statistics on space use and segment pointers.

The functions of this utility include:

- High Performance Load initializes an HDAM and HIDAM database. It also performs as an performance replacement of the IMS HALDB partition data

set Initialization utility (DFSUPNT0). It initializes PHDAM and PHIDAM database partitions for the DBRC RECON record whose *partition init needed* flag is on.

- ▶ High performance Load reloads an HD unloaded database data set with compressed segments in a compressed form without calling the segment compression routine.
- ▶ High Performance supports logical relationships and secondary indexes. This is also true for a HALDB.
- ▶ High Performance Load generates statistics reports on data sets, segments, and segment pointers. The reports serve as a valuable aid in tuning the database.

Using High Performance Load might benefit an installation by:

- ▶ Reducing time and costs for reorganization of IMS databases
- ▶ Having more flexible control over the reorganization of IMS databases
- ▶ Providing partition initialization of PHDAM or PHIDAM databases within its reload processing
- ▶ Providing statistical data needed for database tuning and space analysis

Example 16-2 shows the JCL for High Performance Load. It is basically the same as for the IMS HD Reorganization utility, with the addition of a DD statement.

*Example 16-2 JCL to load the entire database*

---

```
//LOAD      EXEC  PGM=DFSRR00,PARM='ULU,DFSURGL0,CUSTDB,,,,,,,,,Y'
//STEPLIB   DD    DISP=SHR,DSN=IMSPSA.IM0A.MDALIB
//          DD    DISP=SHR,DSN=HPS.SHPSLMDO
//          DD    DISP=SHR,DSN=IMSPSA.IMS0.SDFSRESL
//DFSRESLB  DD    DSN=IMSPSA.IMS0.SDFSRESL,DISP=SHR
//IMS       DD    DISP=SHR,DSN=JOUK04.DBDLIB
//DFSUIIPT  DD    DISP=SHR,DSN=JOUK04.CUSTDB.HPUNLOAD
//SYSPRINT  DD    SYSOUT=*
//SYSUDUMP  DD    SYSOUT=*
//DFSVSAMP  DD    DSN=IMSPSA.IM0A.PROCLIB(DFSVM0S),DISP=SHR
//FRRIN     DD    *
            DATASPACE=YES
```

---

For more information on IMS High Performance Load, refer to the *IMS High Performance Load for OS/390, V1R1, User's Guide, SC27-0938*.

## 16.3 IMS Index Builder

The IMS Index Builder provides the features to build or rebuild IMS secondary indexes, and to rebuild HIDAM and/or PHIDAM primary indexes. Support is provided for both IMS non-HALDB and HALDB.

The IMS Index Builder helps to simplify index recovery and maintenance, reduces index maintenance time, and eliminates the need to image copy index databases. It gives an easy-to-use one-step procedure for maintaining primary and secondary indexes.

Some of benefits of using the IMS Index Builder are:

- ▶ It minimizes the elapsed time needed to build one or more index databases.
- ▶ It uses both parallel sort and parallel load whenever more than one index is being built. This powerful feature significantly reduces the time needed to build multiple indexes of a single physical database.
- ▶ Multiple indexes can be built or rebuilt in a single JOB step.
- ▶ Reorganization time can be reduced by using the Index Builder instead of the unload/load process for each index.
- ▶ Image copy time and external storage (tape or DASD) may be reduced by eliminating the need to copy secondary indexes.
- ▶ When used in a recovery scenario, the lengthy image copy restore and log updates process can be replaced by a single Index Builder job.

*Example 16-3 JCL to build all secondary indexes from HALDB*

```
//IMSIB EXEC PGM=IIUSTART,REGION=OM
//STEPLIB DD DISP=SHR,DSN=IIU.V2R3M0.SIIULMOD
// DD DISP=SHR,DSN=IMSPSA.IM0A.MDALIB
// DD DISP=SHR,DSN=IMSPSA.IMS0.SDFSRESL
//DFSRESLB DD DISP=SHR,DSN=IMSPSA.IMS0.SDFSRESL
//IMS DD DISP=SHR,DSN=JOUK04.DBDLIB
// DD DISP=SHR,DSN=IIU.V2R3M0.SIIULMOD
//IIUIN DD *
PROC BLD_SECONDARY,CUSTDB,SELECTED
INDEX CUSTSI
INPUT IBSCAN,DBRC=Y
SORTID A,MAINSIZE=MAX FILSZ=E3000000
//*
//DFSVSAMP DD DSN=IMSPSA.IM0A.PROCLIB(DFSVSMOS),DISP=SHR
//SYSPRINT DD SYSOUT=*
//IIUSTAT DD SYSOUT=*
//IIUSNAP DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
```

In 8.5, “Migrating databases with secondary indexes” on page 101, we explain how you can use the IMS Index Builder in a migration process. We present more information in “The IMS Index Builder alternative” on page 110.

In “BLDSNDX parameter” on page 187 we explain how you may use the IMS Index Builder to build secondary indexes after an initial load of an indexed database. This is an alternative to building the secondary index as part of the initial load. This use of IMS Index Builder may supply substantial performance benefits.

In 13.5.1, “Using IMS Index Builder for recoveries” on page 241, we explain how IMS Index Builder may be used for recoveries of secondary indexes.

We explain how to use IMS Index Builder to add a secondary index to an existing HALDB database in 15.3, “Changing secondary indexes” on page 268.

For more information on the IMS Index Builder, refer to the *IMS Index Builder for Z/OS, V2R3 User's Guide*, SC27-0930.

## 16.4 IMS Image Copy Extensions

Image Copy Extension (ICE) for OS/390 extends the capabilities of the IMS Database Image Copy utility (DFSUDMP0). It includes all of its capabilities including the CIC option for fuzzy copies. It also has an extension to the capabilities of the IMS Database Recovery utility (DFSURDB0).

The functions of IMS Image Copy Extensions include:

- ▶ Image copies can be generated in compressed format, therefore reducing the number of tapes or DASD space that is needed to create the image copies. You can decompress the image copy records with the database recovery function found in the IMS Image Copy Extensions product.
- ▶ A database pointer can be validated with a hash pointer check. This is done while the image copies are taken. This reduces the elapsed time needed to execute the image copies and pointer checker.
- ▶ Multiple image copies may be written to one tape volume without rewinding the tape and remounting it for each image copy. This reduces the number of mounts issued and the amount of operator intervention required for image copies. This stacking support may be used with or without dynamic allocation.

- ICE eliminates steps by automatically allocating input and output data sets. Input and output data sets do not require DD statements. They will be dynamically allocated.

The tests uses the CUSTOMER database (PHIDAM/OSAM) that was converted from HIDAM to PHIDAM in 9.2, “Migration of a database with a secondary index” on page 142.

Example 16-4 shows the JCL of Image copy extensions for HALDB. This JCL will make an image copy of all HLADB partitions.

*Example 16-4 JCL of ICE for HALDB*

---

```
//ICE      EXEC PGM=FABJMAIN,REGION=OM
//STEPLIB DD DISP=SHR,DSN=HPS.V1R1M0.SHPSLMDO
//          DD DISP=SHR,DSN=IMSPSA.IMOA.MDALIB
//          DD DISP=SHR,DSN=IMSPSA.IMSO.SDFSRESL
//DFSRESLB DD DISP=SHR,DSN=IMSPSA.IMSO.SDFSRESL
//IMS      DD DISP=SHR,DSN=JOUK04.DBDLIB
//HISTORY DD DISP=SHR,DSN=IMSPSA.HPPC.HISTORY
//DFSPRINT DD SYSOUT=*
//ICEPRINT DD SYSOUT=*
//SYSUDUMP DD DUMMY
//MSGOUT   DD SYSOUT=*
//REPORTS  DD SYSOUT=*
//ICEIN    DD *
          GLOBAL DBRC=Y,COMP=Y,ICDALLOC=Y,ICNMRULE=Y,UNIT=SYSDA,
          ICHLQ=IMSPSA,SPACE=(CYL,10,10)
          IC      DBD=CUSTDB,ICOUT=*
/*
```

---

If the parameter PART is omitted in the ICEIN DD statement, as in Example 16-4, all partitions in the database are processed, one after the other.

Example 16-5 shows the ICEIN DD statement with PART parameters. To take advantage of one of the benefits of HALDB, the partitions can be image copied concurrently by running an image copy for each partition explicitly in different jobs, thus reducing the total elapsed time.

*Example 16-5 ICEIN DD statement with PART parameter*

---

```
//ICEIN    DD *
          GLOBAL DBRC=Y,COMP=Y,ICDALLOC=Y,ICNMRULE=Y,UNIT=(SYSDA,3),
          ICHLQ=IMSPSA,SPACE=(CYL,10,10)
          IC      DBD=CUSTDB,PART=CUSTDB1,ICOUT=*
          IC      DBD=CUSTDB,PART=CUSTDB2,ICOUT=*
          IC      DBD=CUSTDB,PART=CUSTDB3,ICOUT=*
          IC      DBD=CUSTDB,PART=CUSTDB4,ICOUT=*
```

---

/\*

---

For more information on Image Copy Extensions, refer to the *IMS Image Copy Extensions for OS/390, V1R1, User's Guide*, SC27-0924.



# Appendixes

This part contains the following appendixes:

- ▶ Appendix A, “Sample GENJCL.USER skeletons for HALDB allocation” on page 281
- ▶ Appendix B, “HALDB functions added by APARs” on page 291
- ▶ Appendix C, “Summary of HALDB utilities” on page 293



## Sample GENJCL.USER skeletons for HALDB allocation

This appendix explains how you may easily allocate IMS HALDB database data sets with the use of DBRC GENJCL.USER commands. HALDB databases often contain many data sets. The creation of jobs to allocate these data sets could be time consuming for two reasons. First, there may be many data sets. Second, the allocation jobs must use the data set names assigned by IMS in the partition definition process. These challenges may be easily overcome by automating the creation of allocation jobs. This appendix explains this automation and provides examples of jobs that use it. These examples may be adapted for use in your installation.

This appendix is a slightly modified version of the technical document that can be found on Web at the following address:

<http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/TD100491>

## Overview

This appendix explains how you may use DBRC GENJCL.USER commands to create data set allocation jobs easily. There are four steps in the process.

1. Creation of skeletal JCL

Create skeletal JCL members for the various types of database data sets. These types are indexes, indirect list data sets (ILDSs), ESDS data sets, and OSAM data sets. The skeletal JCL members include the IDCAMS DEFINE statements for the VSAM data sets or ALLOCATE statements for OSAM data sets.

2. Definition of DBDS groups

Define DBDS groups for each HALDB database when you define the database. There is one DBDS group for each of its data set types. For example, a PHIDAM database would have a group for its indexes, a group for its ILDSs, and a group for each of its data set groups.

3. Execution of GENJCL.USER commands

Use GENJCL.USER commands to generate the allocation jobs. These commands specify the skeletal JCL members and DBDS groups defined in the previous steps. The commands also may specify user variables, such as primary and secondary allocation amounts.

4. Execution of allocation jobs

Execute the jobs generated by the previous step to allocate the data sets. If necessary, you may modify the jobs to change the allocations for specific partitions. A more complete explanation of these steps follows. The examples shown are meant to be illustrations of how you may implement these techniques. You should modify them for your environment.

## Creation of skeletal JCL

A typical installation will want at least four skeletal JCL members. One member is used for allocating PHIDAM prime index and secondary index data sets. These are KSDSs. One member is used for allocating Indirect List Data Sets. These are KSDSs. One member is used for allocating VSAM data data sets. These are ESDSs. Data data sets are the data sets that contain segments in PHIDAM and PHDAM databases. One member is used for allocating OSAM data data sets.

The following shows how these skeletal JCL members could be defined. You should modify them to meet your installation's needs. For example, if SMS-managed storage is used, the values for STORCLAS and MGMTCLAS would have to meet your installation's standards. If you do not use

SMS-managed storage, you would replace STORCLAS and MGMTCLAS with the appropriate VOLUMES parameter for your installation.

## Index skeletal JCL member

You may use the following member to create PHIDAM index and secondary index data sets. In the examples in this publication, its member name is DBDDDEFX. It includes the following user variables:

<b>%UPRIM</b>	Primary space allocation amount in cylinders
<b>%USEC</b>	Secondary space allocation amount in cylinders
<b>%UKSIZE</b>	Size of key
<b>%UKOFFST</b>	Offset of key
<b>%UCISZ</b>	Data component CI size
<b>%URECSZ</b>	Record size
<b>%UFREECI</b>	CI free space percentage
<b>%UFREECA</b>	CA free space percentage

Example A-1 shows the sample member DBDDDEFX.

### *Example: A-1 Sample DBDDDEFX member*

---

```
%DELETE (%STPNO NE '00000')
//S%STPNO EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
%ENDDDEL
%SELECT DBDS(%DBNAME,%DDNAME)
DELETE %DBDSN CLUSTER
DEFINE CLUSTER (NAME(%DBDSN) -
STORCLAS(STANDARD) -
MGMTCLAS(STANDARD) -
CYLINDERS (%UPRIM %USEC) -
REUSE -
SHR(3 3) -
KEYS(%UKSIZE %UKOFFST) -
SPEED -
BWO(TYPEIMS) -
RECORDSIZE(%URECSZ %URECSZ) -
FREESPACE(%UFREECI %UFREECA) ) -
INDEX(NAME(%DBDSN.INDEX)) -
DATA(NAME(%DBDSN.DATA) -
CISZ(%UCISZ) )
%ENDSEL
```

---

## ILDS skeletal JCL member

You may use the following member to create ILDS data sets. In the examples in this section, its member name is DBDDDEFL. It includes the following user variables:

<b>%UPRIM</b>	Primary space allocation amount in cylinders
<b>%USEC</b>	Secondary space allocation amount in cylinders
<b>%UCISZ</b>	Data component CI size

Example A-2 shows the sample member DBDDDEFL.

*Example: A-2 Sample DBDDDEFL member*

---

```
%DELETE (%STPNO NE '00000')
//S%STPNO EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
%ENDDEL
%SELECT DBDS((%DBNAME,%DDNAME))
DEFINE CLUSTER (NAME(%DBDSN) -
STORCLAS(STANDARD) -
MGMTCLAS(STANDARD) -
CYLINDERS (%UPRIM %USEC) -
RECORDSIZE(50 50) -
FREESPACE(25 10) -
SHR(3 3) -
SPEED -
KEYS(9 0)) -
INDEX(NAME(%DBDSN.INDEX)) -
DATA(NAME(%DBDSN.DATA) -
CISZ(%UCISZ) )
%ENDSEL
```

---

## VSAM data skeletal JCL member

You may use the following member to create VSAM data data sets. In the examples in this section, its member name is DBDDDEFV. It includes the following user variables:

<b>%UPRIM</b>	Primary space allocation amount in cylinders
<b>%USEC</b>	Secondary space allocation amount in cylinders
<b>%UCISZ</b>	CI size
<b>%URECSZ</b>	Record size

Example A-3 on page 285 shows the sample member DBDDDEFV.

*Example: A-3 Sample DBDDDEFV member*

---

```
%DELETE (%STPNO NE '00000')
//S%STPNO EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
%ENDDDEL
%SELECT DBDS((%DBNAME,%DDNAME))
DEFINE CLUSTER (NAME(%DBDSN) -
STORCLAS(STANDARD) -
MGMTCLAS(STANDARD) -
CYLINDERS(%UPRIM %USEC) -
REUSE -
SHR(3 3) -
CISZ(%UCISZ) -
RECORDSIZE(%URECSZ %URECSZ) -
NONINDEXED SPEED ) -
DATA(NAME(%DBDSN.DATA))
%ENDSEL
```

---

## OSAM data skeletal JCL member

You may use the following member to create OSAM data data sets. In the examples in this section, its member name is DBDDDEFO. It includes the following user variables:

**%UPRIM** Primary space allocation amount in cylinders  
**%USEC** Secondary space allocation amount in cylinders

Example A-3 shows the sample member DBDDDEFO.

*Example: A-4 Sample DBDDDEFO member*

---

```
%DELETE (%STPNO NE '00000')
//S%STPNO EXEC PGM=IDCAMS,DYNAMNBR=100
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
%ENDDDEL
%SELECT DBDS((%DBNAME,%DDNAME))
ALLOCATE -
DSNAME('%DBDSN') -
FILE(%DDNAME) -
STORCLAS(STANDARD) -
MGMTCLAS(STANDARD) -
DSORG(PS) -
NEW CATALOG -
SPACE (%UPRIM %USEC) CYLINDERS
```

## Defaults member

You may use the following member to contain default values for user variables. In the examples in this section, its member name is DBDDFLTD. The GENJCL.USER commands specify this member in their DEFAULTS parameter.

Example A-3 on page 285 shows the sample member DBDDFLTD.

*Example: A-5 Sample DBDDFLTD*

---

```
%UCISZ = '4096'  
%UFREECI = '40'  
%UFREECA = '10'
```

---

## Definition of DBDS groups

The second step in the process is the definition of DBDS groups for each HALDB database. Typically, you would create the groups for a database when you define the database. These groups are used with GENJCL.USER commands to create the allocation jobs. There is one DBDS group for each of its data set types. A PHIDAM database would have a DBDS group for its indexes, a group for its ILDSs, and a group for each of its data set groups. That is, you would create a DBDS group for the A data sets, another for the B data sets, etc. A PHDAM database would have a DBDS group for its ILDSs, and a group for each of its data set groups. A PSINDEX database would have a DBDS group for its index data sets.

Example A-6 on page 287 shows an example of a PHIDAM database with two data set groups and a secondary index. The database name is RZLDBT. Its partition names are DBTP01, DBTP02, DBTP03, DBTP04, and DBTP05. The secondary index is RZLSIT1. Its partition names are SIT1P01, SIT1P02, SIT1P03, and SIT1P04. The HALDB naming convention creates DD names from these partition names. For example, the DD name of the ILDS data set for the DBTP01 partition is DBTP01L. The DD name for the PHIDAM index of this partition is DBTP01X. The DD names of the data data sets are DBTP01A and DBTP01B. The following DBRC commands create the four DBDS groups for this database and the DBDS group for the secondary index. The groups have been named to indicate the database and the type of data sets in the group. For example, DBDS group RZLDBTA contains the A data sets for the partitions in database RZLDBT.



*Example: A-6 JCL for defining the DBDS groups*

---

```
//DBDSGRP JOB (999,XXX),
// 'INIT DBDSGRPS',
// CLASS=A,MSGCLASS=T,MSGLEVEL=(1,1),
// NOTIFY=LEWIS,
// REGION=4M
//D EXEC PGM=DSPURX00
//STEPLIB DD DISP=SHR,DSN=IMS.SDFSRESL
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//IMS DD DISP=SHR,DSN=IMS.DBDLIB
//*****
//*
//SYSIN DD *
INIT.DBDSGRP GRPNAME(RZLDBTA) MEMBERS( -
(DBTP01 ,DBTP01A), -
(DBTP02 ,DBTP02A), -
(DBTP03 ,DBTP03A), -
(DBTP04 ,DBTP04A), -
(DBTP05 ,DBTP05A))
INIT.DBDSGRP GRPNAME(RZLDBTB) MEMBERS( -
(DBTP01 ,DBTP01B), -
(DBTP02 ,DBTP02B), -
(DBTP03 ,DBTP03B), -
(DBTP04 ,DBTP04B), -
(DBTP05 ,DBTP05B))
INIT.DBDSGRP GRPNAME(RZLDBTL) MEMBERS( -
(DBTP01 ,DBTP01L), -
(DBTP02 ,DBTP02L), -
(DBTP03 ,DBTP03L), -
(DBTP04 ,DBTP04L), -
(DBTP05 ,DBTP05L))
INIT.DBDSGRP GRPNAME(RZLDBTX) MEMBERS( -
(DBTP01 ,DBTP01X), -
(DBTP02 ,DBTP02X), -
(DBTP03 ,DBTP03X), -
(DBTP04 ,DBTP04X), -
(DBTP05 ,DBTP05X))
INIT.DBDSGRP GRPNAME(RZLSIT1A) MEMBERS( -
(SIT1P01 ,SIT1P01A), -
(SIT1P02 ,SIT1P02A), -
(SIT1P03 ,SIT1P03A), -
(SIT1P04 ,SIT1P04A))
```

---

## Execution of GENJCL.USER commands

The third step in the process is the use of GENJCL.USER commands to generate the allocation jobs. User variables, such as primary and secondary allocation amounts, must be specified on the GENJCL.USER commands.

Each GENJCL.USER command opens, writes, and closes the output data set. For this reason, each command uses a different member of data set IMS.JCLOUT. If multiple commands wrote to the same member, later commands would overwrite the output of earlier commands.

Example A-7 shows GENJCL.USER commands that use the DBDS groups and skeletal JCL that were created in the first two steps. OSAM is used for the data data sets. Skeletal JCL member DBDDDEFO is used for these data sets. Skeletal JCL member DBDDDEFX is used for the PHIDAM index data sets and the secondary index data sets. The keys of the PHIDAM index are 5 bytes long at offset 5. The keys of the secondary index are 5 bytes long at offset 34. These values are specified for the %UKSIZE and %UKOFFST user variables. The PHIDAM index record size is 10 bytes. The secondary index record size is 40 bytes. These values are specified for the %URECSZ user variable. Similarly, space and free space variables are also specified. Skeletal member DBDDDEFL is used for the ILDSs.

*Example: A-7 Example GENJCL.USER commands*

---

```
//GENJCLT JOB (999,XXX),
// 'GENJCL.USER',
// CLASS=A,MSGCLASS=T,MSGLEVEL=(1,1),
// NOTIFY=LEWIS,
// REGION=4M
//D EXEC PGM=DSPURX00
//STEPLIB DD DISP=SHR,DSN=IMS.SDFSRESL
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//IMS DD DISP=SHR,DSN=IMS.DBDLIB
//JCLPDS DD DISP=SHR,DSN=IMS.JCLPDS
//JCLOUTA DD DISP=SHR,DSN=IMS.JCLOUT(ALLOCTA)
//JCLOUTB DD DISP=SHR,DSN=IMS.JCLOUT(ALLOCTB)
//JCLOUTX DD DISP=SHR,DSN=IMS.JCLOUT(ALLOCTX)
//JCLOUTL DD DISP=SHR,DSN=IMS.JCLOUT(ALLOCTL)
//JCLOUTS DD DISP=SHR,DSN=IMS.JCLOUT(ALLOCTS)
//*****
//*
//SYSIN DD *
GENJCL.USER GROUP(RZLDBTA) MEMBER(DBDDDEFO) JCLOUT(JCLOUTA) LIST -
USERKEYS((%UPRIM,'2'),(%USEC,'1')) -
DEFAULTS(DBDDFLTD) ONEJOB
GENJCL.USER GROUP(RZLDBTB) MEMBER(DBDDDEFO) JCLOUT(JCLOUTB) LIST -
```

```

USERKEYS((%UPRIM,'2'),(%USEC,'1')) -
DEFAULTS(DBDDFLTD) ONEJOB
GENJCL.USER GROUP(RZLDBTX) MEMBER(DBDDDEFX) JCLOUT(JCLOUTX) LIST -
USERKEYS((%UPRIM,'2'),(%USEC,'1'),(%UKSIZE,'5'), -
(%UKOFFST,'5'),(%UCISZ,'4096'),(%URECSZ,'10')) -
DEFAULTS(DBDDFLTD) ONEJOB
GENJCL.USER GROUP(RZLDBTL) MEMBER(DBDDDEFL) JCLOUT(JCLOUTL) LIST -
USERKEYS((%UPRIM,'1'),(%USEC,'1'))
DEFAULTS(DBDDFLTD) ONEJOB
GENJCL.USER GROUP(RZLSIT1A) MEMBER(DBDDDEFX) JCLOUT(JCLOUTS) LIST -
USERKEYS((%UPRIM,'1'),(%USEC,'1'),(%UKSIZE,'5'), -
(%UKOFFST,'34'),(%UCISZ,'4096'),(%URECSZ,'40')) -
DEFAULTS(DBDDFLTD) ONEJOB

```

---

## Execution of allocation jobs

The final step in the process is the execution of the allocation jobs generated in the third step. In the example, the JCL for the jobs is written to members ALLOCTA, ALLOCTB, ALLOCTX, ALLOCTL, and ALLOCTS in IMS.JCLOUT. The process produces the same allocation parameters for the data sets in every partition. For example, the A data sets for every partition will have the same space specifications. If one or more of your partitions require different parameters, you would have to modify the allocation specifications for such partitions before you execute the jobs.



## **HALDB functions added by APARs**

Some capabilities have been added to HALDB by APARs. Other capabilities will be delivered by APARs. This appendix provides a list of these APARs and the functions they deliver.

## HALDB functions delivered by APARs

Table B-1 lists APARs that deliver HALDB functions not in the original delivery of IMS Version 7. Those that were included in the original delivery of IMS Version 8 are indicated by Included in base.

*Table B-1 Functions added by APAR*

IMS V7 APAR	IMS V8 APAR	Function
PQ35893	Included in base	INIT.DB and INIT.PART DBRC commands.
PQ36991	PQ54227	Support for ILDSMULTI and NOILDS control statements in HD Reload.
PQ38626	Included in base	HALDB support in DBCTL IVP.
PQ48421	PQ73858	Support for DISABLE/ENABLE of partitions so that DBRC recovery information will not be lost when a partition is deleted, then redefined. A new batch export capability to allow the defining of HALDBs at different sites to be exactly the same. For example, partition IDs can be preserved across export/import.
PQ49638	PQ55002	Unconditional partition initialization by the HALDB Partition Initialization utility.
PQ52858	PQ59888	CHANGE.DB, CHANGE.PART, DELETE.DB, and DELETE.PART DBRC commands.
PQ55840	PQ56463	BLDSNDX parameter on the OPTIONS statement to eliminate secondary index updates by initial loads.
PQ57313	PQ58600	HALDB control statement to provide single partition processing by batch jobs.
PQ61206	PQ72776	Eliminate allocation and open of ILDS when database has no logical relationships or secondary indexes.
PQ65486	PQ65489	Allow use of PCB name or PCB label in HALDB control statement for single partition processing.

## Summary of HALDB utilities

This appendix lists all of the database utilities that can be run against HALDBs and refers to the chapters where we discuss more about them. Additional information about the utilities can also be found in the manual *IMS Version 8 Utilities Reference: Database and Transaction Manager*, SC27-1308.

# IMS database utilities for HALDBs

Table C-1 lists all of the databases utilities that can be run against HALDBs.

*Table C-1 Utilities that can be run against HALDBs*

Utility	Description	Comment
DFSMAID0	HALDB Migration Aid	Refer to 8.2, "HALDB Migration Aid utility (DFSMAID0)" on page 93.
DFSPREC0	HALDB Index/ILDS Rebuild	Refer to "Generating Index/ILDS Rebuild utility JCL" on page 41.
DFSUPNT0	HALDB Partition Data Set Initialization	Refer to 6.4, "Database Partition Data Set Initialization utility" on page 76.
%DFSHALDB	HALDB Partition Definition	Invocation of the utility by a CLIST. %DFSHALDB is the TSO invocation of module DSPXPDDU. Refer to Chapter 4, "Partition Definition utility" on page 45.
DFSURGU0	HD Reorganization Unload	Applies to PHDAM, PHIDAM, and PSINDEX. Refer to 8.3.1, "Unloading the existing database" on page 96, 9.1.3, "Unloading the HIDAM database" on page 136, and 14.1.4, "Unloading HALDB partitions and databases" on page 246.
DFSURGL0	HD Reorganization Reload	Applies to PHDAM, PHIDAM, and PSINDEX. Refer to 8.3.8, "Reloading the HALDB database" on page 99, 9.1.10, "Loading HALDB database" on page 141, and 14.1.6, "Reloading HALDB partitions and databases" on page 249.
DFSURPR0	Prereorganization	Refer to 6.3, "Database Prereorganization utility" on page 76, and 9.1.9, "Initializing HALDB partitions" on page 140.



Utility	Description	Comment
DFSUDMP0	Image Copy	Refer to Chapter 13, “Backup and recovery” on page 227.
DFSUICP0	Online Image Copy	
DFSUDMT0	Database Image Copy 2	
DFSUCUM0	Change Accumulation	
DFSURDB0	Database Recovery	
DFSBBO00	Batch Backout	

Database Recovery Control (DBRC) is required for execution of any utility operating on a HALDB. Each utility checks for the presence of DBRC. If DBRC is not present, the utility issues an error message and terminates.

Image copy utilities reject any attempt to image copy HALDB ILDSs or PHIDAM prime index data sets. Recovery utilities reject any attempt to recover HALDB ILDSs or PHIDAM prime index data sets. Both Image Copy and Recovery utilities can only run against a particular data set of a HALDB partition.



# Abbreviations and acronyms

<b>ACB</b>	Application Control Block	<b>HLQ</b>	High-level Qualifier
<b>APAR</b>	Authorized Program Analysis Report	<b>HSAM</b>	Hierarchic Sequential Access Method
<b>APF</b>	Authorized Program Facility	<b>HTML</b>	Hyper Text Markup Language
<b>BMP</b>	Batch Message Program	<b>HTTP</b>	Hyper Text Transfer Protocol
<b>CDS</b>	Control Data Set	<b>IBM</b>	International Business Machines Corporation
<b>CI</b>	Control Interval	<b>IFP</b>	Ims Fast Path Program
<b>CICS</b>	Customer Information Control System	<b>ILDS</b>	Indirect List Entry Data Set
<b>DASD</b>	Direct Access Storage Device	<b>ILE</b>	Indirect List Entry
<b>DB2</b>	Database 2	<b>ILK</b>	Indirect List Key
<b>DBA</b>	Database Administrator	<b>IMS</b>	Information Management System
<b>DBCTL</b>	Database Control	<b>IRLM</b>	Internal Resource Lock Manager
<b>DBD</b>	Database Description	<b>ISPF</b>	Interactive Systems Productivity Facility
<b>DBDS</b>	Database Data Set	<b>ITSO</b>	International Technical Support Organization
<b>DBRC</b>	Data Base Recovery Control	<b>IVP</b>	Installation Verification Program
<b>DEDB</b>	Data Entry Database	<b>JBP</b>	Java Batch Processing Region
<b>DL/I</b>	Data Language/i	<b>JCL</b>	Job Control Language
<b>DLI/SAS</b>	DL/I Separate Address Space	<b>JES</b>	Job Entry Subsystem (Jes2 Or Jes3)
<b>EBCDIC</b>	Extended Binary Coded Decimal Interchange Code	<b>JMP</b>	Java Message Processing Region
<b>EPS</b>	Extended Pointer Set	<b>KR</b>	Key Range
<b>EQE</b>	Error Queue Element	<b>KSDS</b>	Key Sequenced Data Set
<b>ESDS</b>	Entry Sequenced Data Set	<b>LPCK</b>	Logical Parent Concatenated Key
<b>FSEAP</b>	Free Space Element Anchor Point	<b>MPP</b>	Message Processing Program
<b>HALDB</b>	High Availability Large Database	<b>ODBA</b>	Open Database Access
<b>HDAM</b>	Hierarchical Direct Access Method		
<b>HIDAM</b>	Hierarchical Indexed Direct Access Method		
<b>HISAM</b>	Hierarchical Indexed Sequential Access Method		

<b>ORS</b>	Online Recovery Service
<b>OSAM</b>	Overflow Sequential Access Method
<b>PCB</b>	Program Communication Block
<b>PDS</b>	Partitioned Data Set
<b>PDU</b>	Partition Definition Utility
<b>PHDAM</b>	Partitioned Hierarchical Direct Access Method
<b>PHIDAM</b>	Partitioned Hierarchical Indexed Direct Access Method
<b>PID</b>	Partition Identification
<b>PROCOPT</b>	Processing Option
<b>PSB</b>	Program Specification Block
<b>PSINDEX</b>	Partitioned Secondary Index
<b>PTF</b>	Program Temporary Fix
<b>RAP</b>	Root Anchor Point
<b>RBA</b>	Relative Byte Address
<b>RSR</b>	Remote Site Recovery
<b>SMP/E</b>	System Modification Program/extended
<b>SVL</b>	Silicon Valley Laboratories
<b>TB</b>	Twin Backward (Pointer)
<b>TSO</b>	Time Sharing Option
<b>VOLSER</b>	Volume Serial (Number)
<b>VSAM</b>	Virtual Storage Access Method
<b>WWW</b>	World Wide Web
<b>XRF</b>	Extended Recovery Facility

# Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this publication.

## IBM Redbooks

For information on ordering these publications, see “How to get IBM Redbooks” on page 300. Note that some of the documents referenced here may be available in softcopy only.

- ▶ *A DBA's View of IMS Online Recovery Service*, SG24-6112
- ▶ *Ensuring IMS Database Integrity Using IMS Tools*, SG24-6533
- ▶ *IMS DataPropagator Implementation Guide*, SG24-6838
- ▶ *IMS e-business Connectors: A Guide to IMS Connectivity*, SG24-6514
- ▶ *IMS Installation and Maintenance Processes*, SG24-6574
- ▶ *IMS in the Parallel Sysplex, Volume I: Reviewing the IMSplex Technology*, SG24-6908
- ▶ *IMS Primer*, SG24-5352
- ▶ *IMS Version 7 High Availability Large Database Guide*, SG24-5751
- ▶ *IMS Version 8 Implementation Guide - A Technical Overview of the New Features*, SG24-6594
- ▶ *IMS Version 7 Java Update*, SG24-6536
- ▶ *IMS Version 7 Performance Monitoring and Tuning Update*, SG24-6404
- ▶ *IMS Version 7 Release Guide*, SG24-5753
- ▶ *Using IMS Data Management Tools for Fast Path Databases*, SG24-6866

## Other publications

These publications are also relevant as further information sources:

- ▶ *IMS Image Copy Extensions for OS/390, V1R1, User's Guide*, SC27-0924
- ▶ *IMS Index Builder for Z/OS, V2.R User's Guide*, SC27-0930
- ▶ *IMS High Performance Load for OS/390, V1R1, User's Guide*, SC27-0938

- ▶ *IMS High Performance Unload for OS/390, V1R1, User's Guide*, SC27-0936
- ▶ *IMS Version 7 Customization Guide*, SC26-9427
- ▶ *IMS Version 8 Administration Guide: Database Manager*, SC27-1283
- ▶ *IMS Version 8 Command Reference*, SC27-1291
- ▶ *IMS Version 8 Customization Guide*, SC27-1294
- ▶ *IMS Version 8 Utilities Reference: Database and Transaction Manager*, SC27-1308

## Online resources

These Web sites and URLs are also relevant as further information sources:

- ▶ IMS home page  
<http://www.ibm.com/ims>
- ▶ IBM Redbooks home page  
<http://www.ibm.com/redbooks>
- ▶ IBM Data Management Tools home page  
<http://www.ibm.com/software/data/db2imstools/>
- ▶ Technical documents: Flashes, presentations, hints, tips, and Technotes  
<http://www.ibm.com/support/techdocs/>

## How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

[ibm.com/redbooks](http://ibm.com/redbooks)

# Index

## Symbols

/DBD 214, 221  
/DBDUMP 218  
/DBR 214, 221  
/DIS 214  
/LOCK 214, 220  
/START 214, 219, 221  
/START DB ALL 221  
/STOP 214, 220  
/SX 92–93  
/UNLOCK 214, 221

## Numerics

5655-B01 (IMS Version 7) ii  
5655-C56 (IMS Version 8) ii  
5655-E50 (Online Recovery Service Version 1) 231  
5655-I01 (HALDB Conversion Aid Version 1) 152  
5655-K47 (HALDB Conversion and Maintenance Aid Version 2) 151

## A

ACBGEN 130, 142, 147  
ACCESS 91, 138, 219  
access intent 219  
ACCESS operand 26  
ALL keyword 221  
ALLOC 40, 267  
ALLOCATE 139  
allocation 35  
APAR 78, 100–101, 250, 291–292  
authorization 35

## B

backout needed 37  
Batch Backout 241  
Bidirectional logical relationships 10  
BLDSNDX 292  
block size 260  
BMP 10, 178, 221–222  
Buffer subpools 25  
BYTES 92–93

## C

Change Accumulation 230–231  
change accumulation group 42  
change version number 33  
CHANGE.DB 35–36, 68, 75, 78, 266, 292  
CHANGE.DBDS 36–37, 64, 68, 70  
CHANGE.PART 36–37, 68–69, 292  
CI size 260  
CICS 198  
command 213  
CONST 92

## D

data capture exit 258  
Data Capture Exit routine 24  
data conversion user exit 258  
Data Conversion User Exit routine 24  
data set name prefix 18, 33, 259  
database  
    PHIDAM 7  
Database Change Accumulation 230  
database data set group 42  
Database Data Set Initialization 99  
database group 42  
Database Image Copy 228–229  
Database Image Copy 2 228–229  
DATABASE macro 25  
Database Partition Data Set Initialization 76  
Database Partition Data Set Initialization utility 74  
Database Prefix Resolution 76  
Database Prereorganization 99  
Database Prereorganization utility 74–76  
Database Recovery 228, 231, 233  
Database Scan 76  
Database structure 11  
DATASET 91, 138  
DATASET statement 26  
DB record 32  
DBCTL 292  
DBCTL thread 222  
DBD 91, 144  
DBDGEN 7–8, 12, 21–22, 24, 45, 49, 62, 90, 130, 144

- DBDS record 32
- DBORG 40
- DBRC 9, 31
- DBRC authorization for HALDB 35
- DBRC CHANGE commands 260
- DBRC command 35, 261, 263, 292
- DBRC command authorization 47
- DBRC commands 35, 45
- DBRC group 42
- DDNAME 91
- DEFINE 98–99
- DELETE 38
- DELETE.DB 38, 71, 292
- DELETE.DBDS 38
- DELETE.PART 38, 71, 292
- DFS0565I 222
- DFSBB000 241
- DFSDBUX1 258
- DFSMAID0 93, 130, 139, 260, 268
- DFSMDA 9, 35
- DFSMSdss 229
- DFSPREC0 41, 142
- DFSPSE00 84–85
- DFSSRT01 142
- DFSSRTnn 143
- DFSUCUM0 231
- DFSUDMP0 228–229
- DFSUDMT0 228–229
- DFSUICP0 228
- DFSURDB0 228, 231
- DFSURGL0 99, 141
- DFSURGU0 136
- DFSURGUP macro 259
- DFSVSAMP 237
- DFSWRK01 142
- DFSWRKnn 143
- DSGROUP 91–92, 138
- DSGROUP parameter 26
- DSPDBHRC macro 32–33
- DSPDSHRC macro 34
- DSPPTNRC macro 34
- DSPUPJCL 41
- DUMP 229
- dynamic allocation 26, 31, 35

## E

- EPS 14
- EQE 37

- error queue element 37
- extended pointer set 14
- Extended Recovery Facility (XRF) 3

## F

- FIELD 93
- free space 91, 260
- free space element anchor point 11, 94
- FSEAP 11, 94

## G

- GENJCL 40, 227
- GENJCL.CA 231
- GENJCL.IC 230
- GENJCL.OIC 230
- GENJCL.RECOV 233, 235
- GENJCL.USER 41
- GENMAX 37
- Global DMB number 32
- group 42

## H

- H 91
- HALDB Conversion and Maintenance Aid 87
- HALDB Database Data Set Initialization 99
- HALDB master 9
- HALDB Migration Aid utility 260, 268
- HALDB Migration Aid utility (DFSMAID0) 23
- HALDB Partition Initialization utility 292
- HB 91
- HD Reload 90, 249, 265–267
- HD Reorganization Reload 99, 141
- HD Reorganization Unload 136, 147
- HD Unload 90, 249, 259, 265
- HDAM 6
- HIDAM 6–7, 92, 130
- HIER 91
- HIERBWD 91
- high-used-RBA 74
- HIKEY 63

## I

- IBM High Performance Load 74
- IDCAMS 139
- IEC-161I 248
- IEC161I 100, 248
- IFP 193, 222



IHCXHAL 155  
 ILDS 11, 14–15, 34, 37, 98, 142, 259, 269, 292  
 ILDSMULTI 100–101, 250, 292  
 ILDSMULTI control statement 250  
 ILE 14  
 ILK 12, 14  
 image copy 130, 147, 228, 267  
 Image Copy 2 228–229  
 Image Copy Extensions 228  
 image copy needed 34, 37, 75–76, 100, 142, 174, 187, 197, 261, 263–264, 266–267  
 IMS HALDB Conversion and Maintenance Aid 151–152  
 IMS Image Copy Extensions 228  
 IMS Index Builder 259, 265, 269  
 IMS monitor 3  
 IMS Online Recovery Service 231  
 IMS Performance Analyzer 3  
 IMS/ESA Partition Support Product 147  
 IMS/ESA Partition Support Product (PDB) 6  
 Index/ILDS Rebuild utility 41  
 indirect list data set 11, 18, 282  
 indirect list entry 14  
 indirect list key 12, 14  
 INIT 36  
 INIT.DB 24, 32, 36, 62, 82, 292  
 INIT.DBDS 36  
 INIT.PART 24, 33–34, 36, 62, 64, 69, 292  
 IOBF statement 25  
 ISPF 46  
 IVP 292

**J**  
 JBP 10, 221–222  
 JMP 193, 222

**K**  
 key range 80  
 key ranges 10  
 KEYSTRNG 63, 65  
 KR control statement 95, 107, 134

**L**  
 LCHILD 92, 138  
 LIST 38  
 LIST.DB 38  
 LIST.DBDS 38

LIST.HISTORY 38  
 logical relationship 6, 10  
 logical relationships 5, 22

**M**  
 master database 22, 32, 36, 43, 49, 62  
 master database record 32  
 MAX 133  
 MAX control statement 95  
 MIGRATE=YES 90, 96, 101, 136, 147  
 Migration Aid 130, 139, 147  
 Migration Aid utility 260, 268  
 MIGRATX=YES 101, 142  
 MPP 178, 193, 222  
 multi-volume OSAM 140

**N**  
 naming conventions 17  
 NBR 131  
 NBR control statement 95  
 NOILDS 101, 250, 292  
 NOILDS control statement 250  
 NOTWIN 92

**O**  
 ODBA 193, 198  
 ODBA thread 222  
 Online Database Image Copy 228  
 Online Recovery Service 231  
 OPEN 219  
 ORS 231  
 OSAM 5, 22  
 OSAM block size 91, 260  
 OSAM sequential buffering 237

**P**  
 parallel processing 7  
 PART record 32  
 partition database data set record 32  
 partition database record 32–33  
 partition DBDS record 34  
 partition definition process 7  
 Partition Definition utility 8, 22, 24, 26, 33–35, 45–46, 62, 82, 97, 120, 138–139, 208, 260–261, 263  
 Partition Definition utility (PDU) 22  
 partition high key 34

- partition ID 11, 14–15, 34, 267
- partition init needed 33, 37, 67, 75, 248, 260, 274
- Partition init needed flag 34
- partition initialization 35, 73, 90
- Partition Initialization utility 75
- partition name 18, 43, 267
- partition record 32–33
- partition selection 10, 16, 79
- partition selection exit 17, 36
- Partition selection exit routine 32
- partition selection exit routine 11, 75, 79, 81
- partition string 34, 37
- partition structure initialization 83
- partition structure modification 83
- partition structure termination 83
- partitioned HDAM 3, 11
- partitioned HIDAM 3
- partitioned secondary index 3
- PARTSEL 63
- PCB label 292
- PCB name 292
- PDATA 42
- PDB 147
- PDU 46, 97, 139
- PHDAM 3, 11, 14–15, 18, 74, 80, 91
- PHIDAM 3, 7, 11, 14–15, 18, 74, 80, 91–92, 130
- PHIDAM primary index 37
- physical pairing 10, 15
- PILDS 42
- PINDEX 42
- PINIT 34, 37, 260, 263, 266
- PQ35893 62, 292
- PQ36991 100–101, 250, 292
- PQ38626 292
- PQ48421 292
- PQ49638 78, 292
- PQ52858 62, 292
- PQ54227 100–101, 250, 292
- PQ55002 78, 292
- PQ55840 292
- PQ56463 292
- PQ57313 292
- PQ58600 292
- PQ59888 62, 292
- PQ61206 292
- PQ63751 130
- PQ65486 292
- PQ65489 292
- PQ68573 130

- Prefix Resolution 188, 244
- Prefix Resolution utility 244
- Prefix Update 188
- Prereorganization utility 248
- primary index 26, 34, 142
- prohibit further authorization 36
- PSINDEX 3, 11, 15, 18, 80, 91–92
- PSNAME 91
- PSNAME parameter 26
- PTR 91
- PTR=INDX 92
- PTR=SNGL 92
- PTR=SYMB 92

## R

- randomization parameter 260
- randomizing module 260
- RAP 94, 120, 145, 184
- RBA 12–15, 29, 102, 126, 187, 252–254, 256
- read only 37
- RECON 31, 46, 90
- recovery group 42
- recovery needed 34, 37
- RECOVPD 37
- Redbooks Web site 300
  - Contact us xv
- Remote Site Recovery (RSR) 3
- REORG 40
- reorganization 5–7, 17
- reorganization number 11, 14–15
- REUSE 37, 98–99
- RKSIZE 92
- RMNAME 91
- RMNAME parameter 26
- root anchor point 94

## S

- SAMPLE control statement 95
- sample partition selection exit 84
- SCAN 91
- SDFSISRC 238
- secondary index 5–6, 13, 16–17, 22, 87, 90, 251
- Secondary Index Database Maintenance Exit routine 24
- secondary indexes 6
- security 47
- SEED 96
- SEGM 91, 138

Segment Edit/Compression Exit routine 24  
selection exit 10–11  
self healing 245, 251  
self healing pointer 269  
share level 36  
shared secondary index 92  
single partition processing 292  
subsequence 92–93  
symbolic keyword 42  
symbolic pointers 92  
symbolic pointing 17  
system definition 25

## **T**

T 91  
TB 91  
Timestamp recovery 232  
timestamp recovery 16  
TSO 46  
TSO %DFSHALDB 46  
TWIN 91  
TWINBWD 91  
TYPE 215  
TYPEFP 63  
TYPEIMS 63  
TYPHALDB 38, 63

## **U**

Unconditional partition initialization 292

## **V**

VSAM 5, 22  
VSAM CI size 260  
VSRBF statement 25

## **X**

XDFLD 92





## The Complete IMS HALDB Guide All You Need to Know to Manage HALDBs







**Redbooks**

# The Complete IMS HALDB Guide

## All You Need to Know to Manage HALDBs

**Examine how to migrate databases to HALDB**

**Explore the benefits of using HALDB**

**Learn how to administer and modify HALDBs**

This IBM Redbook describes the High Availability Large Database (HALDB) capability available with IMS. IMS HALDB was introduced with IMS Version 7. It allows IMS databases to grow to almost unlimited sizes while providing increased availability. This book updates *IMS Version 7 High Availability Large Database Guide*, SG24-5751, as well as adding topics that were not covered in the previous book.

This publication provides a broad explanation of HALDB and its uses. Specific areas covered include:

- ▶ HALDB overview, definition, and structure
- ▶ Migration from non-HALDB databases
- ▶ Application considerations
- ▶ HALDB database administration

This publication documents our hands-on experience in a test environment. It includes migration and administration examples. Some IBM Data Management Tools for IMS are also discussed in this publication. Special emphasis is given to the IMS HALDB Conversion and Maintenance Aid product. Examples of the use of these tools are provided.

**INTERNATIONAL  
TECHNICAL  
SUPPORT  
ORGANIZATION**

**BUILDING TECHNICAL  
INFORMATION BASED ON  
PRACTICAL EXPERIENCE**

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

**For more information:**  
[ibm.com/redbooks](http://ibm.com/redbooks)