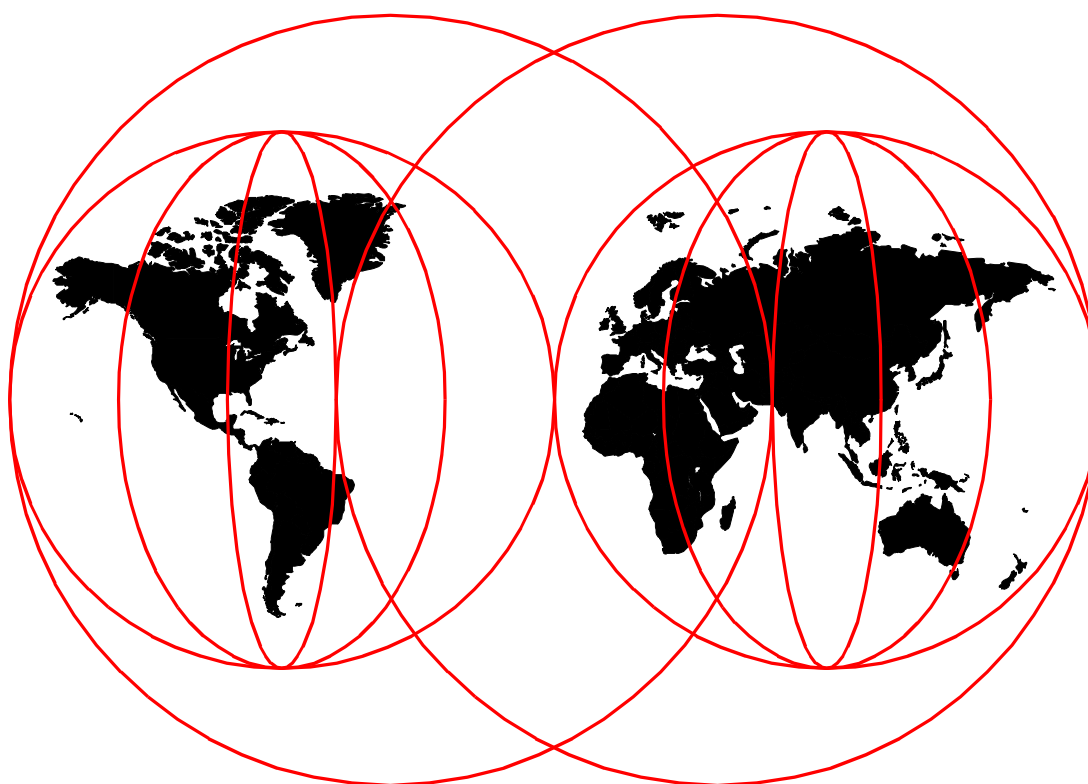


IMS Primer

Rick Long, Mark Harrington, Robert Hain, Geoff Nicholls



International Technical Support Organization

www.redbooks.ibm.com



International Technical Support Organization

SG24-5352-00

IMS Primer

January 2000

Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix A, "Special notices" on page 259.

First Edition (January 2000)

This edition applies to IBM Information Management System (IMS), Transaction and Database Server for System 390 Program Number 5697-B89 for use with the OS/390 operating systems.

Comments may be addressed to:

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

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© Copyright International Business Machines Corporation 2000. All rights reserved

Note to U.S Government Users - Documentation related to restricted rights - Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

Contents

Contents	iii
Figures	xi
Tables	xv
Preface	xvii
The team that wrote this redbook	xvii
Comments welcome	xix
<hr/>	
Part 1. Overview of IMS	1
Chapter 1. Introduction	3
1.1 IMS product	3
1.2 Overview of the IMS product	3
1.2.1 IMS Transaction Manager	4
1.2.2 IMS Database Manager	5
1.2.3 IMS system services	5
1.2.4 IMS and OS/390 operating systems	5
1.3 IMS Transaction Manager	6
1.3.1 Network access to IMS/TM	6
1.3.2 IMS Transaction Manager messages	6
1.3.3 Connecting to other IMS and CICS systems	6
1.4 IMS Database Manager	7
1.4.1 Functions of IMS Database Manager	7
1.4.2 Implementation of IMS databases	7
1.4.3 Full Function IMS DB (DL/1 DB)	8
1.4.4 Fast Path Data Entry Database (DEDB)	9
1.4.5 IMS and DB2	9
1.5 Additional availability and recovery features	9
1.5.1 Database Recovery Control (DBRC)	9
1.5.2 Additional features for increased availability (XRF and RSR)	10
1.6 Description of XRF and RSR	10
1.6.1 Extended Recovery Facility (XRF)	10
1.6.2 Remote Site Recovery (RSR)	11
Chapter 2. IMS and OS/390	13
2.1 Structure of IMS subsystems	13
2.1.1 IMS control region	13
2.1.2 IMS system dependent regions	15
2.1.3 Application dependent regions	16
2.1.4 Batch application address space	18
2.1.5 Internal Resource Lock Manager (IRLM)	19
2.2 Running of IMS subsystems	20
2.3 Running multiple IMS systems on one OS/390 system	21
2.3.1 IMS subsystems	21
2.3.2 Address Spaces	22
2.3.3 Starting application dependent regions	22
2.4 Use of OS/390 services	23
2.4.1 MVS TCP/IP	23
2.4.2 APPC/MVS	24

2.4.3 Security server for OS/390 (RACF)	24
2.4.4 Transaction server for OS/390 (CICS)	25
2.5 Other hardware/operating system platforms	26
Chapter 3. IMS TM and DB general information	27
3.1 IMS startup	27
3.2 IMS shutdown	28
3.3 Logging	28
3.4 Security	28
3.5 IMS generation	28
3.6 IMS recovery	28

Part 2. IMS Transaction Manager	29
--	-----------

Chapter 4. The IMS control region	31
4.1 The IMS message	31
4.2 An IMS transaction flow	32

Chapter 5. Processing input from a terminal	35
5.1 Input message types	35
5.2 Terminal types	36
5.3 Input message origin	36
5.4 Terminal input destination	36
5.5 Message queueing	37
5.5.1 Queue size and performance considerations	38
5.5.2 Multiple message queues	38
5.5.3 Shared Queues	39
5.5.4 Fast Path transactions	39
5.5.5 APPC triggered transactions	39
5.5.6 OTMA triggered transactions	40
5.5.7 Message scheduling	40
5.5.8 Transaction scheduling and priority	41
5.5.9 Scheduling conditions	42
5.5.10 Scheduling in a dependent region	42
5.6 Database processing intent	45
5.6.1 Scheduling a BMP	45
5.6.2 Shared Queues	45

Chapter 6. Fast-Path transactions	47
6.1 IMS Fast Path exclusive transactions	47
6.2 IMS Fast Path potential transactions	47

Chapter 7. Non-terminal related input	49
7.1 Inter-System Communications (ISC)	49
7.2 Multiple Systems Coupling (MSC)	49
7.3 Advanced Program-to-Program Communication (APPC)	50
7.4 Open Transaction Manager Access (OTMA)	51

Chapter 8. The master terminal	53
8.1 The primary master	53
8.2 The secondary master	54
8.3 Using the MVS console as master terminal	54
8.4 Extended MCS/EMCS Console Support	54

Chapter 9. Application program processing overview	55
9.1 MPP processing	55
9.2 Role of the PSB	56
9.3 DL/I message calls	56
9.4 Program isolation and dynamic logging	56
9.5 Internal resource lock manager (IRLM)	58
9.6 Application program abnormal termination	58
9.7 Conversational processing	58
9.8 Output Message Processing	59
9.9 Logging and checkpoint / restart	59
9.9.1 Logging	59
9.9.2 Checkpointing	59
9.10 Message Switching	59

Part 3. IMS Database Manager61

Chapter 10. Database basics	63
10.1 The database design process	63
10.1.1 Entities	63
10.1.2 Data attributes	63
10.1.3 Entity relationships	64
10.1.4 Application functions	64
10.1.5 Access paths	64
10.1.6 Normalization	65
10.2 What is a database ?	65
10.3 Why use a database ?	66
10.4 The database administrator role	67
Chapter 11. The IMS hierarchical database model	69
11.1 Basic segment types in a hierarchical data structure	70
11.2 Sequence fields and access paths	71
11.3 Additional access paths to segments	72
11.4 Logical relationships	72
11.5 Secondary indexing	76
Chapter 12. Implementation of the IMS database model	79
12.1 Segments, records, and pointers	80
12.2 Physical storage of the data	81
12.2.1 HDAM	83
12.2.2 HIDAM	86
12.2.3 Index databases	88
12.2.4 DEDB	89
12.2.5 GSAM	92
12.2.6 Sequential	93
12.3 Selecting database organization	93
12.3.1 When to choose HISAM	93
12.3.2 When to choose HDAM	94
12.3.3 When to choose HIDAM	95
12.4 Physical segment design	95
12.5 Operating system access methods	96
12.5.1 VSAM or OSAM	96
12.5.2 IMS and system managed storage (SMS)	98
12.6 IMS checkpoints: preserving application data integrity	98

12.7 Locking: sharing IMS data between multiple tasks	100
Chapter 13. Choosing the correct type of database	103
13.1 Applications suitable for Full Function (DL/I)	103
13.2 Applications suitable for Fast Path (DEDB)	103
13.2.1 Very large databases	104
13.2.2 High availability requirements	104
13.2.3 Highly active databases	105
13.2.4 Limited data lifetime	105
13.2.5 Extreme performance levels	105
13.2.6 DEDB: reduced I/O usage	106
13.2.7 DEDB: CPU utilization	106
13.3 Applications suitable for Fast Path	106
Chapter 14. Database reorganization processing	109
14.1 Why is reorganization necessary ?	109
14.2 When to reorganize	110
14.3 Monitoring the databases	112
14.4 Reorganization processing overview	113
14.5 The reorganization process description	113
14.5.1 Database unload processing	114
14.5.2 Defining databases	115
14.5.3 Database reload processing	115
14.6 Fast Path reorganization	121
Chapter 15. Database recovery processing	123
15.1 About this chapter	123
15.2 Overview of database recovery	123
15.2.1 When is recovery needed ?	123
15.2.2 Online programs	124
15.2.3 DLI batch update programs	124
15.3 The database utilities	124
15.4 Overview of backup/recovery utilities	125
15.4.1 Database image copy utility (DFSUDMP0)	126
15.4.2 Database change accumulation utility (DFSUCUM0)	127
15.4.3 Database recovery utility (DFSURDB0)	128
15.4.4 Database batch backout utility (DFSBB000)	129
Part 4. IMS application development	131
Chapter 16. Application programming overview	133
16.1 Overview	133
16.2 Program structure	133
16.2.1 Entry to application program	135
16.2.2 Termination	135
16.2.3 Calls to IMS	135
16.2.4 PCB mask	136
16.2.5 Status code handling	140
16.2.6 IMS control blocks	140
16.2.7 Generation of IMS control blocks	141
16.3 The IMS database application programming interface (API)	142
16.3.1 Get unique (GU)	142
16.3.2 Get next (GN)	142

16.3.3	Hold form of get calls	142
16.3.4	Insert	142
16.3.5	Delete	143
16.3.6	Replace	143
16.3.7	System service calls	143
16.4	The data communication PCB	143
16.4.1	The database PCB	144
16.4.2	Additional processing intent options	144
16.4.3	Application control block generation (ACBGEN)	145
16.4.4	IMS/DB2 resource translate table (RTT) assembly	145
Chapter 17. Application coding for IMS Transaction Manager		147
17.1	Application Program Processing	147
17.1.1	Role of the PSB	148
17.1.2	DL/I message calls	148
17.1.3	Application program abnormal termination	148
17.1.4	Conversational processing	149
17.1.5	Output message processing	149
17.1.6	Logging and checkpoint/restart	149
17.2	The data communication design process	150
17.2.1	Concepts of online transaction processing	150
17.2.2	Application characteristics	151
17.2.3	Terminal user characteristics	151
17.2.4	IMS characteristics	151
17.2.5	Transaction response time considerations	151
17.2.6	Choosing the right characteristics	152
17.2.7	Online program design	152
17.2.8	Basic screen design	154
Chapter 18. IMS message format service		157
18.1	Message format service overview	157
18.2	MFS and the 3270	159
18.3	Relationship between MFS control blocks	159
18.3.1	MFS control block chaining	159
18.3.2	Linkage between DFLD and MFLD	160
18.3.3	Linkage between LPAGE and DPAGE	161
18.3.4	Optional message description linkage	161
18.3.5	3270 Device considerations relative to control blocking linkage	162
18.4	MFS Functions	163
18.4.1	Input message formatting	163
18.4.2	Output message formatting	164
18.4.3	MFS formats supplied by IMS	169
18.5	MFS control statements	169
18.5.1	Relations between source statements and control blocks	170
18.5.2	MFS control block generation	170
18.5.3	MFS library maintenance	172
Chapter 19. Application coding for IMS Database Manager		173
19.1	Introduction to database processing	173
19.1.1	Interface to IMS	173
19.1.2	Status code handling	177
19.1.3	Sample presentation of a call	178
19.2	Processing against a single database structure	178
19.2.1	DL/I positioning concept	178

19.2.2	Retrieving segments	179
19.2.3	Updating segments	182
19.2.4	Calls with command codes.	185
19.2.5	Database positioning after DL/I call	187
19.2.6	Using multiple PCBs for one database.	188
19.2.7	Processing GSAM databases.	189
19.3	COBOL programming considerations.	190
19.4	PL/I programming considerations	191
19.5	Processing with logical relationships	192
19.5.1	Accessing a logical child in a physical database	193
19.5.2	Accessing segments in a logical database	193
19.6	Processing with secondary indices	194
19.6.1	Accessing segments via a secondary index	194
19.6.2	Secondary index creation.	196
19.7	Loading databases	196
19.7.1	Loading a database	196
19.7.2	Loading databases with logical relationships	197
19.7.3	Loading a database with secondary indices	198
19.8	Batch checkpoint/restart	200
19.8.1	Using the XRST and CHKP calls	201

Part 5. IMS system administration	205
--	------------

Chapter 20. Database recovery control (DBRC).	207
20.1 DBRC usage	207
20.1.1 DBRC options	207
20.1.2 DBRC configurations	208
20.1.3 Database authorization	209
20.1.4 Access intent	209
20.2 RECON data sets	210
20.2.1 Database related information	211
20.2.2 Subsystem.	211
20.2.3 Database name	211
20.2.4 RECON definition and creation	212
20.3 Initializing RECON data sets	212
20.4 Allocation of RECON data sets to subsystems.	213
20.5 Placement of RECON data sets.	214
20.6 RECON data set maintenance.	214
20.6.1 RECON backup	214
20.6.2 DELETE.LOG INACTIVE command.	214
20.6.3 LIST.RECON STATUS command	215
20.7 RECON reorganization	215
20.8 Reorganizing RECON data sets.	215
20.9 Recreating RECON data sets	216
20.10 PRILOG record size.	217
20.11 Summary of recommendations for RECON data sets.	219
Chapter 21. RECON record types	221
21.1 RECON records.	221
21.1.1 Control records	221
21.1.2 Log records	221
21.1.3 Change accumulation records	221
21.1.4 DBDS group records	221

21.1.5 Database records	222
21.2 RECON header record	222
21.3 RECON header extension record	222
21.4 DB record	223
21.5 DBDS record	224
21.6 SUBSYS record	225
21.7 DBDSGRP record	226
21.8 CAGRP record	227
21.9 CA record	228
21.10 PRILOG/SECLOG record	229
21.11 PRISLDS/SECSLDS record	230
21.12 PRIOLD/SECOLD record	231
21.13 LOGALL record	232
21.14 ALLOC record	233
21.15 IC record	234
21.16 REORG record	235
21.17 RECOV record	236
21.18 AAUTH record	236
21.19 Interim log records	237
Chapter 22. IMS logging	239
22.1 Checkpointing	239
22.2 IMS log buffers	239
22.3 Online log data sets (OLDS)	240
22.3.1 OLDS dual logging	240
22.3.2 Dynamic backout	240
22.3.3 Archiving	241
22.3.4 OLDS I/O errors	242
22.3.5 DBRC	242
22.3.6 Lack of OLDS	242
22.4 Write ahead data sets (WADS)	242
22.4.1 Dual WADS	243
22.4.2 WADS redundancy	243
22.5 System log data sets (SLDS)	243
22.6 Recovery log data sets (RLDS)	244
Chapter 23. IMS system generation process	245
23.1 Types of IMS generation	245
23.2 IMS generation macros	246
23.3 The IMS generation process	248
23.3.1 Stage1	249
23.3.2 Stage2	250
23.3.3 JCLIN	251
23.3.4 Re-apply SMP/E maintenance not accepted	251
23.3.5 IMS security maintenance utility generation	251
23.4 Automating the IMS system generation process	252
Chapter 24. IMS security overview	253
24.1 Background to IMS security	253
24.2 The security macro	253
24.3 Protecting IMS terminals	254
24.4 Protecting IMS commands	254
24.5 Protecting IMS transactions	256
24.6 Protecting IMS dependent region(s) and the IMS online system	256

24.7 Protecting IMS PSBs and online application programs	257
24.8 Protecting IMS control program and region application programs	257
Appendix A. Special notices	259
Appendix B. Related publications	261
B.1 International Technical Support Organization publications	261
B.2 Redbooks on CD-ROMs	261
B.3 Other publications	262
How to get ITSO redbooks	263
IBM Redbook Fax Order Form	264
Glossary	265
List of abbreviations	271
Index	273
ITSO redbook evaluation	277

Figures

1. Interfaces to IMS	4
2. Structure of IMS DB/DC subsystem	14
3. Structure of IMS DBCTL system	18
4. Structure of IMS batch region	19
5. Transmission, message, and segment relationship	32
6. A message segment	32
7. The IMS regions, and their control / data flow	33
8. Input message processing	35
9. Message queuing	37
10. Message scheduling	40
11. Transaction definition in IMS stage 1 input	41
12. MPR PROC statement example	43
13. /ASSIGN CLASS command example	43
14. Display active	44
15. Master terminal screen	53
16. Secondary master spool JCL	54
17. Basic MPP/BMP flow	55
18. Entities, attributes, and relationships	66
19. Hierarchical data structure	69
20. Segment types and their relationships	71
21. Entities and relationships with physical and logical databases mapped on . . .	73
22. Two Logically related physical databases, PARTS and ORDERS	74
23. Two additional logical DB's after relating PARTS and ORDER DB's	75
24. A database and its secondary index database	76
25. Elements of the physical implementation	79
26. Segment Layout	80
27. Database record and pointers	81
28. HDAM — database in physical storage	84
29. HDAM database — free space management	85
30. HIDAM database in physical storage	87
31. Overall structure of Fast Path DEDB	90
32. Database unload processing	114
33. Database reload only	116
34. Reload processing with secondary indices	117
35. Database reload with logical relationships	118
36. Database reload with secondary indices and logical relationships	120
37. Overview of recovery utilities	125
38. Image copy utility	126
39. Change accumulation utility	127
40. Recovery utility	128
41. Batch backout utility	129
42. Structure of an application program	134
43. Application PSB structure	137
44. On-line application PCB mask	137
45. DLI application PCB mask	138
46. Concatenated keys	139
47. Testing Status Codes	140
48. IMS control block generation	141
49. General MPP Structure and Flow	147
50. Message formatting using MFS	157

51. Overview of message format service	158
52. Chained control block linkage	159
53. Linkage between message fields and device fields	160
54. LPAGE -- DPAGE linkage	161
55. Optional message description linkage	162
56. MFS Input Formatting	163
57. MFS output formatting	165
58. An output message definition with one LPAGE	166
59. An output message definition with multiple pages	167
60. Creation of MFS control blocks	171
61. Testing status codes	177
62. Sample call presentation	178
63. Basic GU call	179
64. Unqualified GN call	180
65. Qualified GN call	181
66. GN call with qualified SSA	181
67. Basic REPL call	183
68. Basic DLET call	184
69. Basic ISRT call	185
70. SSA with D and P command codes	185
71. Sample path retrieve call	186
72. COBOL batch program	190
73. PL/I batch program structure	192
74. PSB with secondary index defined	195
75. GU call using a secondary index	195
76. Database load with logical relationships	198
77. Database load with secondary indices	199
78. Database load with logical relationships and secondary indices	200
79. Example of RECON data set definition	212
80. Dynamic allocation of RECON data sets	213
81. PRILOG record size calculation formula	217
82. PRILOG record size calculation formula example 1	218
83. PRILOG record size calculation formula example 2	218
84. HEADER record	222
85. HEADER RECON information	223
86. DB record	223
87. DB information	224
88. DBDS record	224
89. DBDS information	225
90. SUBSYS record	225
91. SUBSYS information	226
92. DBDSGRP record	226
93. DBDSGRP information	227
94. CAGRP record	227
95. CAGRP information	228
96. CA record	228
97. CA information	229
98. PRILOG/SECLOG record	229
99. PRILOG information	230
100. PRISLDS/SECLDS record	230
101. PRISLDS information	231
102. PRIOLDS/SECOLDS record	231
103. PRIOLD/SECOLD information	232

104.LOGALL record	232
105.LOGALL information	233
106.ALLOC record	233
107.ALLOC information	234
108.IC record	234
109.IC information	235
110.REORG record	235
111.REORG information.	235
112.RECOV record.	236
113.RECOV information.	236
114.AAUTH record	236
115.IMS log archive utility.	241
116.Summary of the two stages of system definition processing.	249

Tables

1. Comparison of XRF and RSR	12
2. Support for region types by IMS control region type	16
3. IMS procedure member names	21
4. Valid combinations of the EOS / EOM / EOD symbols	31
5. Database organization types	81
6. IMS access method availability by application address space type	82
7. Program return statements	135
8. IMS call argument list	136
9. PCB statement	144
10. DLI function descriptions	174
11. Segment access	174
12. Segment name, command code and qualifications	175
13. Relational operator values	176
14. GSAM status codes	189
15. Database load status codes	197
16. Types of IMS system definitions	245

Preface

This redbook is called a primer, as it is intended to be an introductory book to help familiarize the reader with the basics of IMS. Much of the original content of this book was in an early IMS manual called the IMS Primer, which was discontinued a number of years ago.

This redbook will help you understand some of the basic functions of IMS. It also tries to position IMS with regard to some of the other IBM products, and it gives a broad understanding of the architecture of IMS. The book is meant as an overview of the IMS product. It contains general information on many of the features of IMS.

The first part of the book contains an overview of the transaction and database managers. This entails describing the basic function of each. It also includes an overview of how IMS interfaces with other products like the OS/390 operating system.

The second part of the book provides a more detailed explanation of the IMS Transaction Manager. It covers a detailed explanation of message processing. It also contains an introduction to the application programming interface to the IMS system.

The third part of the book provides a more detailed explanation of the IMS Database Manager. It starts out with some basic database concepts and includes the hierarchical model and how it is implemented in IMS. It provides some information on database design and choosing the right database types. It also includes an explanation of the database backup, recovery, and reorganization utilities.

The fourth part of the book explains application programming within the IMS environments. This includes both transaction and database programming interfaces as well as the message format services (MFS).

The fifth part of the book explains some basic IMS administration functions. These include database recovery (DBRC), RECON record information, IMS logging and the IMS system generation procedure.

This edition applies to IBM Information Management System (IMS), Transaction and Database Server for System 390 Program Number 5697-B89 for use with the OS/390 operating systems.

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.

Rick Long is an IMS systems specialist at the International Technical Support Organization, San Jose Center. He writes extensively and teaches IBM classes worldwide on all areas of IMS. Before joining the ITSO 1 year ago, Rick worked in the DBDC Systems Programming department, IBM Global Services Australia, as an IMS systems programmer.

Mark Harrington is an IMS systems programmer working for IBM Global Services in the United Kingdom. He holds a degree in Computer Science from Portsmouth Polytechnic, UK. He has been working in IT for the past 22 years, the last 18 years on IBM S/370 and S/390 mainframes. He has worked as an application programmer, application designer, systems programmer, product installer, and database administrator, using IMS and CICS under both MVS and VSE operating systems. His main area of expertise is the IMS Database Manager, used with both IMS and CICS TP monitors.

Robert Hain is an IMS systems programmer, with IBM Global Services, Australia (IGSA), working with the Telstra Alliance, in Melbourne. He gained a Bachelor of Science, majoring in Computer Science from the University of Melbourne, then joined a large bank in Australia where he started working with IMS Fast Path. He joined the phone company Telstra 3-1/2 years ago, and then IGSA as part of the Telstra outsourcing arrangement. He has been an IMS systems programmer for over 13 years, working specifically as a specialist using the IMS Transaction Manager for most of that time. Tasks have included installation, maintenance, tuning, and problem diagnosis, as well as working in detail with IMS Fast Path, APPC, security, and many IMS exits.

Geoff Nicholls is an IMS specialist in the Worldwide IMS Technical Support Team, which is part of the IMS development organization. Previously, he was an IMS specialist with the International Technical Support Organization, San Jose Center. Geoff has a Science degree from the University of Melbourne, Australia, majoring in Computer Science. He worked as an application programmer and database administrator for several insurance companies before specializing in database and transaction management systems with Unisys and IBM. Since joining IBM in 1989, Geoff has worked extensively with IMS customers in Australia and throughout Asia.

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

Jim Boyle
IBM Australia

Phil Vasile, Steve Perry, Pete Sadler
IMS Development, Santa Teresa Lab, San Jose

Chloe Long
International Technical Support Organization, San Jose Center

Yvonne Lyon
International Technical Support Organization, San Jose Center

Elsa Martinez
International Technical Support Organization, San Jose Center

Comments welcome

Your comments are important to us!

We want our redbooks to be as helpful as possible. Please send us your comments about this or other redbooks in one of the following ways:

- Fax the evaluation form found in “ITSO redbook evaluation” on page 277 to the fax number shown on the form.
- Use the online evaluation form found at <http://www.redbooks.ibm.com/>
- Send your comments in an Internal note to redbook@us.ibm.com

Part 1. Overview of IMS

This part consists of three chapters:

1. An introduction to the IMS product and its components. Refer to Chapter 1, "Introduction" on page 3.
2. A discussion of IMS and how it relates to the OS/390 product and services. Refer to Chapter 2, "IMS and OS/390" on page 13.
3. A general look at a few of the IMS startup commands. Refer to Chapter 3, "IMS TM and DB general information" on page 27.

Chapter 1. Introduction

This chapter contains an overview of the entire IMS product. It will include both the data communication and database components. The following topics are covered:

1. 1.1, "IMS product" on page 3
2. 1.2, "Overview of the IMS product" on page 3
3. 1.3, "IMS Transaction Manager" on page 6
4. 1.4, "IMS Database Manager" on page 7
5. 1.5, "Additional availability and recovery features" on page 9
6. 1.6, "Description of XRF and RSR" on page 10

1.1 IMS product

IMS/ESA is an IBM program product that provides transaction management and database management functions for large commercial application systems. It was originally introduced in 1968. There are two major parts to IMS, a data communication manager (DC) and a Database Manager (DB).

IMS TM is a message-based transaction processor that is designed to use the OS/390 or MVS/ESA environment to your best advantage. IMS TM provides services to process messages received from the terminal network (input messages) and messages created by application programs (output messages). It also provides an underlying queueing mechanism for handling these messages.

IMS DB is a hierarchical database manager which provides an organization of business data with program and device independence. It has a built in data share capability.

It has been developed to provide an environment for applications that require very high levels of performance, throughput and availability. The development has been designed to make maximum use of the facilities of the operating system and hardware on which it runs, currently OS/390 on S/390 hardware.

1.2 Overview of the IMS product

IMS consists of three components, the Transaction Manager (TM) component, the Database Manager (DB) component, and a set of system services that provide common services to the other two components. The functions provided by these components are described in more detail in the following chapters.

The Transaction Manager and Database Manager components can be ordered and paid for separately if the functions of the other component are not required. The appropriate system services are provided for the component ordered.

As IMS has developed, new interfaces have been added to meet new business requirements. It is now possible to access IMS resources using a number of interfaces to the IMS components. IMS applications can also access databases managed by IBM's DB2 relational database manager.

IMS has been developed so that each new release of IMS is upwardly compatible, so investment in existing applications is preserved. To accommodate the changing requirements of IT systems, many new features have been added. This has also resulted in a number of IMS features being wholly or partially superseded by newer features normally providing better functionality. This should be kept in mind when looking at IMS documentation. The interfaces to IMS are pictured in Figure 1.

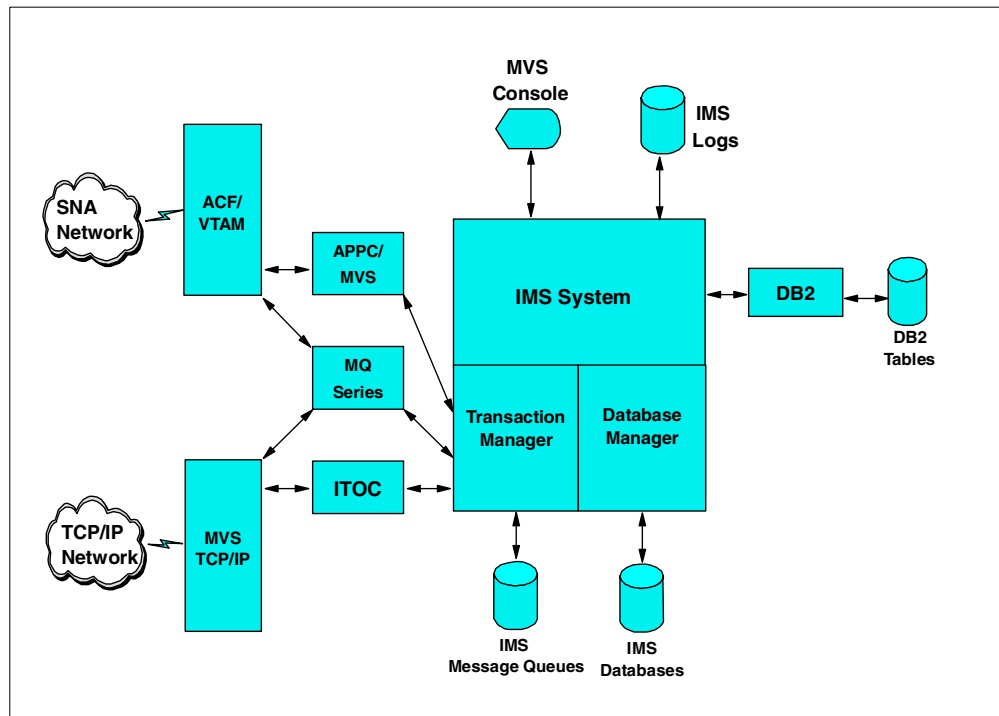


Figure 1. Interfaces to IMS

Applications written to use IMS functions can be written in a number of programming languages. Programming languages currently supported are Assembler, C, COBOL, Pascal, PL/I and REXX. The IMS resources are accessed by the application by calling a number of standard IMS functions. Applications access these functions through a standard application programming interface (API) for both the Transaction Manager and Database Manager components. This interface is normally referred to as Data Language I (DL/I).

1.2.1 IMS Transaction Manager

The IMS Transaction Manager provides users of a network with access to applications running under IMS. The users can be people at terminals or workstations, or other application programs, either on the same OS/390 system, on other OS/390 systems, or on other non-OS/390 platforms.

A transaction is a specific setup of input data that triggers the execution of a specific business application program. The message is destined for an application program, and the return of any results is considered one transaction.

Network access to IMS Transaction Manager was originally via IBM's systems, which evolved into the Network Architecture (SNA), as implemented in the VTAM program product. Also, there are now a number of ways to access IMS resources via networks using the Transmission Control Protocol / Internet Protocol (TCP/IP) network product.

1.2.2 IMS Database Manager

The IMS Database Manager provides a central point of control and access for the data (excluding DB2 Tables) that is processed by IMS applications. The Database Manager component of IMS supports databases using IMS's own hierarchic database model. It provides access to these databases from applications running under the IMS Transaction Manager, the CICS transaction monitor (now known as Transaction Server for OS/390), and OS/390 batch jobs.

It provides facilities for securing (backup/recovery) and maintaining the databases. It allows multiple tasks (batch and/or online) to access and update the data, while retaining the integrity of that data. It also provides facilities for tuning the databases by reorganizing and restructuring them.

The IMS databases are organized internally using a number of IMS's own internal database organization access methods. The database data is stored on disk storage using the normal operating system access methods.

1.2.3 IMS system services

There are a number of functions that are common to both the Database Manager and Transaction Manager:

- Restart and recovery of the IMS subsystems following failures.
- Security — controlling access to IMS resources.
- Managing the application programs — dispatching work, loading application programs, providing locking services.
- Providing diagnostic and performance information.
- Providing facilities for the operation of the IMS subsystems.
- Providing an interface to the other OS/390 subsystems that the IMS applications interface with.

1.2.4 IMS and OS/390 operating systems

IMS runs on S/370 and S/390 architecture IBM or compatible mainframes, running the MVS or OS/390 operating systems. In fact, there is a symbiotic relationship between IMS and OS/390. Both are tailored to provide the most efficient use of the hardware and software components.

An IMS subsystem runs in several address spaces in an OS/390 system. There is one controlling address space and several dependent address spaces providing IMS services and running IMS application programs.

For full details on the compatibility of IMS releases with versions of the operating system and associated products, refer to the current release planning guides. For IMS 5.1, this is *IMS/ESA Release Planning Guide*, GC26-8031. For IMS 6.1, this is *IMS/ESA Release Planning Guide*, GC26-8744.

1.3 IMS Transaction Manager

The IMS Transaction Manager can be ordered and installed with or without the Database Manager.

1.3.1 Network access to IMS/TM

IMS is written to interact with networks via IBM's Systems Network Architecture (SNA), as currently implemented using the VTAM program product. It can now also be accessed by networks using Transmission Control Protocol/ Internet Protocol (TCP/IP).

The Transaction Manager interacts directly with VTAM. Access via TCP/IP is made via another address space, this address space uses the IMS Open Transaction Manager Access (OTMA) function. The other address space can be either one available with IMS, such as the IMS TCP/IP OTMA Connector (ITOC), or another program product such as IBM's MQ Series. For further details on the options available for accessing IMS via TCP/IP, refer to the ITSO Publication *Connecting IMS to the World Wide Web: A Practical Guide to IMS Connectivity*, SG24-2220, and *IMS e-Business Connect Using the IMS Connectors*, SG24-5427.

1.3.2 IMS Transaction Manager messages

The network inputs and outputs to IMS Transaction Manager take the form of messages that are input/output to/from IMS and the "physical" terminals (or application programs) on the network (normally referred to as destinations).

These messages are processed asynchronously (that is, IMS will not always send a reply immediately, or indeed ever, when it receives a message, and unsolicited messages may also be sent from IMS). The messages can be of four types:

- Transactions — the data in these messages is passed to IMS application programs for processing
- Messages to go to another logical destination (network terminals, etc.)
- Commands for IMS to process
- Messages for the IMS APPC feature to process. As IMS uses an asynchronous protocol for messages, but APPC uses synchronous protocols (that is, it always expects a reply when a message is sent), the IMS TM interface for APPC has to perform special processing to accommodate this.

If IMS is not able to process an input message immediately, or cannot send an output message immediately, then the message is stored on a message queue external to the IMS system. IMS will not normally delete the message from the message queue until it has received confirmation that an application has processed the message, or it has reached its destination.

1.3.3 Connecting to other IMS and CICS systems

IMS has special protocols for connecting to other IMS systems, such as Multiple System Connection (MSC), and to other CICS and IMS systems, such as InterSystem Connection (ISC), which allow work to be routed to/from these other systems for processing.

The MSC connections can be via the network to other IMS systems on the same or other OS/390 systems, via channel-to-channel connections to the same or another channel attached OS/390 system, or via cross memory services to another IMS subsystem on the same OS/390 system.

The ISC links to other CICS or IMS systems is provided over the network via VTAM's LU 6.1 protocol.

1.4 IMS Database Manager

The IMS Database Manager can be ordered and installed with or without the IMS Transaction Manager

1.4.1 Functions of IMS Database Manager

A database management system (DBMS) provides facilities for business application transaction or process to access stored information. The role of a DBMS is to provide the following functions:

1. Allow access to the data for multiple users from a single copy of the data.
2. Control concurrent access to the data so as to maintain integrity for all updates.
3. Minimize hardware device and operating systems access method dependencies.
4. Reduce data redundancy by maintaining only one copy of the data.

The IMS Database Manager provides a central point for the control and access to application data. IMS provides a full set of utility programs to provide all these functions within the IMS product.

1.4.2 Implementation of IMS databases

IMS TM and IMS DBCTL both support multiple forms of enterprise databases, so that varied application requirements can be met by exploiting whichever database technology best suits the users' requirements.

These database technologies are:

IMS Database Often referred to as "DL/1 database" or colloquially as "Full Function databases"

IMS DEDBs The Data Entry database, often referred to colloquially as "Fast Path databases"

IMS MSDB Main storage databases

IBM Database2 DB2 provides for relational databases

IMS uses a hierarchical model for its database, described in more detail in Chapter 11, "The IMS hierarchical database model" on page 69. The data stored in the IMS databases is organized internally using a number of internal IMS access methods. Each of these access methods suits certain types of access to the database. The choice of the appropriate access method is discussed in detail Chapter 12, "Implementation of the IMS database model" on page 79.

No single technology is the best option for all applications — even though fashions may suggest that an organization standardize on only one database type. To do this, for example, to say that you wish to use only relational database technology (DB2), would preclude consideration of other technologies that, for suitable applications, would make massive savings in processing or application development costs — far in excess of the small additional cost of introducing DEDBs to your organization.

Each of the database implementations supported by IMS has different characteristics:

DL/1	DL/1 databases provide a hierarchically structured database, that can be accessed by record or sequentially, and by other sequences that were planned and provided for when the database was designed. DL/1 databases are limited in size to 4GB or 8GB per data set unless a portioning database product is used.
DEDBs	DEDBs are particularly suited for use where large databases, or very low processing costs are required, or when particularly high data availability or very high performance is required. DEDBs were originally part of a separately priced, optional feature. This results in the documentation and code being separate from that for the Full Function (FF) databases.
DB2	DB2 provides well for unstructured or unplanned access to data, and so provides flexibility in the support of future application requirements. However, DB2 usually has a significantly higher processing cost than any IMS database.

The IMS access methods are underpinned by the use of operating system access methods to store data on disk storage. The software access methods which IMS makes use of are:

- VSAM
- OSAM

1.4.3 Full Function IMS DB (DL/1 DB)

IMS Full Function Databases were design to support most types of database requirements. These can be used in a wide variety of applications. Most IMS applications make use of Full Function databases unless there are specific requirements for one of the other types of databases. The major characteristics of Full Function databases are:

- Small or large databases.
- Access to records via unique or non-unique keys.
- Many types of segments (up to 15 levels allowed).
- Records can be stored in key sequence, but it is not a requirement.

1.4.4 Fast Path Data Entry Database (DEDB)

The Data Entry Database (DEDB) was designed to support particularly intensive IMS database requirements, primarily in the banking industry, for:

- Large databases containing millions of records, extending well beyond the original 4GB database limits of DL/1 databases
- Access to each database record that can be achieved by access via a key field
- Lower processing costs per database record and per update than are required for DL/1 databases
- The capability to support higher transaction workloads than DL/1 can sustain, while maintaining the per-transaction cost advantages mentioned above
- Improved availability, with reduced requirements for database outage, especially for database maintenance activities such as database reorganizations
- Lower processing costs for particular types of processing, where data are inserted online and retrieved in batches for further processing, and eventually deleted in batches
- The possibility of eliminating transaction-related I/O from database processing.

All the above requirements were satisfied, while maintaining the conventional DL/1 application interface, so that application programming for the exploitation of DEDBs is little different from that for DL/1 databases.

1.4.5 IMS and DB2

IMS applications running in an IMS subsystem can also access data stored in a DB2 database. The updating of the DB2 tables is coordinated with the update to the IMS resources (databases and messages) to ensure all updates are consistently applied.

While the IMS databases provide high performance for the transaction processing environment, you may also want to perform ad-hoc queries on all or part of the data, more suitable to the relational database model implemented by DB2. The IBM Data Propagator product can be used to automatically duplicate data from IMS databases to DB2 tables.

1.5 Additional availability and recovery features

IMS provide many features to provide high availability and complete recovery of the IMS system in all operation environments.

1.5.1 Database Recovery Control (DBRC)

DBRC is that part of IMS which provides the recovery services so much a part of the IMS system. DBRC controls the allocation and use of all IMS logs in an online environment.

DBRC uses a control file, the Recovery Control (RECON) file to store the control information to fulfill these functions.

A more detailed description of DBRC is found in Chapter 20, “Database recovery control (DBRC)” on page 207.

1.5.2 Additional features for increased availability (XRF and RSR)

There are two additional features of IMS that can, optionally, be used to increase the availability of IMS systems and the data in IMS databases. Both rely on duplicating IMS subsystems and data on another OS/390 system.

The first of these is the extended recovery facility (XRF). XRF is delivered as an integral part of the IMS program product. It is intended to provide increased availability for IMS subsystems. There is an overhead, both in machine usage and support, in using XRF. However, if you have an application that can only tolerate minimal outages, then you may wish to consider XRF.

The second of these features is Remote Site Recovery (RSR). RSR is a separately priced component available with IMS. It provides similar facilities to XRF, but with some differences.

Both features rely on having another IMS subsystem, situated on another OS/390 system, that tracks the update activity of the primary IMS subsystem (only one for XRF, one or more for RSR) to provide a backup.

The differences between the two features is discussed elsewhere in this document, but to summarize:

- XRF is suitable for situations where you have a single IMS DB/DC system that requires very high system availability (greater than 99.5%). However the second OS/390 containing the tracking IMS system must be channel attached to the OS/390 System the first IMS is running on.
- RSR is suitable for situations where you have one or more IMS subsystems, running in a number of address spaces on a single OS/390 system, where you wish to minimize data loss in a failure situation, but can tolerate outages of around an hour. RSR uses network connections between the two OS/390 systems, so there are no restrictions on the distance separating them.

1.6 Description of XRF and RSR

XRF and RSR are features of IMS which provide increased availability for the IMS system and their related applications. They provide different types of functions but both increase the availability for IMS Systems.

1.6.1 Extended Recovery Facility (XRF)

XRF works by having a second, alternative, IMS system running. This alternative IMS system runs on a separate OS/390 image, which should be on a physically separate machine. This tracks the work on the active IMS system via the UNS log data sets. You want the ability to perform hardware maintenance and maintenance on other system software products without interrupting the availability of the IMS application.

XRF is delivered as an integral part of the IMS program product. It is intended to provide increased availability for IMS subsystems. There is an overhead, both in machine usage and support, in using XRF. However, if you have an application that can only tolerate minimal outages, then you may wish to consider XRF.

The principal drawbacks of XRF are:

- It will not protect against application errors. If the outage is caused by an application error, the same application message may be re-presented on the alternate IMS and cause it to fail.
- It will not, in itself, protect against network outages. You will have to plan for this separately.
- XRF does not support DB2 databases. However, if you are designing an application of this sort, it would be better to use IMS databases, particularly the Data Entry Database (DEDB). The DEDB has provisions for performing most database maintenance with the databases remaining available. It will also automatically maintain multiple copies of the data sets containing the data to guard against media failure.
- Some maintenance to the IMS software requires it to be applied to both the active and standby IMS systems at the same time.

So, while XRF can prevent most unplanned and planned outages, it cannot keep the IMS system available indefinitely. You will eventually have to have plan outages for software maintenance and upgrades, and some changes to the IMS configuration. IMS systems running with XRF have achieved continuous availability for business applications figures measured in years.

1.6.2 Remote Site Recovery (RSR)

RSR is a separately priced component available with IMS. It provides similar facilities to XRF, but with some differences.

RSR can track details of IMS full function databases, Fast path DEDB's, IMS/TM message queues and the current IMS/TM telecommunication network on an alternate machine. This machine is connected the machine with the active systems on by a network connection using the VTAM APPC protocol. The VTAM connection is between separate transport manager subsystems (TMS) on the active and tracking machines.

The transport manager subsystem on the active machine collects all log data from all IMS systems (DB/DC, DCCTL, DBCTL and batch) that are defined for RSR tracking and sends this data across to the tracking machine.

The TMS on the tracking machine receives this data and passes it to a single IMS DB/DC region. This processes the data and logs it using normal IMS logging. Depending on what level of tracking has been requested, the IMS region may also apply the updates to the IMS databases.

If there are any interruptions to the network connection, RSR will note the gaps in the logging and perform catch up processing when the link is re-established.

The IMS system on the tracking machine normally can only process input from the TMS. It only becomes a fully functioning system if it has to take over.

Not all databases are tracked. You define the databases that are to be tracked by specifying this when you define them to DBRC. Table 1 gives a comparison of the features of XRF and RSR.

Table 1. Comparison of XRF and RSR

RSF	XRF
Uses same physical log data sets and database data sets for active and tracking system	Uses completely separate log data sets and database data sets
Active and tracking system must be within channel attach distance of each other	Active and tracking systems are connected by network, only limit on separation is network response
Active and tracking systems must use IMS/TM	Active systems can be any system updating IMS resources DB/DB, TM only, DB only, or batch. The tracking system must be DB/dC.
One-to-one relationship between active and tracking system.	One tracking system tracks many active systems
All committed updates recorded on tracking system	Possible for gap in data at tracking system after unplanned takeover
Switching to/from alternative comparatively simple.	Takeovers more complex than XRF
Switches over to alternate in order of one minute.	Switch to alternate can take an hour or more.

To summarize:

- XRF is suitable for situations where you have a single IMS DB/DC system that requires very high system availability (greater than 99.5%). However the second OS/390 must be channel attached to the OS/390 system the first IMS is running on.
- RSR is suitable for situations where you have one or more IMS applications, which may run in a number of address spaces, and where you wish to minimize data loss in a failure situation, but can tolerate outages of around an hour. RSR uses network connections between the two OS/390 systems, so there are no restrictions on the distance separating them.

Chapter 2. IMS and OS/390

This chapter describes how IMS subsystems are implemented on an OS/390 system. It then gives an overview of IMS's use of OS/390 facilities. The following topics will be covered:

- 2.1, "Structure of IMS subsystems" on page 13
- 2.2, "Running of IMS subsystems" on page 20
- 2.3, "Running multiple IMS systems on one OS/390 system" on page 21
- 2.4, "Use of OS/390 services" on page 23
- 2.5, "Other hardware/operating system platforms" on page 26

2.1 Structure of IMS subsystems

This section describes the various types of OS/390 address spaces and their relationship with each other. The core of an IMS subsystem is the control region, running in one OS/390 address space. This will have a number of dependent address spaces running in other regions that provide additional services to the control region, or in which the IMS application programs run.

In addition to the control region, some applications and utilities used with IMS run in separate batch address spaces. These are separate to an IMS subsystem and its control region, and have no connection with it.

For historical reasons, some documents describing IMS use the term region to describe an OS/390 address space, for example, IMS Control Region. In this book we have used the term region wherever this is in common usage. You can take the term region as being the same as an OS/390 address space.

2.1.1 IMS control region

The control region (CTL) is an MVS address space that can be initiated through an MVS start command, or by submitting JCL.

The IMS control region provides the central point for an IMS subsystem. It provides the interface to the SNA network for the Transaction Manager functions, and the Transaction Manager OTMA interface for access to non-SNA networks. It provides the interface to OS/390 for the operation of the IMS subsystem. It controls and dispatches the application programs running in the various dependent regions.

The control region provides all logging, restart and recovery functions for the IMS subsystems. The terminals, message queues, and logs are all attached to this region, and the Fast Path database data sets are also allocated by the CTL region address space.

A type 2 supervisor call routine (SVC) is used for switching control information, message and database data between the CTL region, and all other regions, and back.

There are three different types of IMS control region, depending on whether the Database Manager and/or Transaction Manager components are being used. These three control region types are:

- **DB/DC** — This is a control region with both Transaction Manager and Database Manager components installed. It provides the combined functionality of both the other two types of control region. Note that when a DB/DC region is providing access to IMS databases for a CICS region, it is referred to in some documentation as providing DBCTL services, though it may in fact be a full DB/DC region and not just a DBCTL region.
- **DBCTL** — This is a control region with only the Database Manager component installed. This can provide IMS database functions to batch application programs connected to the IMS control region (BMPs, see below), to application transactions running in CICS Transaction Manager regions, and to other OS/390 address spaces (for example, DB2 stored procedures) via the Open DataBase Access (ODBA) interface.
- **DCCTL** — This type of control region has only the Transaction Manager component installed. It provides access to the IMS message queues for IMS applications running in the MPP, IFP and BMP application address spaces described Figure 2 below.

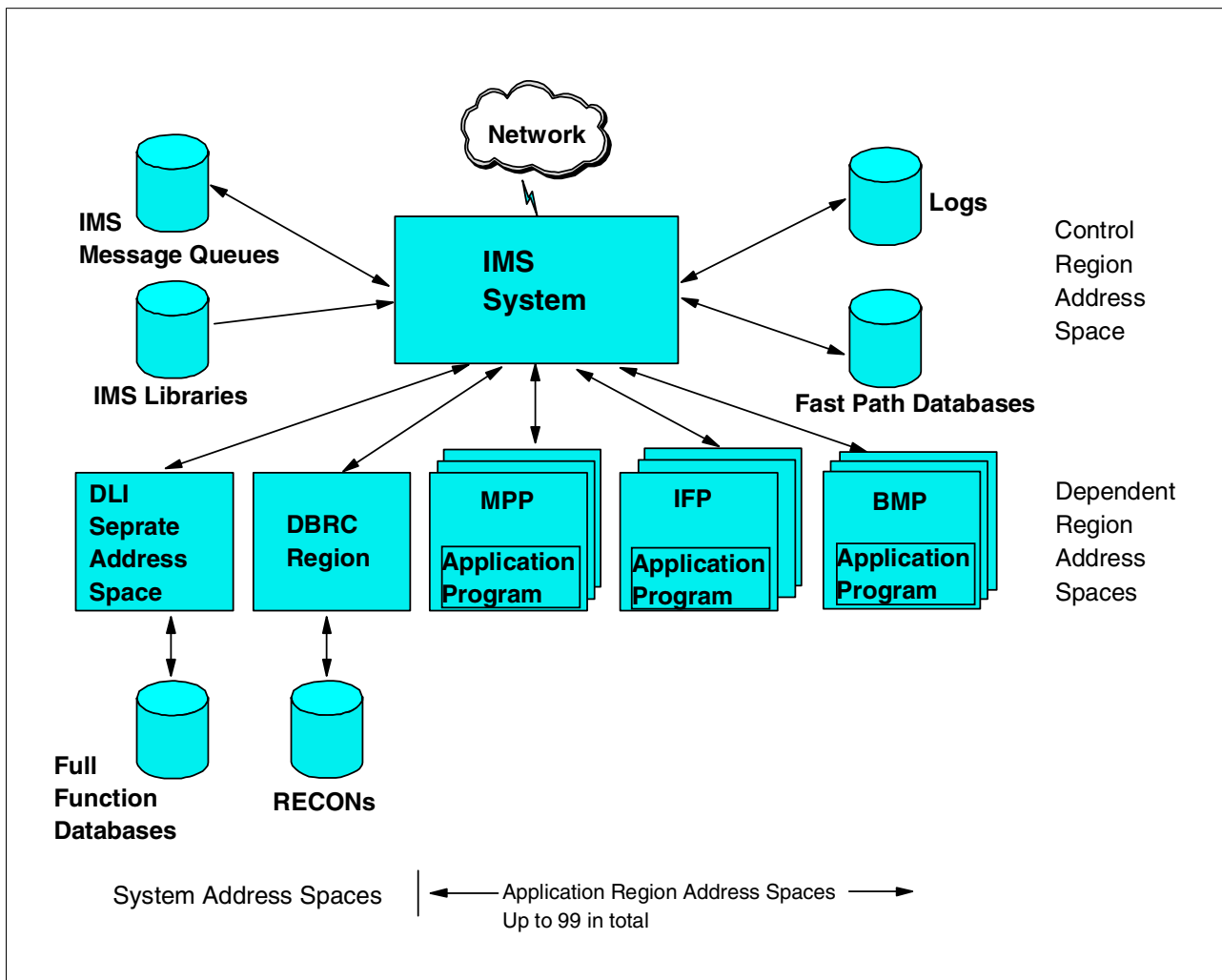


Figure 2. Structure of IMS DB/DC subsystem

In some of the IMS documentation, the above terms are also used to refer to what sort of IMS system is being generated during an MSGEN, that is, for what functions will code be generated into the IMS code libraries. This is distinct from the functions provided by a single IMS subsystem, which we are describing here.

2.1.2 IMS system dependent regions

The control region will have a number of dependent system address spaces (dependent regions) to provide some of the services of the IMS subsystem.

These dependent regions are automatically started by the IMS control region as part of its initialization, and the control region will not complete initialization until these dependent regions have started and connected to the IMS control region. Every IMS control region has a DBRC region. The other two dependent system address spaces are optional, depending on the IMS features used. For the DL/I, separate address space options can be specified at IMS initialization.

2.1.2.1 DBRC region

This address space contains the code for the DBRC component of IMS. It processes all access to the DBRC recovery control data sets (RECON). It also performs all generation of batch jobs for DBRC, for example, for archiving the online IMS log. All IMS control regions have a DBRC address space, as it is needed, at a minimum, for managing the IMS logs.

2.1.2.2 DLI separate address space (DLISAS)

This address space performs most data set access functions for the IMS Database Manager component (except for the fast path DEDB databases, described later). The FF database data sets are allocated by this address space. It also contains some of the control blocks associated with database access and some database buffers.

This address space is not present with a DCCTL system, since the Database Manager component is not present.

For a DBCTL control region, this address space is required, and always present.

For a DB/DC control region, you have the option of having IMS database accesses performed by the control region, or having the DB/DC region start a DL/I separate address space. For performance and capacity reasons, you would normally use a DLI separate address space.

2.1.2.3 Common queue server (CQS) address space

This is used by IMS DCCTL and DB/DC control regions only if they are participating in an OS/390 sysplex sharing of the IMS message queues. It provides access to the shared IMS message queues in the sysplex coupling facility, which replace the IMS messages queue data sets on DASD. The CQS address space is only available with IMS Version 6 onwards. See the ITSO publication *IMS/ESA Version 6 Shared Queues*, SG24-5088 for further details.

2.1.3 Application dependent regions

IMS provides dependent region address spaces for the execution of system and application programs that use the services provided by the IMS. The previously discussed region types (DBRC and DLISAS), as mentioned in “IMS system dependent regions” on page 15, are automatically started by the IMS control region.

These application dependent regions are started as the result of JCL submission to the operating system by the IMS CTL region, following an IMS command being entered.

Once they are started, the application programs are scheduled and dispatched by the control region. In all cases, the OS/390 address space executes an IMS region control program. The application program is then loaded and called by the IMS code.

There can be up to 999 application dependent regions connected to one IMS control region, made up of any combination of the following dependent region types:

- Message processing region (MPP)
- IMS Fast Path region (IFP)
- Batch message processing region (BMP)
- DBCTL thread (DBT)
- Other utility type regions, such as HSSP processing (BMH) and Fast Path utility program (FPU)

The combination of what region type can be used in the various types of IMS TL regions, can be found in Table 2.

Table 2. Support for region types by IMS control region type

Application Address Space type	DCCTL	DBCTL	DB/DC
MPP	Y	N	Y
IFP	Y	N	Y
BMP (txn oriented)	Y(1)	N	Y
BMP (batch)	N	Y	Y
Batch	N	N	N
DBT	N	Y	Y

(1) BMP attached to a DCCTL control region can only access the IMS message queues and DB2 databases.

2.1.3.1 Message processing region (MPP)

This type of address space is used to run applications to process messages input to the IMS Transaction Manager component (that is, online programs). The address space is started by IMS submitting the JCL as a result of an IMS command. The address space does not automatically load an application program but will wait until work becomes available.

There is a complex scheme for deciding which MPP to run the application program. We will give a brief description below. When the IMS dispatching function determines that an application is to run in a particular MPP region, the application program is loaded into that region and receives control. It processes the message, and any further messages for that transaction waiting to be processed. Then, depending on options specified on the transaction definition, the application either waits for further input, or another application program will be loaded to process a different transaction.

2.1.3.2 IMS Fast Path message region (IFP)

These address spaces also run application programs to process messages for transactions which have been defined as Fast Path transactions.

IMS Transaction Manager component, the applications are broadly similar to the programs that run in an MPP. Like MPPs, the IFP regions are started by the IMS control regions submitting the JCL as a result of an IMS command. The difference with IFP regions is in the way IMS loads and dispatches the application program, and handles the transaction messages. To allow for this different processing, IMS imposes restrictions on the length of the application data that can be processed in an IFP region as a single message.

IMS employs a user exit, which you have to write, to determine whether a transaction message should be processed in an IFP region, and which IFP region it should be processed in. The different dispatching of the transaction messages by the control region is called Expedited Message Handling (EMH). The intention is to speed the processing of the messages by having the applications loaded and waiting for input messages, and, if the message is suitable, dispatching it directly in the IFP region, bypassing the IMS message queues. Fast Path was originally a separately priced function available with IMS, intended to provide faster response and allow higher volumes of processing. It is now part of the IMS base product.

2.1.3.3 Batch message processing region (BMP)

Unlike the other two types of application dependent regions, the BMP is not started by the IMS control region, but is started by submitting a batch job, for example by a user via TSO, or via a job scheduler such as OPC/ESA. The batch job then connects to an IMS control region defined in the execution parameters. There are two types of applications run in BMP address spaces:

- Message Driven BMPs (also called transaction oriented), which read and process messages off the IMS message queue.
- Non-message BMPs (batch oriented), which do not process IMS messages.

BMPs have access to the IMS databases, providing that the control region has the Database Manager component installed. BMPs can also read and write to OS/390 sequential files, with integrity, using the IMS GSAM access method DBCTL Thread (DBT)

When a CICS system connects to IMS DBCTL, or to an IMS DB/DC system using DBCTL facilities, each CICS system will have a pre-defined number of connections with IMS. Each of these connections is called a thread. See Figure 3.

Although these threads are not jobs in their own right, from IMS's perspective, each thread appears just like another dependent region, and when CICS requires a DL/I call to IMS, the program will effectively be running in one of these DBT regions.

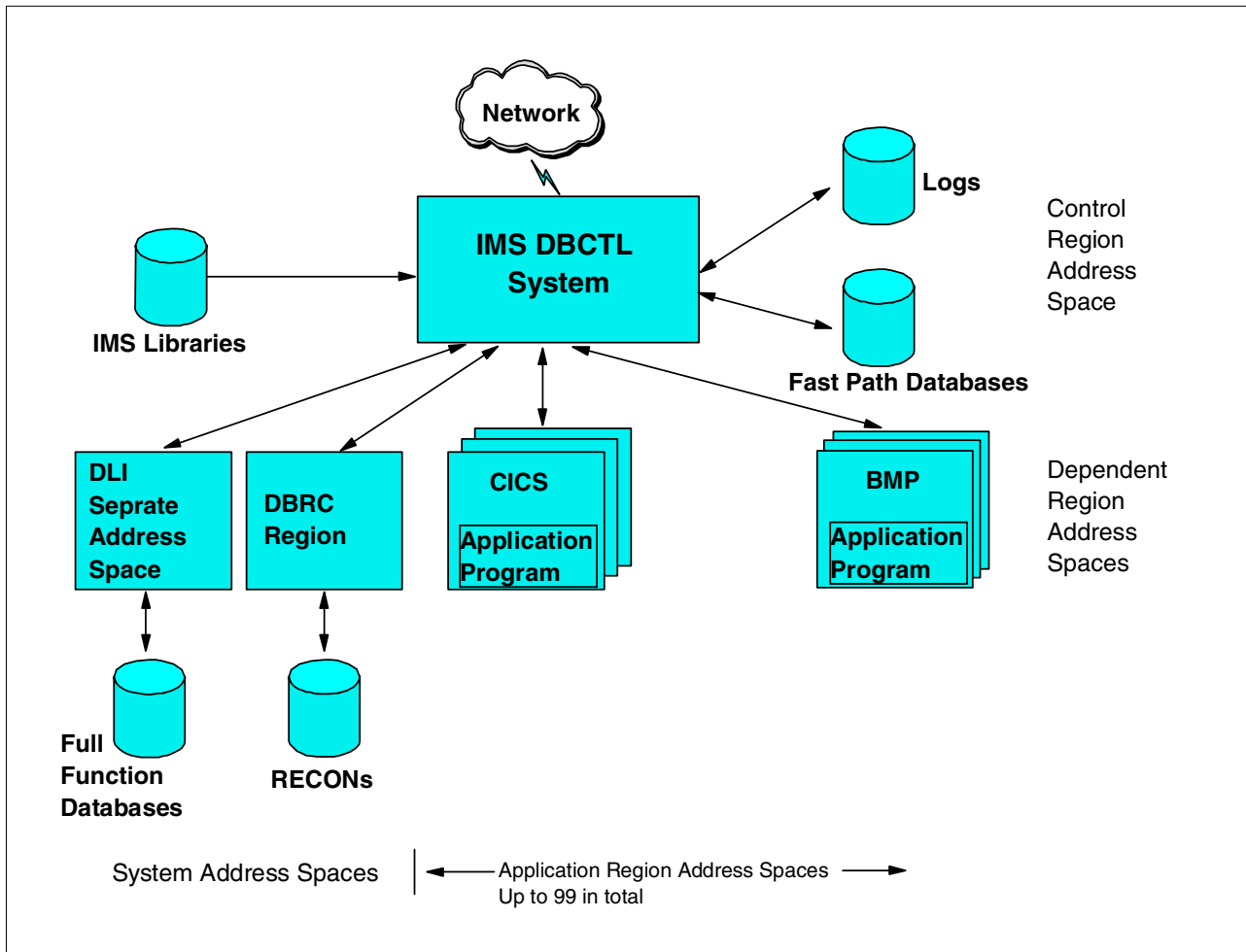


Figure 3. Structure of IMS DBCTL system

2.1.3.4 Other utility regions

Other types of regions are BMH, used for HSSP processing, and FPU, used for Fast Path utility programs. For further discussion on these, please refer to the appropriate level of the *IMS/ESA V6 Install Volume 2*, GC26-8737.

2.1.4 Batch application address space

In addition to the dependent application address spaces above, IMS application programs that only use IMS Database Manager functions can be run in a separate MVS address space, not connected to an IMS control region. This would normally be done for very long running applications, that perform large numbers of database accesses.

This is similar to a BMP, in that the JCL is submitted via TSO or a job scheduler. However, all IMS code used by the application resides in the address space the application is running in. The job executes an IMS region controller that then loads and calls the application. See Figure 4.

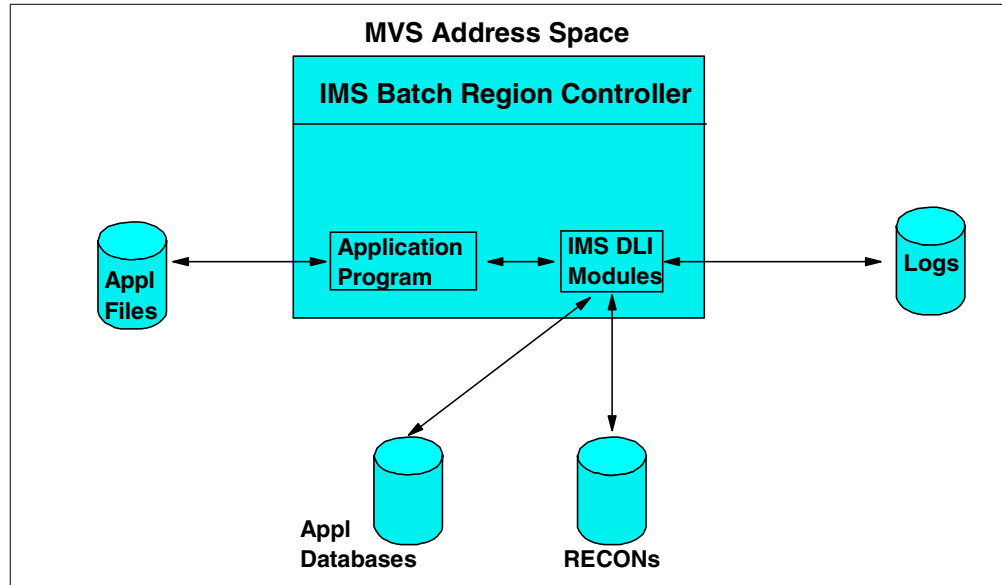


Figure 4. Structure of IMS batch region

The batch address space opens and reads the IMS database data sets directly. If there are requirements for other programs, either running via an IMS control region or in other batch regions, to access the databases at the same time, then see the discussion elsewhere in this book on methods of data sharing.

The batch address space writes its own separate IMS log. In the event of a program failure it may be necessary to take manual action (for example, submit jobs to run IMS utilities) to recover the databases to a consistent point (with dependent application address spaces this would be done automatically by the IMS control region). DBRC, if properly set up, can be used to track the IMS logs and ensure that correct recovery action is taken in the event of a failure.

An application can be written so that it can run in both a batch and BMP address space without change. Some reasons you may want to change programs between batch and BMP address spaces include length of run time, need of other applications to access the data at the same time, and your procedures for recovering from application failures.

2.1.5 Internal Resource Lock Manager (IRLM)

There is one final address space that is, optionally, used with IMS. This is the IRLM address space, and is only needed if you are going to use block level or sysplex data sharing for the IMS databases. The IRLM address space is started before the IMS control region, via the MVS start command. The IMS control region, if the start up parameters specify use of IRLM, connects to the IRLM specified on startup, and will not complete initialization until connected.

There is one IRLM address space running on each OS/390 system to service all IMS subsystems sharing the same set of databases. For more information on data sharing in sysplex environment, see *IMS/ESA Data Sharing in a Parallel Sysplex*, SG24-4303.

IRLM is delivered as an integral part of the IMS program product, though as mentioned, you do not have to install or use it unless you wish to perform block level or sysplex data sharing. IRLM is also used as the (only) lock manager for the DB2 database program product, and for DB2 you must install IRLM. Because the tuning requirements of IMS and DB2 are different, and conflicting, you are recommended **not** to use the same IRLM address space for IMS and DB2. Since the IRLM code is delivered with both the IMS and DB2 program products, and interacts closely with both these products, you may wish to install the IRLM code for IMS and DB2 separately (that is, separate SMP/E zones) so you can maintain release and maintenance levels independently. This can be helpful if you need to install prerequisite maintenance on IRLM for one database product, as it will not affect the use of IRLM by the other.

2.2 Running of IMS subsystems

The procedures to run IMS address spaces are supplied by IBM. The procedures will be available in the PROCLIB data set. There are procedures for each type of region:

- DB/DC control region
- DCC control region
- DBCTL control region
- DLI separate address space
- DBRC address space
- IRLM address space
- Message processing region (MPR)
- IMS batch region (BMP)
- Fast Path region (IFP)
- Fast Path utility region
- DLI batch region
- IMSRDR region

These procedures should be modified with the correct data set names for each IMS system. Table 3 contains the procedure member names as found in the PROCLIB.

Table 3. IMS procedure member names

Region Name	Procedure Member Name
DB/DC control region	IMS
DC control region	DCC
DBCTL control region	DBC
DLI separate address space	DLISAS
DBRC	DBRC
IRLM	DXRJPROC
Message processing region (MPR)	DFSMPR
IMS batch processing region (BMP)	IMSBatch
Fast Path region (IFP)	IMSFP
Fast Path utility region	FPUTIL
DLI batch region	DLIBATCH
IMSRDR region	IMSRDR

For details of these and other procedures supplied in the PROCLIB, see Procedures in the *IMS/ESA Installation Volume 2: System Definition and Tailoring*, GC26-8737.

2.3 Running multiple IMS systems on one OS/390 system

Multiple IMS systems can be run on a single OS/390 image. One instance of an IMS system (control region and all dependent regions) is referred to as one IMS subsystem. In many cases these would be production and testing subsystems.

2.3.1 IMS subsystems

Each IMS subsystem should have unique VTAM ACB and IMSID names. The application dependent regions use the IMSID to connect to the corresponding IMS control region. If the dependent region starts and there is no control region running using that IMSID, it will produce a message to the MVS system console and then wait for a reply. Each IMS subsystem can have up to 99 dependent regions. However, there are other limiting factors.

If the IRLM is used, it must be started before the IMS control region is. If IMS starts to come up first, it will write a message to the MVS system console and wait for a reply. If the IRLM is specified, IMS will not run without it.

The number of subsystems you can run on a single image of OS/390 will depend on a lot of factors. In most installations you can run up to 4 subsystems, although some installations have gotten as many as 8 small ones running concurrently. The number will vary depending on the size of each IMS systems. The amount of CSA required by each IMS system is often one of the most limiting factors in the equation.

2.3.2 Address Spaces

All the address spaces can either run as a started task or as a JOB. In most cases the IMS control region and the system dependent regions will run as started tasks. The application dependent regions are run as either.

When a control region is started, it will issue an MVS start command as shown in the example below:

```
/START xxxxxxxx, PARM= (DLS, imsid)  
/START xxxxxxxx, PARM= (DRC, imsid)
```

The xxxxxxx fields are the procedure names. These commands will start the DLISAS and DBRC regions respectively.

2.3.3 Starting application dependent regions

IMS will not automatically start application dependent regions. There are several ways to have these regions started.

- The Time Control Option (TCO) of IMS can be used to issue /START REGION commands.
- Automation of some kinds can issue either IMS or MVS start commands.
- A JOB scheduling system can submit JOBS based on time or the notification of IMS being start via some automated messages.

2.3.3.1 Message processing regions

IMS MPR regions are normally started by an IMS start region command as shown below:

```
/START REGION xxxxxxxx
```

The xxxxxx fields are the member names in a library. These members contain the JOBS for the MPR regions. The IMSRDR procedure is used if the MPRs are JOBS. The IMSRDR procedure is customized to point at the correct library to find the JOB JCL. If you are running multiple IMS systems on an MVS system, they normally use a different version of the IMSRDR procedure each pointing at different libraries. The procedure name is specified on the IMSCTF macro in the system definition. See IMSCTF MACRO in *IMS/ESA Installation Volume 2: System Definition and Tailoring*, GC26-8737, for more information.

2.3.3.2 Fast Path application regions

Fast Path application regions are normally treated just like MPRs and follow the same rules and procedures.

2.3.3.3 Batch message processing regions

These regions are almost always started outside of IMS. Most BMPs are scheduled at appropriate times to meet application requirements. As long as the IMS control region is available, the BMPs can be run. BMP can execute even though there are no MPR running at the time.

2.4 Use of OS/390 services

IMS is designed to make the best use of the features of the OS/390 operating system. This includes:

1. Running in multiple address spaces — IMS subsystems (except for IMS/DB batch applications and utilities) normally consist of a control region address space, dependent address spaces providing system services, and dependent address spaces for application programs. Running in multiple address spaces gives the following advantages:
 - Maximizes use of CPUs when running on a multiple processor CPC. Address spaces can be dispatched in parallel on different CPUs.
 - Isolates the application programs from the IMS systems code. Reduces outages from application failures.
2. Runs multiple tasks in each address space — IMS, particularly in the control regions, creates multiple OS/390 subtasks for the various functions to be performed. This allows other IMS subtasks to be dispatched by OS/390 while one IMS subtask is waiting for system services.
3. IMS uses OS/390 cross memory services to communicate between the various address spaces making up an IMS subsystem. It also uses the OS/390 Common System Area (CSA) to store IMS control blocks that are frequently accessed by the address spaces making up the IMS subsystem. This minimizes the overhead in running in multiple address spaces.
4. IMS uses the OS/390 subsystem feature — IMS dynamically registers itself as an OS/390 subsystem. It uses this facility to detect when dependent address spaces fail, prevent cancellation of dependent address spaces (and to interact with other subsystems like DB2 and MQ?).
5. From V5 of IMS Database Manager and V6 of IMS Transaction Manager, IMS can make use of an OS/390 sysplex. Multiple IMS subsystems can run on the OS/390 systems making up the sysplex and access the same IMS databases (IMS V5) and the same message queue (IMS V6). This gives:
 - Increased availability — OS/390 systems and IMS subsystems can be switched in and out without interrupting the service.
 - Increased capacity — the multiple IMS subsystems can process far greater volumes.

For further details on sysplex data sharing and shared queues, refer to the ITSO publications *IMS/ESA Data Sharing in a Parallel Sysplex*, SG24-4303, and *IMS/ESA Version 6 Shared Queues*, SG24-5088.

2.4.1 MVS TCP/IP

IMS uses a function called OTMA to provide access to TCP/IP application. Any TCP/IP application can have access to IMS via the OTMA. IMS has introduced a function called the IMS TOC Connector, which provides an interface. For more details on OTMA and the IMS TOC Connector, refer to the redbook *IMS e-Business Using the IMS Connectors*, SG24-5427.

2.4.2 APPC/MVS

Advanced Program to Program Communications/IMS (APPC/IMS) support for Logical Unit type 6.2 supports the formats and protocols for program-to-program communication.

APPC/IMS enables applications to be distributed throughout your entire network and to communicate with each other regardless of the underlying hardware.

In the host environment, APPC/VTAM has been added to the VTAM base to facilitate the implementation of APPC/IMS support. In addition, APPC/MVS works with APPC/VTAM to implement APPC/IMS functions and communication services in an MVS environment. APPC/IMS takes advantage of this structure and uses APPC/MVS to communicate with LU 6.2 devices. Therefore, all VTAM LU 6.2 devices supported by APPC/MVS can access IMS using LU 6.2 protocols to initiate IMS application programs, or conversely be initiated by IMS.

APPC/IMS provides compatibility with non-LU 6.2 device types by providing a device-independent API. This allows an application program to work with all device types (LU 6.2 and non-LU 6.2) without any new or changed application programs.

IMS supports APPC conversations in two scenarios:

- Implicit** In this case, IMS supports only a subset of the APPC functions, but enables an APPC incoming message to trigger any standard IMS application that is already defined in the normal manner to IMS, and uses the standard IMS message queue facilities, to schedule the transaction into any appropriate dependent region.
- Explicit** In this case, the full set of CPIC command verbs can be used, and the IMS application is written specifically to cater only for APPC triggered transactions. The standard IMS message queues are not used in this case, and the IMS control region only helps create the APPC conversation directly between the APPC client and the IMS dependent region to service this request. The IMS control region takes no further part, regardless of how long the conversation may be active.

For further details, refer to 7.3, “Advanced Program-to-Program Communication (APPC)” on page 50.

2.4.3 Security server for OS/390 (RACF)

IMS was developed prior to the introduction of RACF. As a result, it initially incorporated its own security mechanisms to control user access to the various IMS resources, transactions, databases, etc. This security was controlled by a number of means. A number of security exits were provided. Also, a series of bitmaps defined users and their access to resources. This is referred to as a security matrix. These are load modules, produced by the IMS security maintenance utility following the generation of an IMS system.

With the introduction of RACF, IMS has been enhanced to make use of RACF for controlling access to IMS resources. It is now possible to use the original IMS security features, the new RACF features, and combinations of these. Using RACF provides more flexibility than the older security features, so we would recommend using these where possible.

The normal features of RACF can also be used to protect IMS data sets, both system and database.

For further information, refer to Chapter 24., “IMS security overview” on page 253, and for full details on using RACF to protect IMS resources, refer to the *IMS/ESA Administration Guide: System*, SC26-8013.

2.4.4 Transaction server for OS/390 (CICS)

The Transaction Server for OS/390, historically referred to as CICS, is able to connect to IMS in several different ways.

2.4.4.1 DBCTL

Prior to IMS Version 3, any CICS users who required access to IMS databases were required to use CICS Local DLI. In this case, the IMS system was not run on its own, but the CICS system was used to manage IMS hierarchical databases. The IMS software needed to be installed, and then the relevant components linked into the CICS system and ran in the CICS address space.

IMS Version 3 introduced IMS DBCTL, which is an IMS system that is connected to CICS transaction management systems. DBCTL is an IMS subsystem that allows access to DL/I Full Function databases and Data Entry Databases (DEDBs) from CICS environments. Access from transaction management subsystems (excluding the IMS/ESA Transaction Manager) is provided through the DBCTL coordinator controller (CCTL) interface. The IMS/ESA Database Manager may be connected through DBCTL to any CICS system from CICS/ESA Version 3, or higher.

With IMS/ESA Version 5, CICS applications can no longer access DL/I databases through local DL/I. Therefore, IMS DBCTL must be used.

IMS can be generated to run purely in DBCTL mode, if IMS TM has not been installed. If IMS TM has been installed, then a standard IMS DB/DC system can still be used to provide the DBCTL services required by CICS.

2.4.4.2 Intersystem Communication (ISC) links

ISC links between IMS and CICS use the standard LU 6.1 protocol available within the network. They can use standard VTAM connections, or direct connections.

ISC is a part of the IMS Transaction Manager. It is one of the ways to connect multiple subsystems. The other means of connection is multiple systems coupling (MSC).

As defined under SNA, ISC is an LU 6.1 session that:

- Connects different subsystems to communicate at the application level.
- Provides distributed transaction processing permitting a terminal user or application in one subsystem to communicate with a terminal or application in a different subsystem and, optionally, to receive a reply. In some cases, the application is user-written; in other cases, the subsystem itself acts as an application.
- Provides distributed services. Thus, an application in one subsystem can use a service (such as a message queue or database) in a different subsystem.

SNA makes communication possible between unlike subsystems and includes SNA-defined session control protocols, data flow control protocols, and routing parameters.

For further details, refer to 7.1, “Inter-System Communications (ISC)” on page 49.

2.5 Other hardware/operating system platforms

The non-OS/390 mainframe platforms where all or part of the IMS functions can be used are as follows:

- A subset of the IMS Database Manager component functions are available on the IBM VSE/ESA operating system running on the S/390 platform. As the Transaction Manager component is not available, this is normally used with CICS for transaction processing. The internal structures of the databases are, broadly, similar. However, it is not normally possible to access IMS databases on an OS/390 system from a VSE/ESA system, and it is not possible to access an IMS database on a VSE/ESA system from an OS/390 system. Limited access from CICS systems running on one operating system platform to IMS databases on the other operating system platform are available via CICS function shipping.
- The IBM P/390 product, which, by means of a standard expansion board, implements OS/390 functionality on a PC workstation or UNIX server, enables an IMS subsystem to be run on that workstation/server. However, this would not normally be able to support production volumes, though it may be suitable for a legacy application with very low throughput. It is however, suitable for development and unit testing by individual application programmers. It provides one possible solution to the problem of running multiple application copies for unit testing. As the P/390 card implements the S/390 instruction set as specified in the S/390 Principles of Operation, it is also suitable for systems programming development, testing and problem investigation. However, it cannot be used with functions implemented in microcode and hardware (for example, logical partitioning (LPARs), running sysplex coupling facility code).
- Third party products available from Micro Focus and Computer Associates implement part or all of the IMS API on a PC workstation. This provides similar facilities to the P/390, although as it only implements the application interfaces, it is really only suitable for application development.

Chapter 3. IMS TM and DB general information

This chapter contains general information about the IMS start processes.
The topics covered are:

1. 3.1, “IMS startup” on page 27
2. 3.2, “IMS shutdown” on page 28
3. 3.3, “Logging” on page 28
4. 3.4, “Security” on page 28
5. 3.5, “IMS generation” on page 28
6. 3.6, “IMS recovery” on page 28

3.1 IMS startup

This section describes the common types of IMS startup commands, applicable to both transaction management and/or database management.

- Cold start

An IMS CTL region cold start is done at the first time you start the system. During cold start, we format (initialize) the message queues, dynamic log and restart data sets.

- Emergency restart

In case of failure, IMS is restarted with the log active at the time of failure. Restart processing will back-out the database changes of incomplete MPPs and BMPs. The output messages inserted by these incomplete MPPs will be deleted.

After back-out, the input messages are re-enqueued, the MPPs are restarted, and the pending output messages are (re)-transmitted. If a BMP was active at the time of failure, it must be resubmitted via MVS job management. If the BMP uses the XRST/CHKP calls, it must be restarted from its last successful checkpoint. In this way, missing or inconsistent output is avoided.

- Normal restart

Normal restart or warm start is done from a previous normal IMS termination. The message queues are preserved in this way.

- Automatic restart

In most cases, this should be the default. With an automatic restart, IMS will startup, using either normal restart, or emergency restart, depending on the previous shutdown status.

If the last IMS shutdown was successful, then a normal restart will be done. If the last IMS shutdown was an abend, then an emergency restart will be automatically done.

- Other restarts

There are numerous other types of manual starts possible with IMS, each with unique requirements. Refer to the *IMS Operator's Reference Manual*, SC26-8742.

3.2 IMS shutdown

There are also several different ways of shutting down IMS, depending on what type of control region you are running (DB/BC, DBCTL, DCCTL), and whether or not the IMS message queues are required following the next startup. Refer to the *IMS Operator's Reference Manual*, SC26-8742, for further details.

3.3 Logging

During IMS execution, all information necessary to restart the system in the event of hardware or software failure is recorded on a system log data set.

For further details, refer to Chapter 22, "IMS logging" on page 239.

3.4 Security

IMS itself has security built in, and also has the ability of more extensive security by using RACF, or with user exits.

The security can cover such things as:

- Signon security
- Transaction security
- Program security

For further details, refer to Chapter 24, "IMS security overview" on page 253.

3.5 IMS generation

All optional features of IMS, including what type of control region is required (DB/DC, DBCTL, DCCTL), must be specified in the IMS generation.

Almost all programs, databases, transactions, and terminals (unless the ETO feature is used) within IMS must also be predefined within the IMS generation.

For further details, refer to Chapter 23, "IMS system generation process" on page 245.

3.6 IMS recovery

There are also a number of tools and features available with IMS to help in recovery scenarios:

- Extended Recovery Facility (XRF), for use in having a hot standby system ready to take over within the same site.
- Remote Site Recovery (RSR), for use in complete site disasters, to recover the complete IMS system(s) very quickly at another site.

Both of these are mentioned here for completeness. For more details, refer to the relevant IMS manuals.

Part 2. IMS Transaction Manager

This part is intended to provide the data communication and system analyst with a detailed description of the IMS data communication and online functions. The following main topics are covered in these chapters:

- A description of the different roles the control region has to play, what an IMS message looks like, and a summary of the life of a transaction. Refer to Chapter 4, “The IMS control region” on page 31.
- A description of how inputs to IMS are processed when they originate from a terminal. Refer to Chapter 5, “Processing input from a terminal” on page 35.
- A description of IMS Fast Path transactions. Refer to Chapter 6, “Fast-Path transactions” on page 47.
- A description of non-terminal related IMS inputs. Refer to Chapter 7, “Non-terminal related input” on page 49.
- A description of the IMS master terminal. Refer to Chapter 8, “The master terminal” on page 53.
- An overview of application processing within IMS. Refer to Chapter 9, “Application program processing overview” on page 55.

Chapter 4. The IMS control region

The control region (CTL) is an MVS address space that can be initiated through an MVS start command, or by submitting JCL. The terminals, message queues, and logs are all attached to this region. The control region will start two dependent regions: the DLI separate address space (DLISAS), and the DBRC address space. The databases will be attached to the DLISAS region. The application databases will be attached to the DLISAS region. A type 2 supervisor call routine is used for switching control information, message and database data to the other regions, and back.

The CTL region normally runs as a system task and uses MVS access methods for terminal and database access.

4.1 The IMS message

The goal of IMS is to perform online transaction processing. This consists of:

1. Receiving a request for work to be done. The request is entered at a remote terminal. It is usually made up of a transaction code which identifies to IMS the kind of work to be done, and some data which is to be used in doing the work.
2. Initiating and controlling a specific program which will use the data in the request to do the work the remote operator asked to be done, and to prepare some data for the remote operator in response to the request for work (for example, acknowledgment of work done, answer to a query, etc.).
3. Transmission of the data prepared by the program back to the terminal originally requesting the work.

The above sequence is the simplest form of a transaction. It involves two messages: an input message from the remote operator requesting that work be done, and an output message to the remote operator containing results or acknowledgment of the work done.

A message, in the most general sense, is a sequence of transmitted symbols. In the context of IMS, this is called a transmission. A transmission may have one or more messages, and a message may have one or more segments. A segment is defined by an end-of-segment (EOS) symbol, a message is defined by an end-of-message (EOM) symbol, and a transmission is defined by an end-of-data (EOD) symbol. The valid combinations of the conditions represented by EOS, EOM, and EOD can be found in Table 4.

Table 4. Valid combinations of the EOS / EOM / EOD symbols

Condition	Represents
EOS	End of segment
EOM	End of segment / end of message
EOD	End of segment / end of message / end of data

The relationship between transmission, message, and segment is shown in Figure 5.

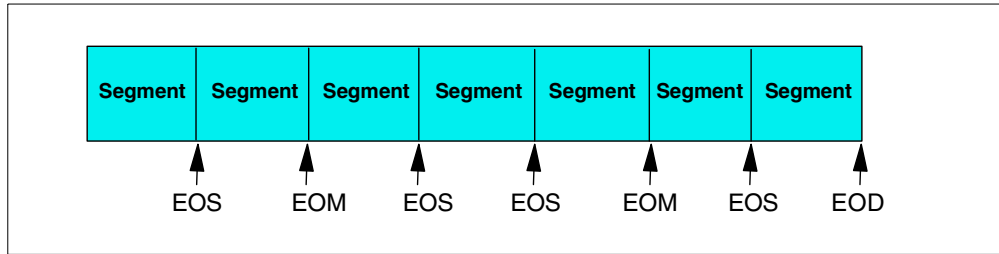


Figure 5. Transmission, message, and segment relationship

The character values or conditions that represent the end of segment and/or message are dependent on the terminal type.

In our subset, 3270 terminals only, the physical terminal input will always be a single segment message and transmission. The EOS, EOM and EOD condition will all be set after the enter or program function key is pressed and the data is transmitted.

On the output side, a message can be divided into multiple segments. Also an application program can send different messages to different terminals, that is, a message to a printer terminal and a message to the input display terminal. Each segment requires a separate insert call by the application program.

The format of a message segment as presented to or received from an application program is shown in Figure 6, where:

- LL** Total length of the segment in bytes, including the LL and ZZ fields.
- ZZ** IMS system field
- DATA** Application data, including the transaction code.

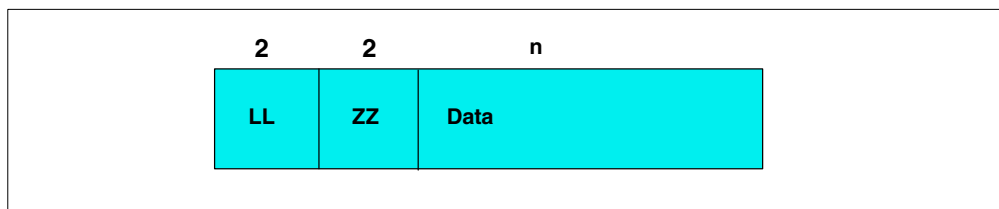


Figure 6. A message segment

4.2 An IMS transaction flow

Once the CTL region is started, it will start the system dependent regions (DLISAS and DBRC). The MPRs and BMP regions can be started various ways.

- By IMS commands
- By JOB submission
- By automated operations commands

The general flow of a message from a message processing program (MPP) is shown in Figure 7. This diagram is intended to give a general flow of the message through the system and not a complete detailed description.

5. The communication modules retrieve the message from the message queues, and send it to the output terminal. MFS is used to edit the screen and printer output.
6. All changes to the message queues and the databases are recorded on the logs. In addition, checkpoints for system (emergency) restart and statistical information are logged.

Notes:

- The physical logging modules run as a separate task and use MVS ESTAE for maximum integrity.
 - The checkpoint identification and log information are recorded in the restart and RECON data sets.
7. Program Isolation (PI) locking assures database integrity when two or more MPPs update the same database. It also backs out (undoes) database changes made by failing programs. This is done by maintaining a short-term, dynamic log of the old database element images. IRLM is also optionally available to replace the PI locking, but required if the IMS is taking part within a data sharing environment.

The control module provides multi-tasking for the above activities. These separate functions, that is, input processing, queueing, MPP processing, database call processing, output processing, etc., can be executed asynchronously for multiple transactions. However, they will be executed in sequence for a unique transaction occurrence. The MVS EVENT facility is used for the control of the multi-tasking and input/output operations.

Chapter 5. Processing input from a terminal

IMS can accept input messages from 3270 type terminals. This input can consist of three types of messages. Refer to Figure 8 for the following discussion.

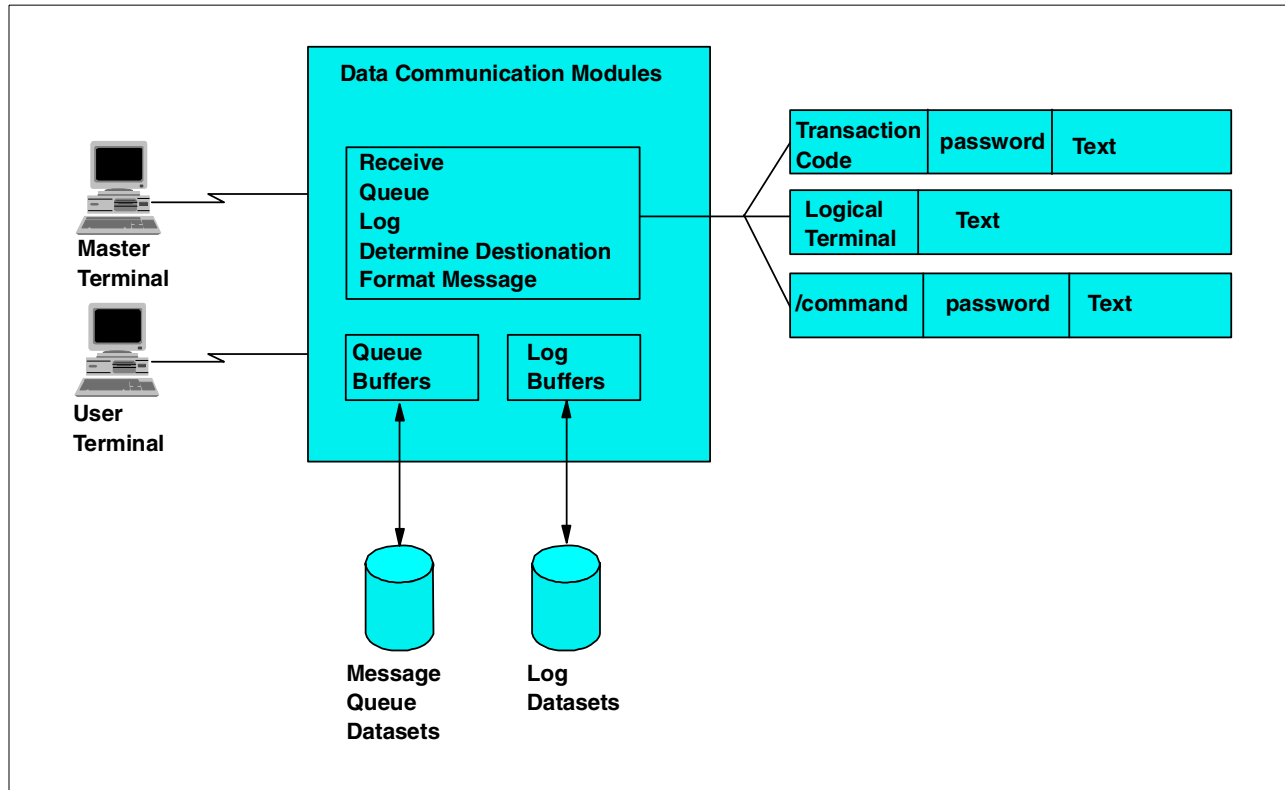


Figure 8. Input message processing

When IMS reads data from a terminal via the telecommunication access method, it first checks the type of input data.

5.1 Input message types

The three basic types of terminal input are:

- An input transaction message

This message is routed to an application program for processing with the first 1-to-8 bytes of the message identifying the transaction code.

- A message switch

This message is routed to another terminal, with the first 1-to-8 bytes used as the name of the destination logical terminal (LTERM). The LTERM may be a USERID if the Extended Terminal Option (ETO) is used.

A program may issue an input message to another application program with the first 1-to-8 bytes of the message identifying the transaction code.

- A command

A command starts with a slash (/), and is processed by IMS itself.

5.2 Terminal types

There are two basic types of terminals that can connect to IMS. They are either:

- Static** The terminal is specifically defined to IMS in the IMS gen, and this determines what physical terminal name (NODE NAME), and logical terminal name (LTERM) is available for use.
- Dynamic** If a terminal is not statically defined to IMS in the IMS gen, IMS can create a dynamic terminal definition. This requires either the IMS Extended Terminal Option (ETO), a separately ordered feature of IMS or other third-party vendor products. Dynamic terminals have not been previously defined to IMS — their definitions are generated by ETO when the user logs on/ signs on.

If a terminal user attempts to connect to IMS using a terminal that is defined to IMS as static, then the user will use the defined NODE NAME / LTERM name combination.

If a terminal user attempts to connect to IMS using a terminal that is not defined to IMS as static, and dynamic terminal support is available, then the dynamic terminal product (such as ETO) will be used to determine what the LTERM name is; and whether it is based on the users USERID, the NODE NAME, or some other value.

If a terminal user attempts to connect to IMS using a terminal that is not defined to IMS as static, and dynamic terminal support is not enabled, then the user will be unable to logon to IMS.

5.3 Input message origin

IMS maintains the name of the terminal or user from which an input message is received. When a message is passed to an application program, this is also made available to that program, via its program communication block (PCB).

This origin is the logical terminal name (LTERM). The LTERM name may be specific to the user, or may be specific to the physical location, depending on how the IMS system is defined. Refer to 5.2, “Terminal types” on page 36.

5.4 Terminal input destination

The destination of the terminal input is dependent upon the type of input.

An input command goes directly to the IMS command processor modules, while a message switch or a transaction are stored on the message queue. When a 3270-based message is received by IMS, the message input is first processed by message format service (MFS), except when input is from previously cleared or unformatted screen. MFS provides an extensive format service for both input and output messages. It is discussed in detail in Chapter 18, “IMS message format service” on page 157.

When the input message is enqueued to its destination in the message queue, the input processing is completed. If more than one LTERM is defined or assigned to a physical terminal, they are maintained in a historical chain: the oldest defined/assigned first. Any input from the physical terminal is considered to have originated at the first logical terminal of the chain. If, for some reason (such as security or a stopped LTERM), the first logical terminal is not allowed to enter the message, all logical terminals on the input chain are interrogated in a chain sequence for their ability to enter the message. The first appropriate LTERM found is used as message origin. If no LTERM can be used, the message is rejected with an error message.

5.5 Message queuing

All Full Function input and output messages in IMS are queued in message queues. Refer to Figure 9. For Fast Path transactions — refer to 5.5.4, “Fast Path transactions” on page 39.

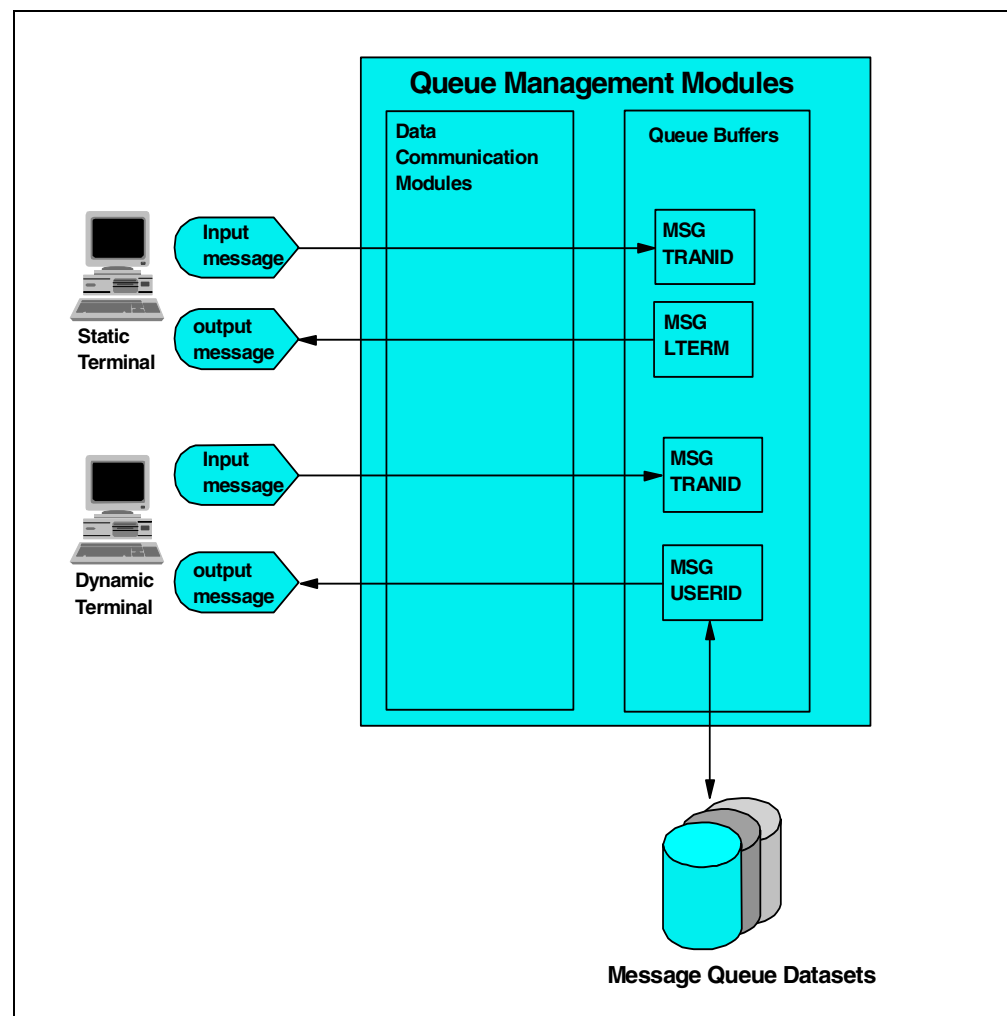


Figure 9. Message queuing

The use of message queues allows input processing, output processing, command processing, and application program processing to be performed asynchronously, to a large extent. This means, for example, that the input processing of message A can be done in parallel with the database processing for message B and the output processing for message C. Messages A, B, and C can be different occurrences of the same or different message types and/or transaction codes.

Messages in the IMS message queues are stored by destination, priority, and the time of arrival in IMS. A destination can be:

- A message processing program (MPP), which is for transaction input. Ordering is by transaction code.
- A logical terminal (LTERM), which is for a message switch, command responses, and output generated by application programs.
- A USERID, which is for a message created by this USERID on any physical terminal.

The message queue buffers are maintained in main storage (defined by the MSQUEUE macro). Should the memory-based message queue buffers become full, messages are then stored on the message queue data sets on DASD. The queue blocks in main storage and on direct access storage are reusable. As far as possible messages are stored in the message queue buffers, to minimize the number of I/O operations required during processing.

5.5.1 Queue size and performance considerations

Messages in the IMS message queue are primarily held in buffers in main storage. However, when messages are added to the queues faster than IMS can process these messages, the message queue buffers can fill. In this situation, any new messages are written to the message queue data sets. The performance of these data sets then becomes very important. The data sets should be on a DASD volume with fast response times, and the data sets should be appropriately sized to ensure that there is always space available.

5.5.2 Multiple message queues

Since IMS Version 4, the IMS Queue Manager now supports concurrent I/O operations to its message queue data sets, allowing the IMS message queue to be distributed across multiple physical queue data sets. This enhancement supports the long and short message queue data sets.

This enhancement is activated when more than one DD statement per message queue data set is provided. You can supply up to ten DD statements for each queue data set. These DD statements can be allocated on different device types, but LRECL and BLKSIZE must be the same for all the data sets of a single queue.

It is strongly recommended that multiple queue data sets be used, so that in an emergency situation, the IMS systems performance will not degrade while trying to handle a large volume of messages going to and from the message queue data sets.

Refer to the *IMS/ESA V6 Install Volumes 1 & 2* (GC26-8736 and GC26-8737 respectively) for further detailed guidelines for selecting message queue parameters such as block sizes, QPOOL size, queue data set allocation etc.

5.5.3 Shared Queues

IMS Version 6 provides the facility for multiple IMS systems in a sysplex to share a single set of message queues. This function is known as IMS Shared Queues, and the messages are held in structures in a coupling facility. All the IMS subsystems in the sysplex share a common set of queues for all non-command messages (that is, input, output, message switch, and Fast Path messages). A message that is placed on a Shared Queue can be processed by any of several IMS subsystems in the share queues group as long as the IMS has the resources to process the message.

The Shared Queues function of IMS version 6 is optional, and you may continue to process with the message queue buffers and message queue data sets provided in earlier versions of IMS.

The benefits in using Shared Queues enables automatic workload balancing across all IMS subsystems in a Sysplex. New IMS subsystems can be dynamically be added to the Sysplex, and share the queues as workload increases, allowing in incremental growth in capacity. The use of Shared Queues can also provide increased reliability and failure isolation: if one IMS subsystem in the Sysplex fails, any of the remaining IMS subsystems can process the work that is waiting in the Shared Queues.

5.5.4 Fast Path transactions

Fast Path transactions do not use the standard IMS message queues. Fast Path transactions are scheduled by a separate function within the IMS transaction manager, called the Expedited Message Handler (EMH). For further scheduling information, refer to Chapter 6, “Fast-Path transactions” on page 47.

5.5.5 APPC triggered transactions

There are two types of APPC transactions, implicit and explicit. With implicit APPC transactions, IMS receives a transaction request via APPC. This transaction is placed onto the IMS message queues in the same way as a 3270-generated transaction. The message is passed to an MPP for processing, and the response is routed back to the originating APPC partner. The MPP program uses the DL/I interface to receive the message from the message queue, and put the response back onto the message queue.

With explicit APPC transactions, IMS schedules a program into an MPR (message processing region). This program uses APPC verbs to communicate with the APPC partner program to process the transaction.

For further details, refer to 7.3, “Advanced Program-to-Program Communication (APPC)” on page 50.

5.5.6 OTMA triggered transactions

OTMA allows IMS to receive a message through a different communications protocol (for example, TCP/IP sockets, MQ, remote procedure calls, etc.). The message is received by IMS, and it is placed into the IMS message queue for processing in the usual manner. The response message is passed back to the originator through OTMA.

5.5.7 Message scheduling

Scheduling is the loading of the appropriate program into a message processing region. IMS can then pass messages stored on the IMS message queue to this program when it issues the GU IOPCB call.

Once an input message is available in the message queue, it is eligible for scheduling. Scheduling is the routing of a message in the input queue to its corresponding application program in the message processing region. See Figure 10.

The linkage between an input message (defined by its transaction code) and an application program (defined by its name) is established at system definition time. Multiple transaction codes can be linked to a single application program, but only one application program can be linked to a given transaction code.

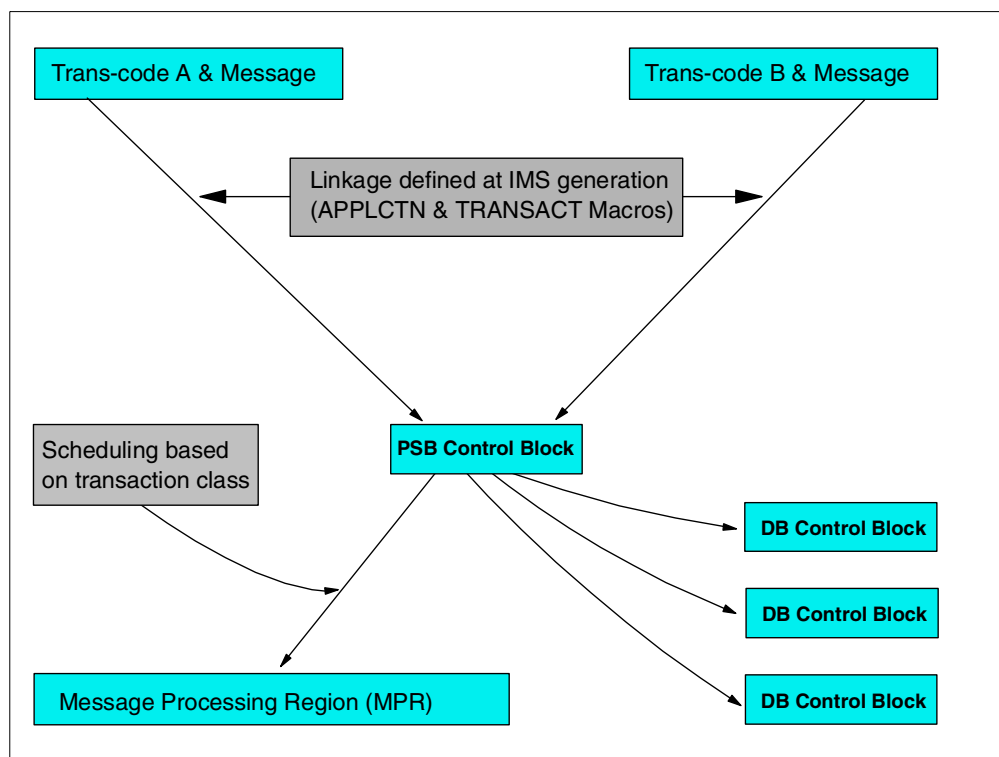


Figure 10. Message scheduling

The class in which a transaction code with run is defined in two ways:

- On the APPLCTN macro
- On the MSGTYPE parameter of the TRANSACT macro

If the class is specified on the APPLCTN macro, it need not be defined on the TRANSACT macro. If it is specified on both, then the class on the TRANSACT macro will override the APPLCTN macro specification. Figure 11 illustrates the definition of a transaction.

APPLCTN PSB=DFSIVP1,PGMTYPE=TP	
TRANSACT CODE=IVINO,MODE=SNGL,	X
MSGTYPE= (SNGLSEG,NONRESPONSE,1)	
APPLCTN PSB=DFSIVP2,PGMTYPE= (TP,1)	
TRANSACT CODE=IVTNO2,MODE=SNGL,	X
MSGTYPE= (SNGLSEG,NONRESPONSE)	

Figure 11. Transaction definition in IMS stage 1 input

Notice the following about these transaction definitions:

- Transaction DFSIVP1 has the class defined as the third parameter on the MSGTYPE parameter on the TRANSACT macro.
- Transaction DFSIVP2 has the class defined on the APPLCTN macro.
- Both transactions are assigned to class 1.

5.5.8 Transaction scheduling and priority

The transaction scheduling algorithm can be a very sophisticated algorithm, as it needs to make use of all the IMS and system resources in the most efficient manner possible. However, most users do not need to use the power of the scheduling algorithms, as the resources available in IMS today (such as the number of message processing regions) are much greater than when these algorithms were designed several decades ago.

There are a few parameters on the transaction definition which will affect the scheduling options:

- PROCLIM
- PARLIM
- MAXRGN
- PRTY

5.5.8.1 Parallel scheduling

A transaction will only process in one MPR at a time unless parallel processing is specified. To allow more than one MPR to schedule a transaction type at a time, code the SCHDTYP parameter on the APPLCTN macro:

```
APPLCTN PSB=DFSIVP1,PGMTYPE= (TP,1) ,SCHDTYP=PARALLEL
```

Unless there are application restrictions on processing the message in strict first-in, first-out sequence, parallel scheduling should be applied to all transactions. This will allow IMS to make the best use of IMS resources while providing the best possible response time to individual transactions.

The PARMLIM parameter on the TRANSACT macro will determine when a transaction will be scheduled in another region. When the number of messages on the queue for this transaction exceeds the value on the PARLIM, another region will be used.

The MAXRGN parameter is used to restrict the number of MPRs which can process a transaction at any one time. This is done to avoid the situation of all the MPRs being tied up by a single transaction type.

5.5.8.2 Priority

The PRTY parameter on the TRANSACT macro sets the priority of a transaction. This is how to differentiate one transaction from another if they run in the same transaction class. A transaction of a higher priority will be scheduled before a lower priority one. However an MPR will process a transaction in a higher class (for this MPR, see 5.5.10, “Scheduling in a dependent region” on page 42 for more details) before a transaction in a lower class regardless of the priority. A transaction priority will increase once the number of transactions on the message queue exceed the value set on the third value of the PRTY keyword. It will increase to the value set on the second parameter of the PRTY keyword. This has the effect of trying to avoid a long queue on any single transaction code by giving it a higher priority.

Another factor in transaction scheduling is the PROCLIM value. This is discussed in 5.5.10.2, “PROCLIM processing” on page 44.

5.5.9 Scheduling conditions

The following conditions must be met for a successful scheduling:

1. An MPR region must be available. Actually, the termination of a prior transaction running in an MPR region triggers the scheduling process.
2. There must be a transaction input message in the queue.
3. The transaction and its program are not in a stopped state.
4. Enough buffer pool storage is available to load the program specification block (PSB) and the referenced database control blocks if not already in main storage.
5. The database processing intent does not conflict with an already active application program (a BMP for instance). Processing intent is discussed in more detail in 12.5, “Data Base Processing Intent” on page 91.

If the first transaction code with a ready input message does not meet all the above conditions, the next available input transaction is interrogated, and so forth. If no message can be scheduled, the scheduling process is stopped until another input message is enqueued. If the scheduling is successful, the IMS routines in the dependent region load the corresponding MPP and pass control to it.

5.5.10 Scheduling in a dependent region

The IMS scheduler will assign the application transaction processing to an dependent MPR. The number of MPRs available to an IMS system is 999 dependent regions.

The transactions are assigned to classes. The maximum number of transactions classes is set at system generation time by the MAXCLAS parameter of the IMSCTRL macro.

5.5.10.1 Class processing

Each dependent MPR can run up to four transaction classes. The order in which they are specified is a priority sequence. That means that the transaction class named first is the highest and the one named last is the lowest. Each MPR can have a different sequence of the same or different transaction combinations. The classes are named on the PROC statement of the JCL running the MPR. Figure 12 shows an example of the MPR JCL. The MPR can be run as a JOB or a started task.

```
//IVP6TM11 EXEC PROC=DFSMPR,TIME=(1440),  
//          AGN=BMP01,                AGN NAME  
//          NBA=6,  
//          OBA=5,  
//          SOUT='*',                SYSOUT CLASS  
//          CL1=001,                TRANSACTION CLASS 1  
//          CL2=006,                TRANSACTION CLASS 2  
//          CL3=013,                TRANSACTION CLASS 3  
//          CL4=000,                TRANSACTION CLASS 4  
//          TLIM=10,                MPR TERMINATION LIMIT  
//          SOD=,                SPIN-OFF DUMP CLASS  
//          IMSID=IMSY,            IMSID OF IMS CONTROL REGION  
//          PREINIT=DC,            PROCLIB DFSINTXX MEMBER  
//          PWF=N                PSEUDO=WFI  
//*
```

Figure 12. MPR PROC statement example

The classes which the MPR runs can be changed while the MPR is running. This is done through the /ASSIGN command. When the ASSIGN command is executed, only those classes specified on the command will be available to that MPR. The changes will be maintained until the MPR is restarted, at which time the values on the PROC statement will be used again. Figure 13 illustrates an example of an ASSIGN command. Again the order of classes on the command is the priority sequence of those classes.

```
/ASSIGN CLASS 1 4 5 8 TO REGION 3
```

Figure 13. /ASSIGN CLASS command example

To list the classes assigned to an MPR the /DISPLAY ALL command can be used. Figure 14 shows the /DISPLAY ACTIVE command and the output.

```

/DIS ACTIVE
  REGID JOBNAME  TYPE TRAN/STEP PROGRAM STATUS      CLASS      IMSY
    1 SJIMSYM1  TP              WAITING      1, 4, 6, 9  IMSY
    2 SJIMSYM2  TP              WAITING      2, 3, 5, 1  IMSY
      BATCHREG  BMP      NONE    IMSY
      FPRGN     FP       NONE    IMSY
      DBTRGN    DBT      NONE    IMSY
      SJIMSYDB  DERC                     IMSY
      SJIMSYDL  DLS        IMSY
VTAM ACB OPEN  -LOGONS DISABLED  IMSY
IMSLU=N/A.N/A  APPC STATUS=DISABLED  IMSY
OTMA GROUP=IMSCGRP  STATUS=ACTIVE  IMSY
APPLID=SCSIM6YA  GRSNAME=        STATUS=DISABLED  IMSY
LINE ACTIVE-IN -    1 ACTIV-OUT -    0  IMSY
NODE ACTIVE-IN -    0 ACTIV-OUT -    0  IMSY
*99298/155826*  IMSY

```

Figure 14. Display active

Note the following from the information from Figure 14:

- There are two MPRs.
- The MPR named SJIMSYM1 run classes 1, 4, 6, and 9.
- The MPR named SJIMSYM2 runs classes 2, 3, 5, 1.
- Class 1 has the highest priority in MPR SJIMSYM1 and the lowest in MPR SJIMSYM2.

When an MPR is looking to find the a transaction to schedule, it will use the following criteria:

1. The highest priority transaction ready in the highest priority class
2. Any other transaction in the highest priority class
3. The highest priority transaction ready in the second highest priority class
4. Any other transaction in the second priority class

This sequence of priorities will be used for all the available classes for this MPR.

NOTE: If a transaction has a class for which there are no MPRs currently allowed to run that class, the transaction will not be scheduled and will remain on the input queue.

5.5.10.2 PROCLIM processing

IMS also tries to increase throughput of the MPR by processing more than one message for the same transaction. This is to make use of the fact that the program has already been loading into the MPR's storage, and the PSB and DBD control blocks also have been loaded. This will increase the throughput of the number of messages processed by this MPR, as it will avoid some of the overhead with reloading the program and control blocks.

At the completion of the transaction, IMS will check the PROCLIM value on the TRANSACT macro for this transaction. The MPR will process the number of messages allowed in the first value of this keyword before looking to see what other transactions are available to be scheduled. This means the MPR can process more transactions without having to go through the scheduling logic for each transaction.

5.6 Database processing intent

A factor that significantly influences the scheduling process is the intent of an application program toward the databases it uses. Intent is determined by examining the intent last associated with the PSB to be scheduled. At initial selection, this process involves bringing the intent list into the control region. The location of the intent list is maintained in the PSB directory. If the analysis of the intent list indicates a conflict in database usage with a currently active program in MPP or BMP region, the scheduling process will select another transaction and try again.

The database intent of a program as scheduling time is determined via the PROCOPT= parameters in the PSB.

An conflicting situation during scheduling will only occur if a segment type is declared exclusive use (PROCOPT=E) by the program being scheduled and a already active program references the segment in its PSB (any PROCOPT), or vice versa.

5.6.1 Scheduling a BMP

A BMP is initiated in a standard MVS address space via any regular job submission facility. This could be from either:

- TSO and SUBMITing the job
- Some job scheduling system

However, during its initialization the IMS scheduler in the control region is invoked to assure the availability of the database resources for the BMP.

5.6.2 Shared Queues

Scheduling of transactions in a Shared Queues environment is similar to those in a non-Shared Queues environment, however, all the checks are across all the IMS systems in the Shared Queues environment, and obviously, there are extra considerations as well. For further information on this, refer to the appropriate IMS manuals, as well as *IMS/ESA Version 6 Shared Queues*, SG24-5088, and *IMS/ESA Shared Queues: Planning Guide*, SG24-5257.

Chapter 6. Fast-Path transactions

Apart from standard IMS transactions, there are two types of Fast Path online transactions. They are:

- Fast Path exclusive
- Fast Path potential

6.1 IMS Fast Path exclusive transactions

Fast Path schedules input messages by associating them with a load balancing group. A load balancing group (LBG) is a group of Fast Path input messages that are ready for balanced processing by one or more copies of a Fast Path program. One LBG exists for each unique Fast Path message-driven application program.

The messages for each LBG are processed by the same Fast Path program. The EMH controls Fast Path messages by:

- Managing the complete execution of a message on a first-in-first-out basis
- Retaining the messages that are received in the control program's storage without using auxiliary storage or I/O operations
- Supporting multiple copies of programs for parallel scheduling
- Requiring that programs operate in a wait-for-input mode

6.2 IMS Fast Path potential transactions

Fast Path potential transactions are a mixture of standard IMS Full Function and Fast Path exclusive transactions.

The same transaction code can be used to trigger either a Full Function, or a Fast Path transaction, with an exit used to determine whether this instance of the transaction will be Full Function, or Fast Path.

Chapter 7. Non-terminal related input

This chapter covers the non-terminal related input, not already covered. This includes:

- Inter-System Communications (ISC)
- Multiple Systems Coupling (MSC)
- Advanced Program to Program Communication (APPC)
- Open Transaction Manager Access(OTMA)

7.1 Inter-System Communications (ISC)

ISC links between IMS and CICS use the standard LU 6.1 protocol available within the network.

They can use standard VTAM connections, or direct connections.

ISC is a part of the IMS Transaction Manager. It is one of the ways to connect multiple subsystems. The other means of connection is Multiple Systems Coupling (MSC).

As defined under SNA, ISC is an LU 6.1 session that:

- Connects different subsystems to communicate at the application level.
- Provides distributed transaction processing permitting a terminal user or application in one subsystem to communicate with a terminal or application in a different subsystem and, optionally, to receive a reply. In some cases, the application is user written; in other cases, the subsystem itself acts as an application.
- Provides distributed services. Thus, an application in one subsystem can use a service (such as a message queue or database) in a different subsystem.

SNA makes communication possible between unlike subsystems and includes SNA-defined session control protocols, data flow control protocols, and routing parameters.

7.2 Multiple Systems Coupling (MSC)

Multiple Systems Coupling (MSC) permits you to link multiple IMS systems and to distribute processing loads and system databases among them to satisfy your particular geographic and business requirements. Transactions entered in one IMS system can be passed to another IMS system for processing and the results returned to the initiating terminal. Terminal operators are unaware of these activities; their view of the processing is the same as that presented by interaction with a single system.

MSC links between 2 IMS systems use an IMS internal protocol for communications, making MSC only available between 2 IMS systems, known as the “front-end” system, which is where the transaction is entered by the terminal user, and the “back-end” system, where the transaction is processed.

The transaction is entered in the “front-end” system, and based on the definitions in the IMS stage 1 generation, the transaction is “shipped” to the “back-end” system. Once there, all standard IMS scheduling techniques apply. After processing, the results are shipped back to the “front-end” system, who then returns the results to the terminal user, who was unaware that any of this occurred.

7.3 Advanced Program-to-Program Communication (APPC)

Advanced Program-to-Program Communications/IMS (APPC/IMS) support for Logical Unit type 6.2 supports the formats and protocols for program-to-program communication.

APPC/IMS enables applications to be distributed throughout your entire network and to communicate with each other regardless of the underlying hardware.

In the host environment, APPC/VTAM has been added to the VTAM base to facilitate the implementation of APPC/IMS support. In addition, APPC/MVS works with APPC/VTAM to implement APPC/IMS functions and communication services in an MVS environment. APPC/IMS takes advantage of this structure and uses APPC/MVS to communicate with LU 6.2 devices. Therefore, all VTAM LU 6.2 devices supported by APPC/MVS can access IMS using LU 6.2 protocols to initiate IMS application programs, or conversely be initiated by IMS.

APPC/IMS provides compatibility with non-LU 6.2 device types by providing a device-independent API. This allows an application program to work with all device types (LU 6.2 and non-LU 6.2) without any new or changed application programs.

IMS supports APPC conversations in two scenarios:

- Implicit** In this case, IMS supports only a subset of the APPC functions, but enables an APPC incoming message to trigger any standard IMS application that is already defined in the normal manner to IMS, and uses the standard IMS message queue facilities, to schedule the transaction into any appropriate dependent region.
- Explicit** In this case, the full set of CPIC command verbs can be used, and the IMS application is written specifically to cater only for APPC triggered transactions. The standard IMS message queues are not used in this case, and the IMS Control Region only helps create the APPC conversation directly between the APPC client and the IMS dependent region to service this request. The IMS Control Region takes no further part, regardless of how long the conversation may be active for.

7.4 Open Transaction Manager Access (OTMA)

Open Transaction Manager Access (OTMA) is a function of IMS that was introduced with IMS/ESA Version 5. OTMA is a transaction based connectionless client-server protocol that provides an access path and an interface specification for sending and receiving transactions and data from IMS.

IMS supports various network protocols, such as Systems Network Architecture (SNA). These protocols are network-based, and so require specific installation and tuning as your network grows or changes. OTMA is not network-based; it is transaction-based, and so need not be changed as your network grows or changes. OTMA uses MVS sysplex services, in particular the MVS cross-system coupling facility (XCF). OTMA does not use VTAM, or any other IBM access method to manage its message processing.

The OTMA client handles all device-dependencies for a particular network protocol so that IMS TM can operate without needing to understand any of the device characteristics. And the OTMA client need not know anything of how the IMS system is defined. Because the OTMA client is the interface between IMS TM and the network, you do not need to change your application programs to use the network.

In a client/server environment, the IMS Transaction Manager is the high-performance server; the OTMA feature lets you attach many different MVS client subsystems. An OTMA client acts as an interface between IMS TM and:

- Other IMS TM systems
- MQSeries for MVS/ESA
- Transmission Control Protocol / Internet Protocol (TCP/IP)
- Distributed Computing Environment / Remote Procedure Call (DCE/RPC)

By itself, OTMA is not sufficient for a connection to one of the above. You need an OTMA client, which is an MVS application that acts an interpreter between the network or messaging protocol it supports and the IMS TM messaging protocol. Thus, for example, you can use one OTMA client for a network file system (NFS) connection to IMS TM, and another client for a TCP/IP connection to IMS TM (ITOC).

Chapter 8. The master terminal

When the IMS system is generated, the IMS master terminal MUST be included, and consists of two components:

- Primary master
- Secondary master

All messages are routed to both the primary and secondary master terminals. Special MFS support is used for the master terminal.

8.1 The primary master

Traditionally, the primary master was a 3270 display terminal of 1920 characters (24 lines by 80 columns). A sample traditional IMS master terminal is shown in Figure 15.

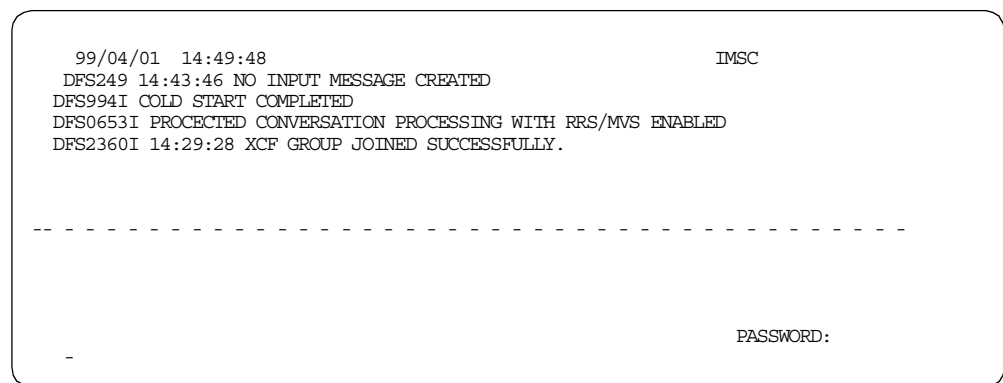


Figure 15. Master terminal screen

The display screen of the master terminal is divided into four areas.

- The message area is for IMS command output (except /DISPLAY and /RDISPLAY), message switch output that uses a message output descriptor name beginning with DFSMO (see MFS), and IMS system messages.
- The display area is for /DISPLAY and /RDISPLAY command output.
- The warning message area is for the following warning messages: MASTER LINES WAITING, MASTER WAITING, DISPLAY LINES WAITING, and USER MESSAGE WAITING. To display these messages or lines, press PA 1. An IMS password may also be entered in this area after the “PASSWORD” literal.
- The user input area is for operator input.

Program function key 11 or PA2 request the next output message and program function key 12 request the Copy function if it is a remote terminal.

8.2 The secondary master

Traditionally, the secondary master was a 3270 printer terminal.

This usage has also been phased out in many sites, who now have the secondary master defined as spooled devices to IMS, in effect writing the messages to physical data sets.

In this way, the secondary master can be used as an online log of events within IMS. To accomplish this, the definitions in Figure 16 needs to be put into the IMS Stage 1 gen. These definitions need to follow the COMM macro and before any VTAM terminal definitions.

```
*  
      LINEGRP DDNAME=(SPL1,SPL2),UNITYPE=SPOOL  
      LINE BUFSIZE=1420  
      TERMINAL FEAT=AUTOSCH  
      NAME      (SEC,SECONDARY)
```

Figure 16. Secondary master spool JCL

To complete the definitions simply code SPL1 and SPL2 DD statements in the IMS Control region JCL. The data sets should be allocated with the following DCB information:

```
DCB=(RECFM=VB,LRECL=1404,BLKSIZE=1414) .
```

8.3 Using the MVS console as master terminal

IMS always has a communications path with the MVS system console. The write-to-operator (WTO) and write-to-operator-with-reply (WTOR) facilities are used for this. Whenever the IMS CTL region is active, there is an outstanding message requesting reply on the MVS system console. This can be used to enter commands for the CTL region. All functions available to the IMS master terminal are available to the system console. The system console and master terminal can be used concurrently, to control the system. Usually, however, the system console's primary purpose is as a backup to the master terminal. The system console is defined as IMS line number one by default.

8.4 Extended MCS/EMCS Console Support

Since IMS Version 6 for IMS TM systems (and IMS Version 3 with DBCTL systems), IMS can be also communicated with, using the MCS/EMCS console support.

Any MVS console can issue a command directly to IMS, using either a command recognition character (CRC) as defined at IMS startup, or using the 4 character IMS-ID to be able to issue commands.

This interface has the option of command security, by using RACF, or exits. For further details, refer to Chapter 24, "IMS security overview" on page 253.

Chapter 9. Application program processing overview

Once an application program is scheduled in a dependent region, it is loaded into that region by IMS.

9.1 MPP processing

After the load of the scheduled program in the MPR, it is given control. The normal processing steps of an MPP are described below in Figure 17.

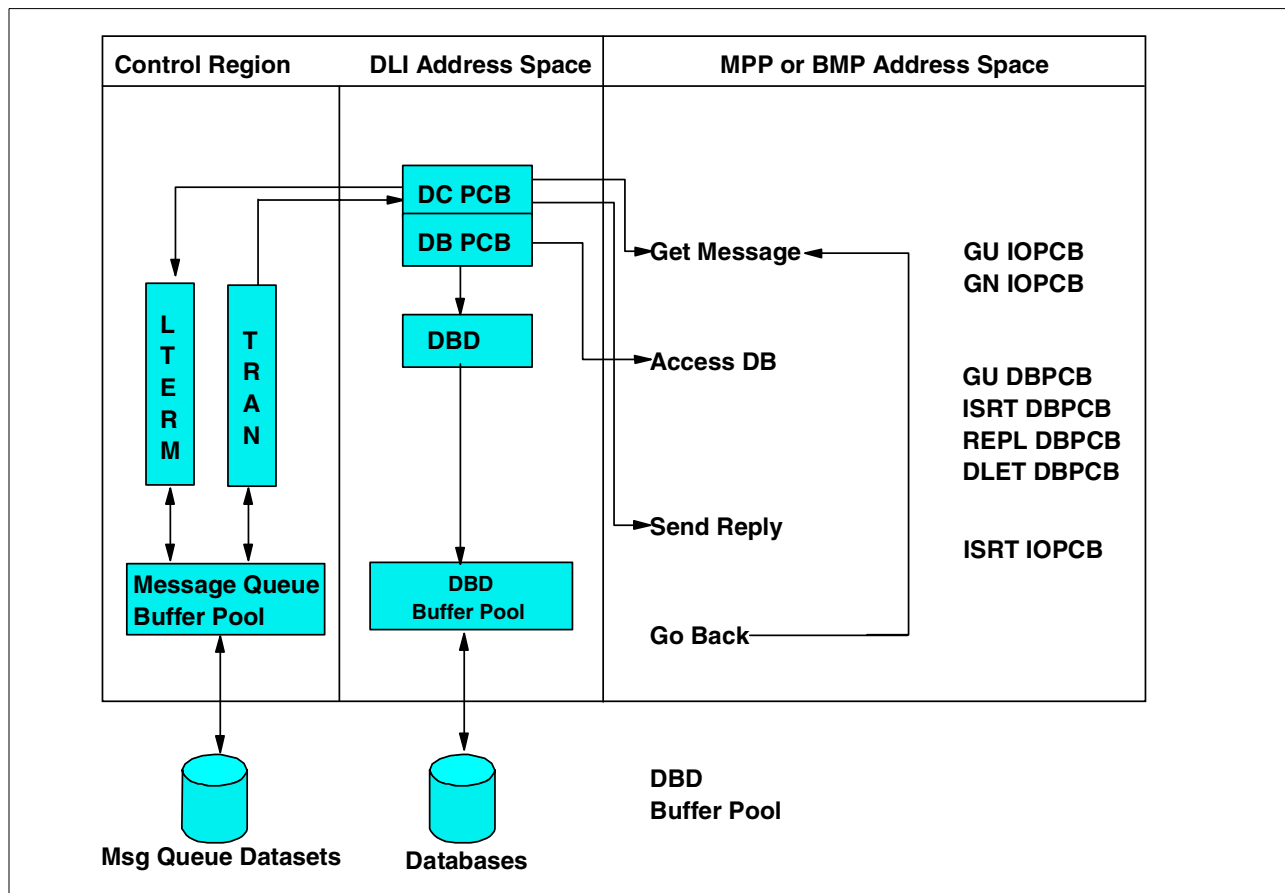


Figure 17. Basic MPP/BMP flow

1. Retrieve the input message via a DL/I message call.
2. Check the input message for syntax errors.
3. Process the input message, requesting necessary DL/I database accesses.
4. Send output to the originating and/or other (for example, printer) logical terminals via DL/I message calls.
5. Retrieve the next input message or terminate.

9.2 Role of the PSB

The program specification block (PSB) for an MPP or a BMP contains, besides database PCBs, one or more PCB (s) for logical terminal linkage. The very first PCB always identifies the originating logical terminal (IOPCB). This PCB must be referenced in the get unique (GU) and get next (GN) message calls. It must also be used when inserting output messages to that LTERM. In addition, one or more alternate output PCBs can be defined. Their LTERM destinations can be defined in the PCBs or set dynamically with change destination calls.

9.3 DL/I message calls

The same DL/I language interface which is used for the access of databases is used to access the message queues.

The principal DL/I message call function codes are:

- GU, get unique. This call must be used to retrieve the first, or only, segment of the input message.
- GN, get next. This call must be used to retrieve second and subsequent message segments.
- ISRT, insert. This call must be used to insert an output message segment into the output message queue. Note: these output message (s) will not be sent until the MPP terminates or requests another input message via a get unique.
- CHNG, change destination. This call can be used to set the output destination for subsequent insert calls.

9.4 Program isolation and dynamic logging

When processing DL/I database calls, the IMS program isolation function will ensure database integrity.

With program isolation, all activity (database modifications and message creation) of an application program is isolated from any other application program(s) running in the system until an application program commits, by reaching a synchronization point, the data it has modified or created. This ensures that only committed data can be used by concurrent application programs. A synchronization point in our subset is established with a get unique call for a new input message (single mode) and/or a checkpoint call (BMP only), or program normal termination (GOBACK or RETURN).

Program isolation allows two or more application programs to concurrently execute with common data segment types even when processing intent is segment update, add, or delete.

This is done by a dynamic enqueue/dequeue routine which enqueues the affected database elements (segments, pointers, free space elements, etc.) between synchronization points.

At the same time, the dynamic log modules log the prior database record images between those synchronization points.

This makes it possible to dynamically back out the effects of an application program that terminates abnormally, without affecting the integrity of the databases controlled by IMS. It does not affect the activity of other application program(s) running concurrently in the system.

With program isolation and dynamic backout, it is possible to provide database segment occurrence level control to application programs. A means is provided for resolving possible deadlock situations in a manner transparent to the application program.

One example of a deadlock occurs in the following sequence of events:

1. Program A updates database element X.
2. Program B updates database element Y.
3. Program A requests Y and must wait for the synchronization point of program B.
4. Program B in turn requests X and must wait for the synchronization point of program A.

A deadlock has now occurred: both programs are waiting for each other's synchronization point. The dynamic enqueue/dequeue routines of IMS intercept possible deadlocks during enqueue processing (in the above example, during enqueue processing of event 4).

Upon detecting a deadlock situation, one of the application programs involved in the deadlock is abnormally terminated (pseudo abend). The activity of the terminated program will be dynamically backed out to a previous synchronization point. Its held resources are freed. This allows the other program(s) to process to completion. The transaction that was being processed by the abnormal terminated program will be saved. The application program is rescheduled if it was an MPP. For a BMP region, the job must be restarted. This process is transparent to application programs and terminal operators.

There are two situations where the enqueue/dequeue routines of program isolation are not used in processing a database call:

1. If PROCOPT=GC (read only) is specified for the referenced segment (s) of the call.
2. If PROCOPT=E (exclusive) is specified for the referenced segment (s) in the call.

Notice that possible conflicts with exclusive extent are resolved during scheduling time and, as such, cannot occur at call time.

Notes:

1. With the GO option, a program can retrieve data which has been altered or modified by another program still active in another region, and database changes made by that program are subject to being backed out.
2. Exclusive intent may be required for long-running BMP programs that do not issue checkpoint calls. Otherwise, an excessively large enqueue/dequeue table in main storage may result.

3. Even when PROCOPT=E is specified, dynamic logging will be done for database changes. The ultimate way to limit the length of the dynamic log chain in a BMP is by using the XRST/CHKP calls. The chain is deleted at each CHKP call because it constitutes a synchronization point.
4. If, as can occur in our subset, one MPP and one BMP get involved in a deadlock situation, the MPP will be subject to the abnormal termination, backout, and reschedule process.

9.5 Internal resource lock manager (IRLM)

When IMS is involved in a data sharing environment with other IMS systems, then IRLM is used instead of program isolation. Refer to 2.1.5, "Internal Resource Lock Manager (IRLM)" on page 19 for further details.

9.6 Application program abnormal termination

Upon abnormal termination of a message or batch-message processing application program for other reasons than deadlock resolution, internal commands are issued to prevent rescheduling. These commands are the equivalent of a /STOP command. They prevent continued use of the program and the transaction code in process at the time of abnormal termination. The master terminal operator can restart either or both stopped resources. At the time abnormal termination occurs, a message is used to the master terminal and to the input terminal that identifies the application program, transaction code, and input terminal. It also contains the system and user completion codes. In addition, the first segment of the input transaction, in process by the application at abnormal termination, is displayed on the master terminal. The database changes of a failing program are dynamically backed-out. Also, any of its output messages that were inserted in the message queue since the last synchronization point are cancelled.

9.7 Conversational processing

A transaction code can be defined as belonging to a conversational transaction during IMS system definition. If so, an application program that processes that transaction, can interrelate messages from a given terminal. The vehicle to accomplish this is the scratch pad area (SPA). A unique scratch pad area is created for each physical terminal which starts a conversational transaction. Each time an input message is entered from a physical terminal in conversational mode, its SPA is presented to the application program as the first message segment (the actual input being the second segment). Before terminating or retrieving another message (from another terminal), the program must return the SPA to the control region with a message ISRT call. The first time a SPA is presented to the application program when a conversational transaction is started from a terminal, IMS will format the SPA with binary zero's (X'00). If the program wishes to terminate the conversation, it can indicate this by inserting the SPA with a blank transaction code.

9.8 Output Message Processing

As soon as an application reaches a synchronization point, its output messages in the message queue become eligible for output processing. A synchronization point is reached whenever the application program terminates or requests a new message/SPA from the input queue via a GU call.

In general, output messages are processed by the message format service before they are transmitted via the telecommunications access method.

Different output queues can exist for a given LTERM, depending on the message origin. They are, in transmission priority:

1. Response messages, that is, messages generated as a direct response (same PCB) to an input message from this terminal.
2. Command responses.
3. Alternate output messages, messages generated via an alternate PCB.

9.9 Logging and checkpoint / restart

To ensure the integrity of its databases and message processing, IMS uses logging and checkpoint/restart. In case of system failure, either software or hardware, IMS can be restarted. This restart includes the repositioning of users' terminals, transactions, and databases.

9.9.1 Logging

For further information on IMS logging facilities, refer to Chapter 22, "IMS logging" on page 239.

9.9.2 Checkpointing

At regular intervals during IMS execution, checkpoints are written to the logs. This is to limit the amount of reprocessing required in the case of emergency restart. A checkpoint is taken after a specified number of log records are written to the log tape after a checkpoint command. A special checkpoint command is available to stop IMS in an orderly manner.

A special disk restart data set is used to record the checkpoint identification and log tape volume serial numbers. This restart data set (IMSVS.RDS) is used during restart for the selection of the correct restart checkpoint and restart logs.

9.10 Message Switching

In this case, a program that is already executing can request that a new transaction be put on the IMS message queues for standard scheduling and execution.

This second transaction:

- Can continue the processing of the first transaction (which, in this case, has probably terminated), and respond (if required) to the originating terminal, which is probably still waiting for a response.

- Can be a second transaction, purely an offshoot from the first, without any relationship or communications with the originating terminal. In this case, the original transaction must respond to the terminal, if required.

The basic format of a message switch is the destination LTERM name followed by a blank and the message text.

Part 3. IMS Database Manager

This part consists of the following chapters:

- An introduction to database basics: what is a database, and why would you use one? There is also a discussion of the role of a central database administrator. Refer to Chapter 10, “Database basics” on page 63.
- An overview of the IMS hierarchical database model. Refer to Chapter 11, “The IMS hierarchical database model” on page 69.
- A description of the physical implementation of the IMS database model. Refer to Chapter 12, “Implementation of the IMS database model” on page 79.
- Some reasons why one database type would be chosen over another. Refer to Chapter 13, “Choosing the correct type of database” on page 103.
- The database reorganization tasks that need to be performed by the IMS database administrator function. Refer to Chapter 14, “Database reorganization processing” on page 109.
- The backup and recovery tasks that need to be performed by the IMS database administrator function. Refer to Chapter 15, “Database recovery processing” on page 123.

Chapter 10. Database basics

This chapter gives an overview of basic database concepts. The database models discussed are intentionally kept simple, as is the description of the design choices for IMS databases. Many more design options are available for IMS databases than are addressed in this chapter. For more details on database models, please visit your favorite local bookshop. For more details on IMS database design options, see the IMS library.

10.1 The database design process

The process of database design, in its simplest form, can be described as follows: The structuring of the data elements for the various applications, in such an order that:

- Each data element is readily available by the various applications, now and in the foreseeable future.
- The data elements are efficiently stored.
- Controlled access is enforced for those data elements with specific security requirements.

Database design is an area in which a number of different models for databases have been developed over the years (such as relational or object) so that there is no consistent vocabulary for describing the concepts involved.

10.1.1 Entities

A database contains information about entities. An entity is something that:

- Can be uniquely defined
- We may collect substantial information about, now or in the future

In practice, this definition is limited to the context of the applications and/or business under consideration. Examples of entities are: parts, projects, orders, customers, trucks, etc. It should be clear that defining entities is a major step in the database design process. The information we store in databases about entities is described by data attributes.

10.1.2 Data attributes

A data attribute is a unit of information that specifies a fact about an entity. For example, suppose the entity is a part. Name=Washer, Color=Green, and Weight=143 are three facts about that part. Thus these are three data attributes. A data attribute has a name and a value. A data attribute name tells the kind of fact being recorded; the value is the fact itself. In the above example, Name, Color, and Weight are data attribute names; while Washer, Green, and 143 are values. A value must be associated with a name to have a meaning.

An occurrence is the value of a data attribute for a particular entity. An attribute is always dependent on an entity. It has no meaning by itself. Depending on its usage an entity can be described by one single data attribute or more. Ideally, an entity should be uniquely defined by one single data attribute, for example, the order number of an order. Such a data attribute is called the key of the entity. The

key serves as the identification of a particular entity occurrence, and is a special attribute of the entity. Keys are not always unique. In such cases, entities with equal key values are called synonyms. For instance, the full name of a person is generally not a unique identification. In such cases we have to rely on other attributes such as full address, birthday or an arbitrary sequence number. A more common method is to define a new attribute, which serves as the unique key, for example, employee number.

10.1.3 Entity relationships

The entities identified will also have connections between them, *relationships*. For example, an order will be for a number of parts. Again these relationships only have meaning within the context of the application and business.

These relationships can be one to one (that is, one occurrence of an entity relates to a single occurrence of another entity), one to many (one occurrence of an entity relates to many occurrences of another entity) or many to many (many occurrences of one entity have a relationship with many occurrences of another entity).

Relationships can also be recursive, an entity can have a relationship with other occurrences of the same entity. For example a part, say a fastener, may consist of several other parts, bolt, nut, washer.

10.1.4 Application functions

Data itself is not the ultimate goal of a database management system. It is the application processing performed on the data which is important. The best way to represent that processing is to take the smallest application unit representing a user interacting with the database. For example, one single order, one part inventory status. In the following chapters we will call this an application function.

Functions are processed by application programs. In a batch system, large numbers of functions are accumulated into a single program (that is, all orders of a day), then processed against the database with a single scheduling of the desired application program. In the online system, just one or two functions may be grouped together into a single program to provide one iteration with a user. Although functions are always distinguishable, even in batch, some people prefer to talk about programs rather than functions. But, especially in a DB/DC environment, a clear understanding of functions is mandatory for good design. Once you have identified the functional requirements of the application, you can decide how to best implement them as programs using IMS. The function is, in some way, the individual usage of the application by a particular user. As such, it is the focal point of the DB/DC system.

10.1.5 Access paths

Each function bears in its input some kind of identification with respect to the entities used (for example, the part number when accessing a Parts database). These are referred to as the access paths of that function. In general, functions require random access, although for performance reasons sequential access is sometimes used. This is particularly true if the functions are batched, and if they are numerous, relative to the database size, or if information is needed from most database records.

For efficient random access, each access path should utilize the entities key. With proper database design, DL/I generally provides fast physical access via a key. Therefore, identification of the function's access path is essential for a design to yield good performance.

10.1.6 Normalization

One formal technique that is used with entity/relationship data models is normalization, sometimes referred to as getting the data model to third normal form. This technique is not covered in detail, but is mentioned here, as some of the techniques used are useful for getting the data model to a form where you can design the hierarchical structures for an IMS database. Some of the goals of normalization, which are relevant to the final IMS database design, are to get the data model to a state where:

1. All entities are uniquely identified.
2. There is only one occurrence of each data attribute in each entity, that is, it has no repeating fields.
3. There are no many-to-many relationships between entities.

To transfer the data model to a physical IMS database design, you must achieve the first goal for entities that will become root segments. It can also aid the design if you try and achieve this for other entities that will become dependents.

To achieve the second goal, normalization will create a new entity to contain each of the occurrences of the data attribute, that will have a one to many relationship with the original entity. If there is only a fixed number of occurrences of the attribute, and you are certain there will only ever be that maximum number of occurrences, then you may want to leave the attribute in the entity. If you are unsure of the number of occurrences, its better to create another entity.

If you leave a many to many relationship in the data model, then it is not possible to implement this in an IMS database. Normalization will create another new entity in the path of the relationship that has a one to many relationship with each of the two original entities.

If you do decide to normalize the data model, you should adopt a pragmatic approach when you map the design to physical IMS databases. You may want to go back on some of the changes you made for normalization for performance reasons. Normalization is just a technique for getting a data model that provides a sound starting point for the physical design.

10.2 What is a database ?

A database provides for the storing and control of business data, independent from (but not separate from the processing requirements of) one or more applications. If properly designed and implemented, the database should provide a single consistent view of the business data, that can be centrally controlled and managed.

One way of describing a logical view of this collection of data is to use an entity/relationship model. The database will record details (attributes) of particular items (entities) and the relationships between the different types of entities. For example, for the stock control area of an application, you would have Parts, Purchase Orders, Customers, Customer Orders (entities). Each entity would have attributes, the PART would have a Part No, Name, Unit Price, Unit Quantity, etc. These entities would also have relationships between them; a Customer would be related to orders he had placed, these would be related to the part that had been ordered, and so on. Figure 18 illustrates an entity relationship mode.

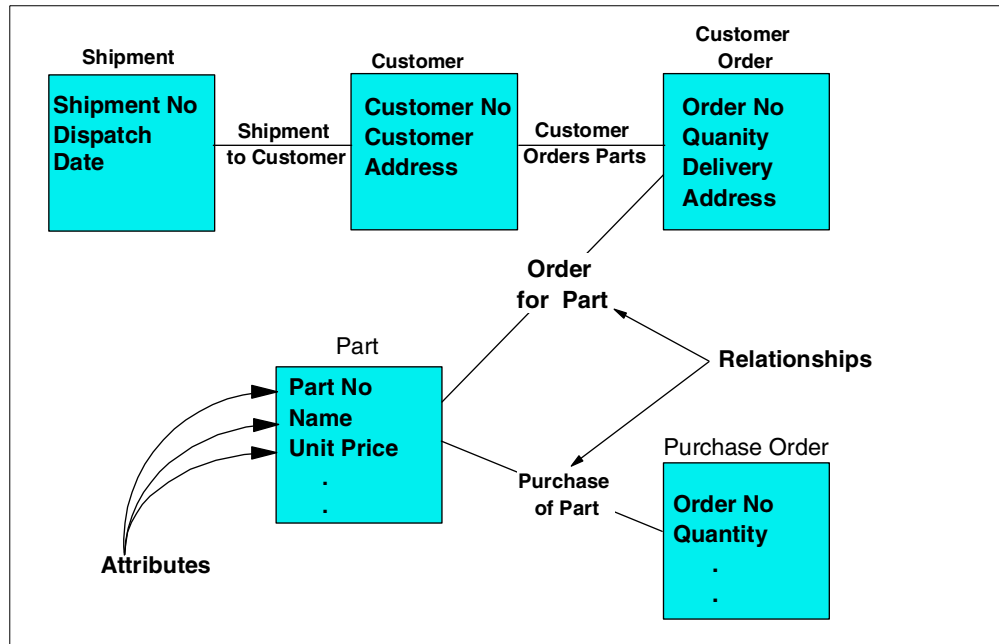


Figure 18. Entities, attributes, and relationships

A database management system (DBMS), such as the IMS Database Manager component, or the DB2 product, provide a method of storing and using the business data in the database

10.3 Why use a database ?

When computer systems were first developed, the data was stored on individual files, unique to an application, or even a small part of an individual application.

The same details might be stored in several different places, for example the details of a customer might be in both the ordering and invoicing application. This caused a number of problems:

- As the details were stored and processed independently, details that were supposed to be the same, for example a customers name and address, might be inconsistent in the various different applications.
- When common data had to be changed, it had to be changed in several places, causing a high workload. If any copies of the data were missed, it resulted in the problems detailed in the previous point.

- There was no central point of control for the data, to ensure it was secure, both from loss and from unauthorized access
- The duplication of the data wasted space on storage media.

The use of a database management system, such as the IMS Database Manager component, to implement the database also provides additional advantages:

- The DBMS allows multiple tasks to access and update the data simultaneously while preserving database integrity. This is particularly important where large numbers of users are accessing the data via an online application.
- The DBMS will provide facilities for the application to update multiple databases/database records and ensure the application data in the different records remains consistent even if an application failure occurs.
- The DBMS provides utilities that control and implement backup and recovery of the data, preventing loss of vital business data.
- The DBMS will provide utilities to monitor and tune the access to the data.

The use of a database and database management system will not, in itself, produce the advantages detailed above. It also requires the proper design and administration of the databases, and development of the applications. This book describes, by use of examples, one way of doing this using the IMS product.

10.4 The database administrator role

The centralization of data and control of access to this data is inherent to a database management system. One of the advantages of this centralization is the availability of consistent data to more than one application. As a consequence, this dictates tighter control of that data and its usage. Responsibility for an accurate implementation of control lies with the database administrator (DBA) function. Indeed, to gain the full benefits of using a centralized database, you must have a central point of control for it. Because the actual implementation of the DBA function is dependent on a company's organization, we limit ourselves to a discussion of the roles and responsibilities of a DBA. The group fulfilling the DBA role will need experience in both application and systems programming.

DBA roles and responsibilities in a typical installation might be as follows:

- The DBA provides standards for, and controls the administration of, the databases and their use.
- The DBA provides guidance, review, and approval of database design.
- The DBA determines the rules of access to the data and monitors their security.
- The DBA controls the database integrity and availability, monitoring the necessary activities for reorganization backup/recovery.
- The DBA is not responsible for the actual content of databases — this is the responsibility of the user. Rather, the DBA enforces procedures for accurate, complete, and timely update of the databases.
- The DBA approves the operation of new programs with existing production databases, based on results of testing with test data.

In general, the DBA is responsible for the maintenance of current information about the data in the database. Initially, this responsibility might be carried out using a manual approach. But it can be expected to grow to a scope and complexity sufficient to justify, or necessitate, the use of a data dictionary program.

Chapter 11. The IMS hierarchical database model

IMS uses a hierarchical model as the basic method of storing data. Unlike the relational model used by DB2, which was the result of theoretical work, this was arrived at as a pragmatic way of storing the data and implementing the relationships between the various type of entities.

In this model, the individual entity types are implemented as segments in a hierarchical structure. The hierarchical structure is determined by the designer of the database, based on the relationship between the entities and the access paths required by the applications.

Note that in the IMS program product itself, the term database is used slightly differently to its use in other DBMS's. In IMS, a database is commonly used to describe the implementation of one hierarchy, so that an application would normally access a large number of IMS databases. Compared to the relational model, an IMS database is approximately equivalent to a table.

The hierarchical data structure in Figure 19 describes the data as seen by the application program. It does not represent the physical storage of the data. The physical storage is of no concern to the application program.

The basic building element of a hierarchical data structure is the parent/child relationship between segments of data, also illustrated in Figure 19.

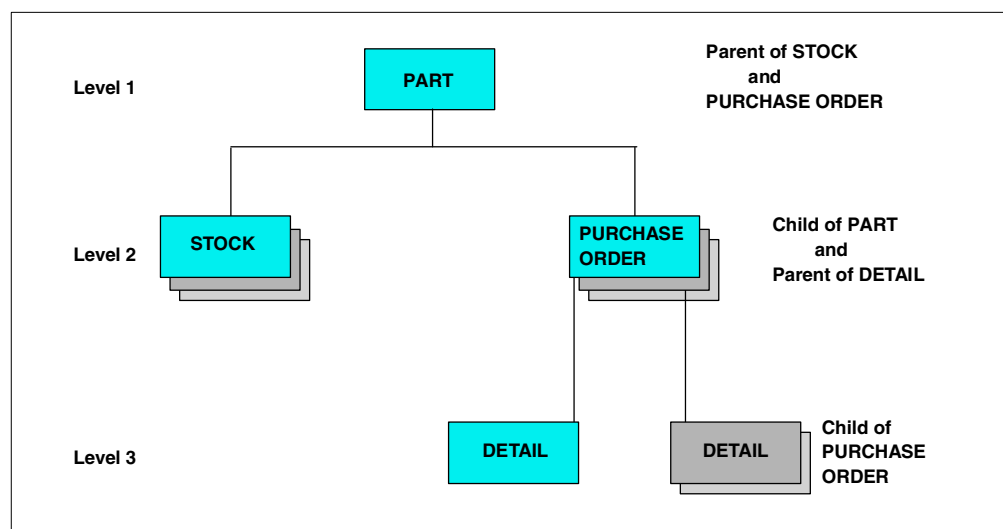


Figure 19. Hierarchical data structure

Each occurrence (or instance) of a parent segment has associated with it 0, 1, 2, or more occurrences of a child segment. Each child segment occurrence has associated with it one, and only one, occurrence of a parent segment.

Sometimes it is necessary to distinguish between a segment type, that is, the kind of segment; and the segment occurrence, that is, the particular instance of its contents and location.

As shown in Figure 19, a parent can have several child segment types. Also, a child segment can, at the same time, be a parent segment; that is, it can have children itself. The segment with no parent segment, that is, the one at the top, is called the root segment.

All the parent/child occurrences for a given root segment are grouped together in a DL/I database record. The collection of all these like database records is a DL/I database.

Only one segment can appear at the first level in the hierarchy, but multiple segments can appear at lower levels in the hierarchy. For example, multiple STOCK and ORDER segments can exist for one PART segment. Since each dependent segment in the hierarchy has only one parent, or immediate superior segment, the hierarchical data structure is sometimes called a tree structure. Each branch of the tree is called a hierarchical path. A hierarchical path to a segment contains all consecutive segments from the top of the structure down to that segment.

In Figure 19, each PART segment with its dependent STOCK, ORDER, and DETAIL segments constitutes a database record. The collection of all these records for all PARTS is called a database, that is, the PARTS database.

Through the concept of program sensitivity, DL/I allows a program to be restricted to “seeing” only those segments of information that are relevant to the processing being performed. For example, an inventory program could be written to see only the PART and STOCK segments of the database record shown in Figure 19 on page 69. The program need not be aware of the existence of the ORDER segment.

DL/I allows a wide variety of data structures. The maximum number of different segment types is 255 per hierarchical data structure. A maximum of 15 segment levels can be defined in a hierarchical data structure. There is no restriction on the number of occurrences of each segment type, except as imposed by physical access method limits.

11.1 Basic segment types in a hierarchical data structure

Following is a detailed description of the various segment types and their interrelations within the hierarchical data structure. Refer to Figure 19 on page 69 and Figure 20 on page 71 when reading this description.

- The segment on top of the structure is the root segment. Each root segment normally has a key field which serves as the unique identifier of that root segment, and as such, of that particular database record (for example, the part number).
- A dependent segment relies on the segments above it in the hierarchy for its full meaning and identification.
- A parent/child relationship exists between a segment and its immediate dependents.
- Different occurrences of a particular segment type under the same parent segment are twin segments.
- Segment occurrences of different types under the same parent are sibling segments.

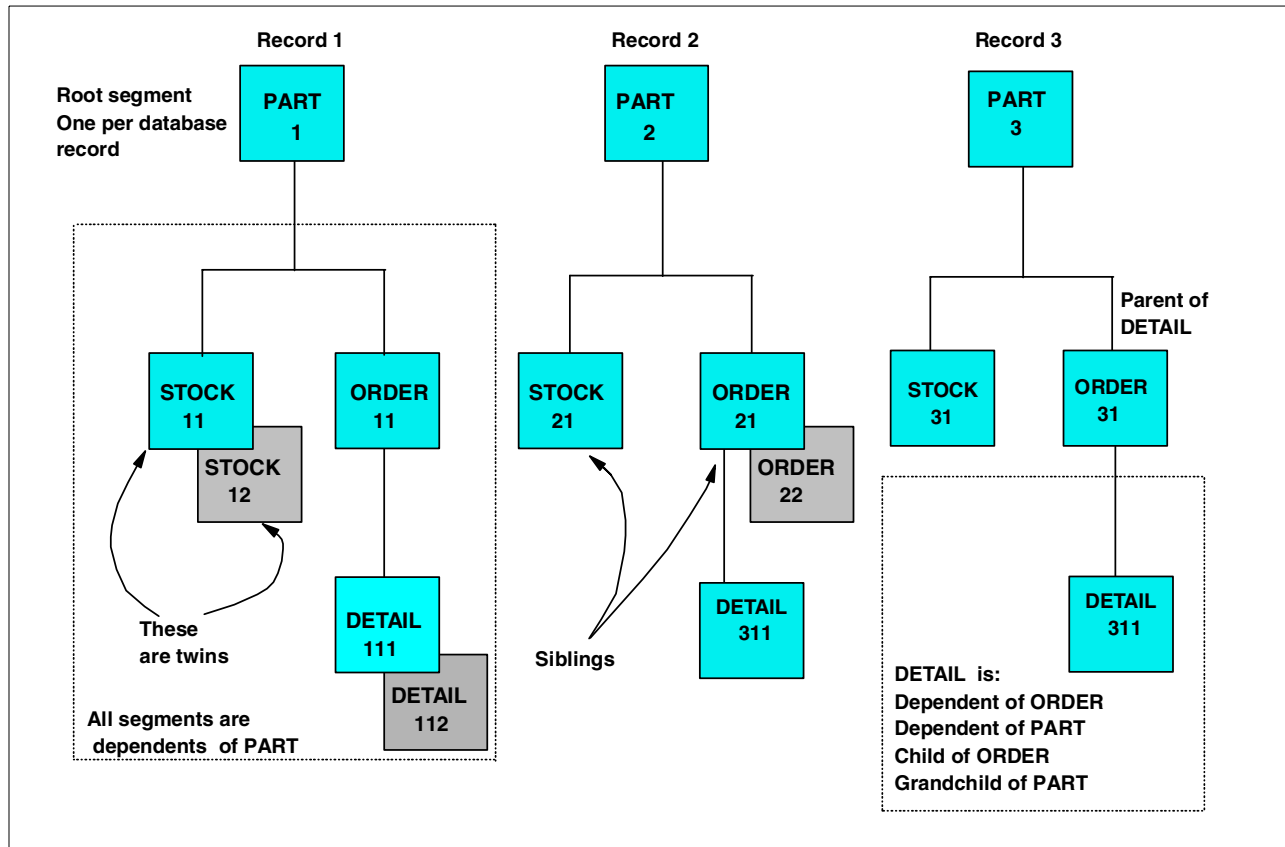


Figure 20. Segment types and their relationships

11.2 Sequence fields and access paths

To identify and to provide access to a particular database record and its segments, DL/I uses sequence fields. Each segment normally has one field denoted as the sequence field. The sequence fields in our subset should be unique in value for each occurrence of a segment type below its parent occurrence. However, not every segment type need have a sequence field defined. Particularly important is the sequence field for the root segment, since it serves as the identification for the database record. Normally, DL/I provides a fast, direct access path to the root segment of the database record based on this sequence field. This direct access is extended to lower level segments if the sequence fields of the segments along the hierarchical path are specified, too.

Note: The sequence field is often referred to as the key field, or simply the key.

Using Figure 20, an example of an access path would be the PART, ORDER and DETAIL segments. It must always start with the root segment. This is the access path as used by DL/I. The application program, however, can directly request a particular DETAIL segment of a given ORDER of a given PART in one single DL/I request, by specifying a sequence field value for each of the three segment levels.

11.3 Additional access paths to segments

In addition to the basic DL/I facilities discussed so far, IMS provides two additional methods of defining access paths to a database segment. These are:

- Logical relationships
- Secondary indices

Both provide a method for an application to have a different access path to the physical databases. They are defined to IMS in addition to the basic hierarchical structure already defined. The logical relationships and secondary indices are automatically maintained by IMS, transparent to the application.

Logical relationships allow a logical view to be defined of one or more physical databases. To the application this will look like a single database.

Secondary indices give an alternate access path, via a root or dependent segment, to the database record in one physical database.

You should only use these extra facilities if there are strong application and/or performance reasons for doing so. Both involve additional overheads. The following two sections (11.4, “Logical relationships” on page 72, and 11.5, “Secondary indexing” on page 76) describe these facilities in more detail and indicate where you might wish to use them.

11.4 Logical relationships

Through logical relationships, DL/I provides a facility to interrelate segments from different hierarchies. In doing so, new hierarchical structures are defined which provide additional access capabilities to the segments involved. These segments can belong to the same database or to different databases. A new database can be defined called a logical database. This logical database allows presentation of a new hierarchical structure to the application program. Notice that although the connected physical databases could constitute a network data structure, the application data structure still consists of one or more hierarchical data structures.

For example, given the entities and relationships illustrated in Figure 21, it may have been decided that, based on the applications most common access paths, the data should be implemented as two physical databases, with hierarchies as shown in Figure 22 on page 74. However, there are some reasons why other applications need to use the relationship between the PART and order DETAIL (reasons for wanting to do this are discussed below). So a logical relationship is to be built between PART and DETAIL.

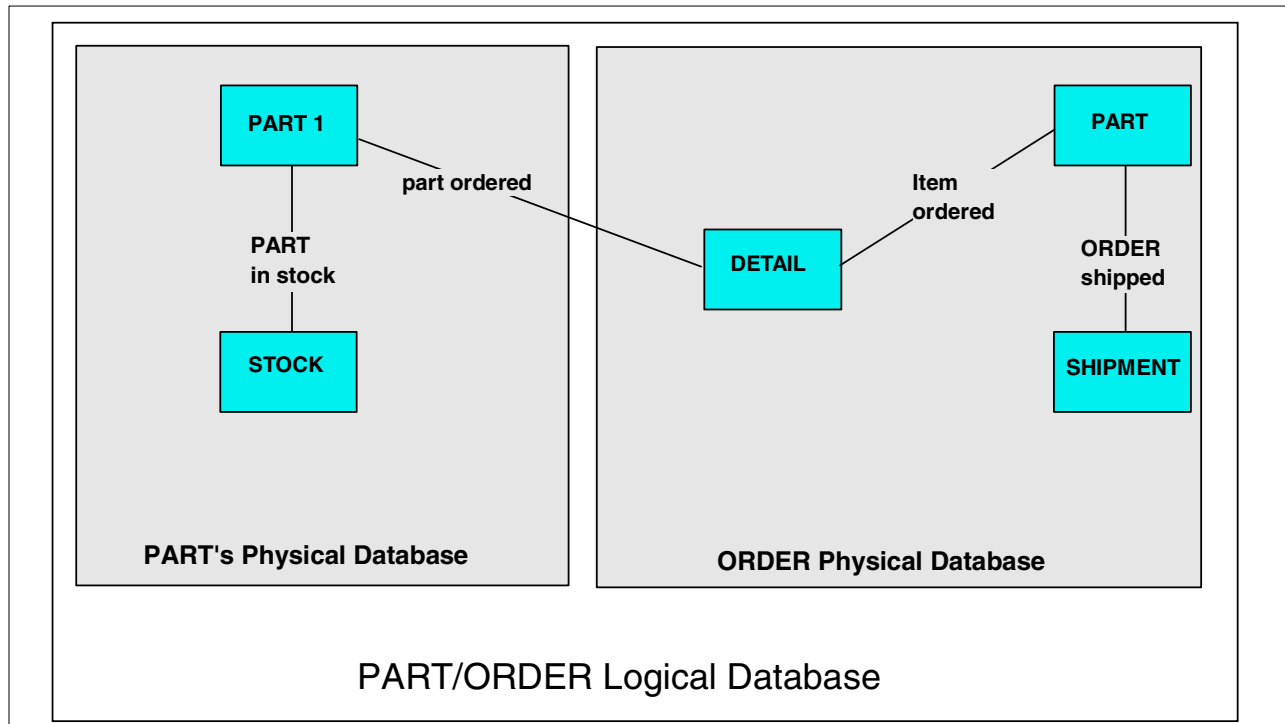


Figure 21. Entities and relationships with physical and logical databases mapped on

The basic mechanism used to build a logical relationship is to specify a dependent segment as a logical child, by relating it to a second parent, the logical parent.

In Figure 22, the logical child segment **DETAIL** exists only once, yet participates in two hierarchical structures. It has a physical parent, **ORDER**, and logical parent, **PART**. The data in the logical child segment and in its dependents, if any, are called intersection data.

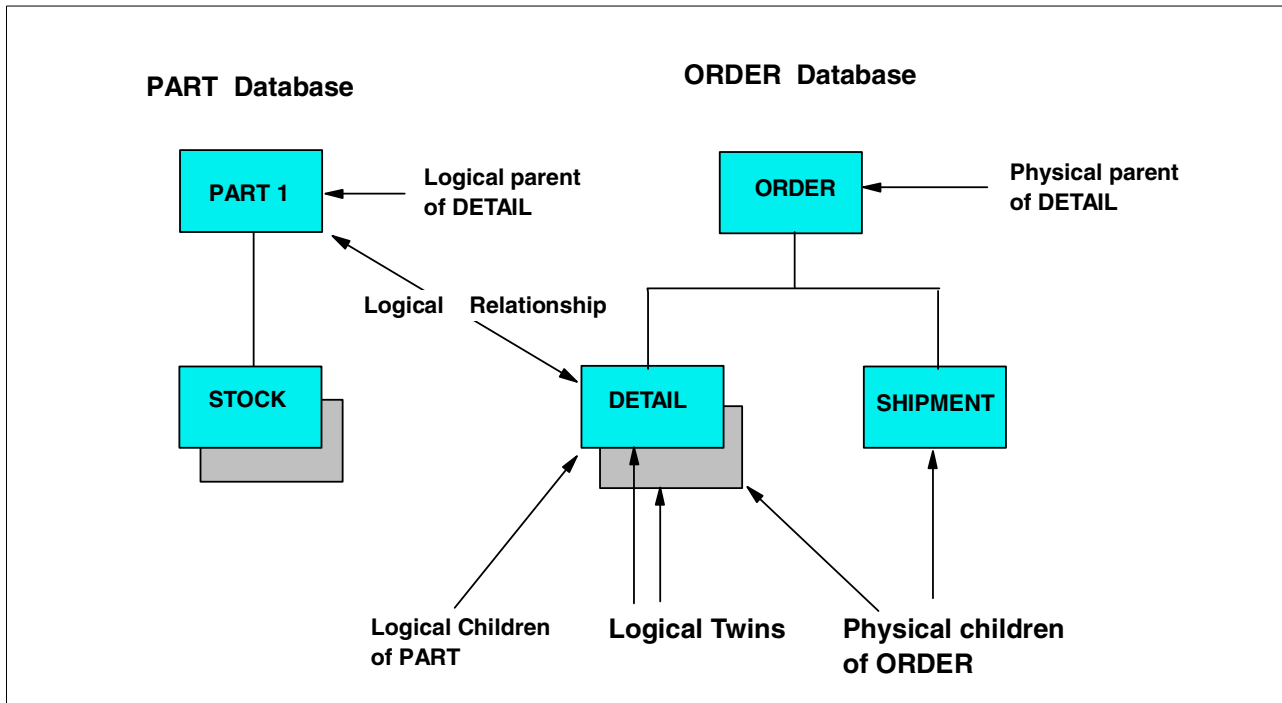


Figure 22. Two Logically related physical databases, PARTS and ORDERS

By defining two additional logical databases, two new logical data structures shown in Figure 23 can be made available for application program processing, even within one single program.

The DETAIL/PART segment in Figure 23 is a concatenated segment. It consists of the logical child segment plus the logical parent segment. The DETAIL/ORDER segment in Figure 23 is also a concatenated segment, but it consists of the logical child segment plus the physical parent segment. Logical children with the same logical parent are called logical twins, for example, all DETAIL segments for a given PART segment. As can be seen in Figure 22, the logical child has two access paths. One via its physical parent, the physical access path, and one via its logical parent, the logical access path. Both access paths are maintained by DL/I and can be concurrently available to one program.

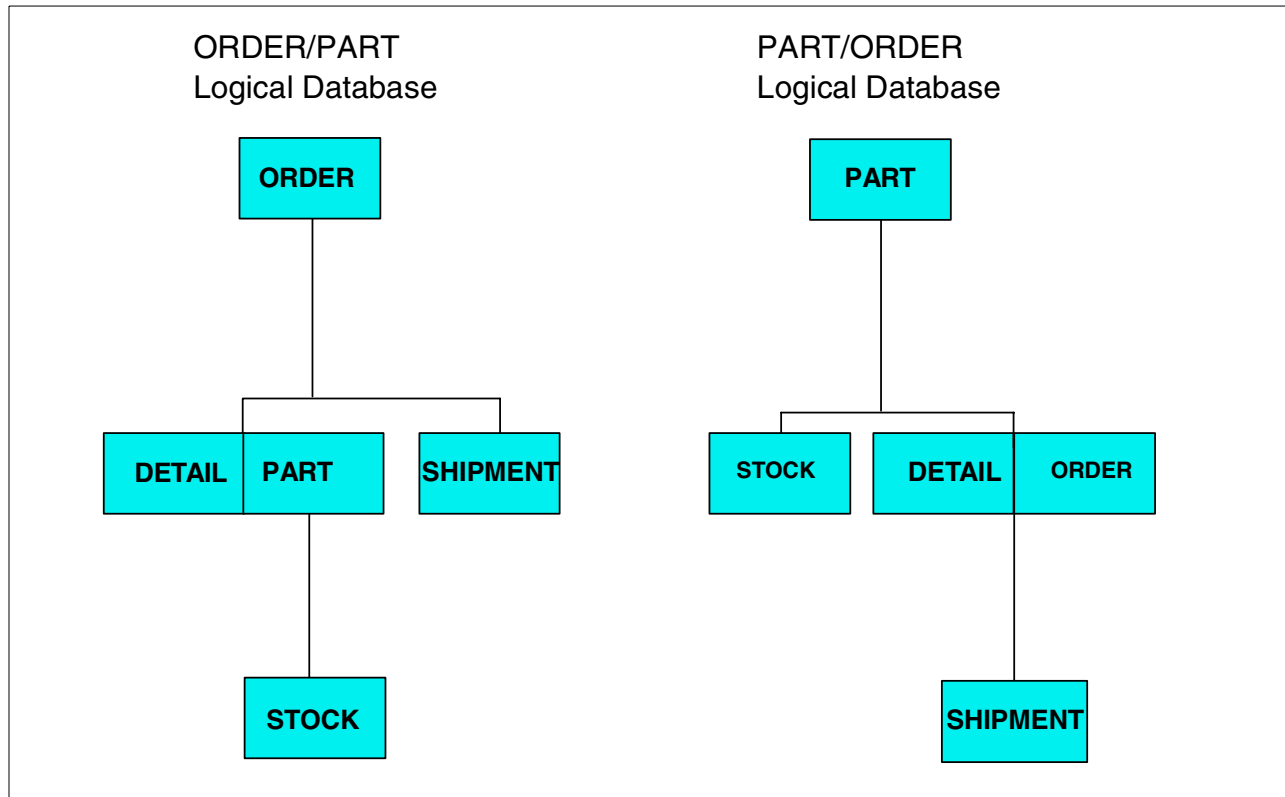


Figure 23. Two additional logical DB's after relating PARTS and ORDER DB's

Some reasons you may want to use logical relationships are:

- They provide an alternate access path for the application. For example, they allow (depending on pointer choice) an application to have direct access from a segment in one physical database to a lower level segment in another physical database, without the application having to access the second physical database directly and read down through the hierarchy.
- They provide an alternate hierarchical database structure for an application, so that different applications, or parts of applications, can have a view of the physical databases that most closely matches that application's view of the data.
- They can make IMS enforce a relationship between two segments in two physically separate databases (that is, it will preserve referential integrity). You can define the relationship such that a logical parent cannot be deleted if it still has logical children, and a logical child cannot be added if there is no logical parent. For example, referring to Figure 22 on page 74, you could define the relationship such that no order DETAIL could be inserted if there were no corresponding PART, and no PART could be deleted if there were still order DETAILS for that part. Any application attempting to do this would have the database call rejected by IMS.

Potential disadvantages in using logical relationships are:

- There are performance overheads in maintaining the pointers used in the logical relationships. Every time a segment participating in a logical relationship is updated, the other segment (in another physical database) that participates in the relationship may need to be updated. This can result in an appreciable increase in physical I/Os to auxiliary storage.
- When a database needs to be reorganized, except with some very limited pointer choices, all other databases that are logically related must be reorganized at the same time, as the pointers used to maintain the logical relationships rely on the physical position of the segments in that database, which can be altered by the reorganization.

Before choosing to use logical relationships, you need to carefully weigh up the performance and administrative overheads against the advantages of using logical relationships.

For further details on implementing logical relationships refer to *IMS/ESA Administration Guide: Database Manager, SC26-8012*

11.5 Secondary indexing

DL/I provides additional access flexibility with secondary index databases. Each secondary index represents a different access path to the database record other than via the root key. The additional access paths can result in faster retrieval of data. For example, the PART and ORDER segments in Figure 24 could be retrieved based on the order number in the ORDER segment, if an index were defined for that field. Once an index is defined, DL/I will automatically maintain the index if the data on which the index relies changes, even if the program causing that change is not aware of the index.

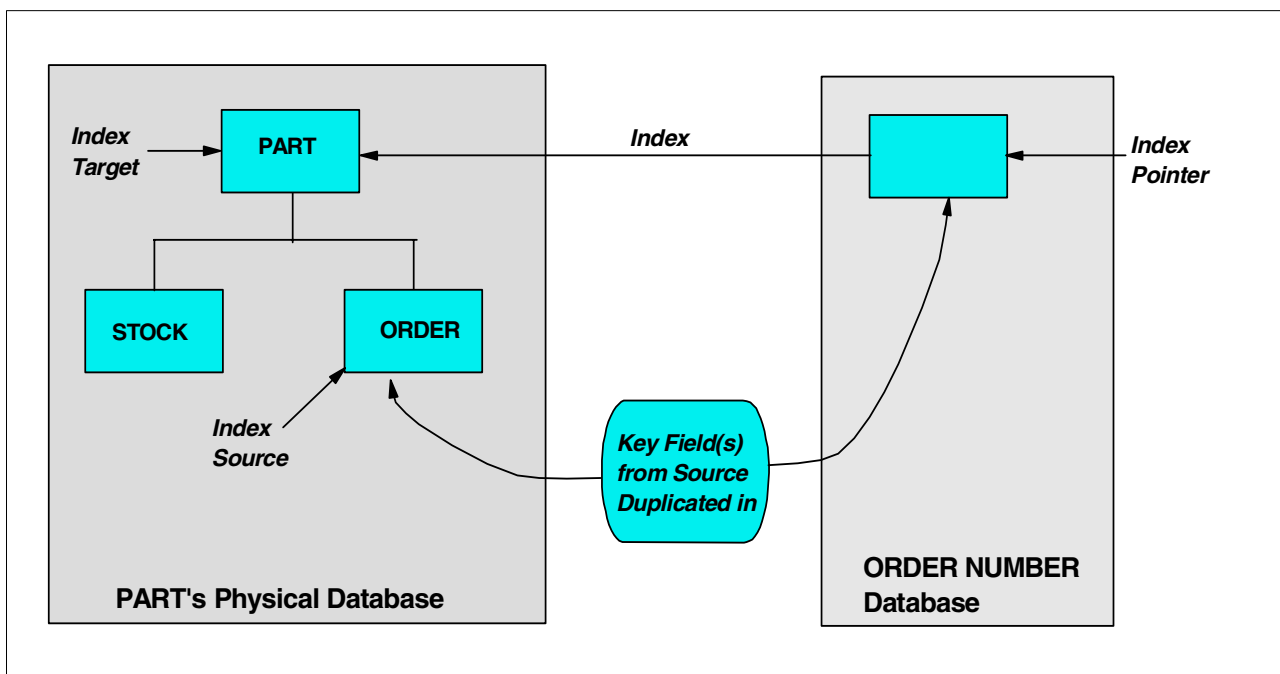


Figure 24. A database and its secondary index database

The secondary index is implemented by defining a secondary Index database to IMS. This contains segments that point to the segment in the main physical database that contains the key values that constitute the secondary index key. As this Index database is itself a physical database, it may be accessed independently by the applications.

The segments involved in a secondary index are depicted in Figure 24.

The index source segment contains the source field (s) on which the index is constructed, for example, ORDER#.

- The index pointer segment is the segment in the index database that points to the index target segment. The index pointer segments are ordered and accessed based on the field (s) contents of the index source segment, for example, the order number. This is the secondary processing sequence of the indexed PARTS database. There is, in general, one index pointer segment for each index source segment, but multiple index pointer segments can point to the same index target segment.
- The index target segment is the segment which becomes initially accessible via the secondary index. It is in the same hierarchical record as the index source segment and is pointed to by the index pointer segment in the index database. Quite often, but not necessarily, it is the root segment.
- The index source and index target segment may be the same, or the index source segment may be a dependent of the index target segment as shown in Figure 24.

The secondary index key (search field) is made up of from one to five fields from the index source segment. The search field does not have to be a unique value, but it is strongly recommended you make it a unique value to avoid the overhead in storing and searching duplicates. There are a number of fields that can be concatenated to the end of the secondary index search field to make it unique:

- A subsequence field, consisting of from one to five more fields from the index source segment. This is maintained by IMS but, unlike the search field, cannot be used by an application for a search argument when using the secondary index.
- A system defined field that uniquely defines the index source segment, the /SX variable.
- A system defined field that defines the concatenated key (i.e. the concatenation of the key values of all the segment occurrences in the hierarchical path leading to that segment) of the index source segment, the /CX variable.

A further technique that can be used with secondary indices is sparse indexing. Normally IMS will maintain index entries for all occurrences of the secondary index source segment. However it is possible to cause IMS to suppress index entries for some of the occurrences of the index source segment. You may wish to do this if, say, you were only interested in processing segments that had a non-null value in the field. In the example in Figure 24 say that the ORDER had a field set in it to indicate the order could not be fulfilled immediately, but needed to be back ordered. You could define a secondary index including this field, but suppress all entries that did not have this field set, giving rapid access to all back orders. As a rule of thumb only consider this technique if you expect 20% or less

of the index source segments to be created. The suppression can be either by specifying all bytes in the field should be a specific character (NULLVAL parameter) or by selection with the Index maintenance exit routine.

Some reasons you might want to use secondary indices are:

- Quick access, particularly random access by online transactions, by a key which is not the primary key the database has been defined with.
- Access to the index target segment without having to negotiate the full database hierarchy (particularly useful if the index target segment is not the root segment). This is similar to using logical relationships, but provides a single alternate access path into a single physical database. If this is all that is required, then a secondary index is the better technique to use.
- Ability to process the index database separately, for example by a batch process that needs to process only the search fields (but see also in the manuals about adding user data to the secondary index database,
- A quick method of accessing a small subset of the database records via a sparse index.

Potential disadvantages in using secondary indices are:

- The performance overheads in updating the secondary Index database every time any of the fields making up the search field in the index source segment is updated, or the index source segment is inserted or deleted.
- The administrative overheads in setting up, monitoring, backing up and tuning the secondary index database.
- When the database containing the index source segment is reorganized, the secondary index must also be re-built as the pointers used to maintain the connection between the source segment and the secondary index database rely on the physical position of the source segment in the database, which can be altered by the reorganization.

As with logical relationships, consider carefully whether the benefits of using a secondary index outweigh the performance and administrative overheads.

For further details on implementing secondary indices, refer to *IMS/ESA Administration Guide: Database Manager*, SC26-8012. The index maintenance exit routine is described in *IME/ESA Customization Guide*, SC26-8020.

Chapter 12. Implementation of the IMS database model

The previous chapter described the logical model for IMS databases. This chapter looks at how this model is physically implemented using the IMS Database Manager component, and OS/390 services.

The application interfaces with IMS, both Database Manager and Transaction Manager components, through functions provided by the IMS DL/I application programming interface (API). In this section we will only address the functions relevant to IMS Database Manager.

The individual elements that make up the database, segments, and database records, are organized using a number of different IMS access methods: HDAM, HIDAM, DEDB, etc. These methods are described in detail below. The choice of access method can influence the functionality available to your application, the order in which data is returned to the application, the functionality available to the application, and the performance the application receives from the Database Manager.

Underlying these IMS access methods, IMS uses various operating system access methods to store the data on DASD and move the data between the DASD and the buffers in the IMS address space, where it is manipulated.

The structure of the IMS databases, and a program's access to them, is defined by a set of IMS control blocks, the DBD, PSB and ACB. These are coded as sets of source statements that then have to be generated into control blocks for use by the Database Manager and application. This structure is described in Figure 25.

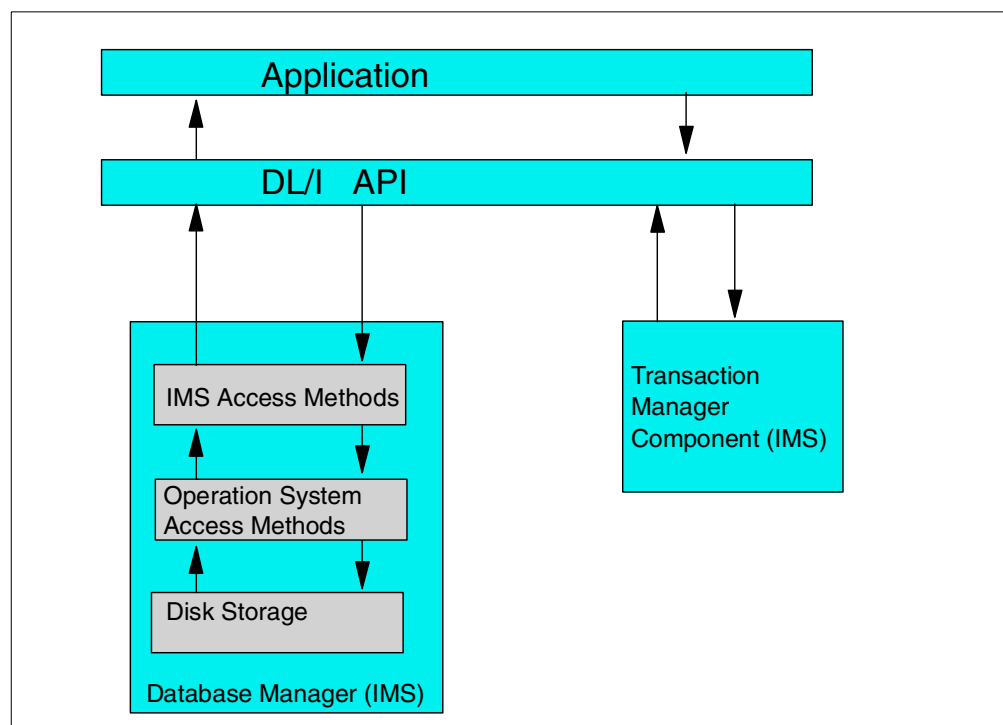


Figure 25. Elements of the physical implementation

12.1 Segments, records, and pointers

As described above, a segment is used to represent one entity, or grouping of related fields. In IMS, unlike DB2 or many other DBMSs, it is not mandatory to define all the fields to IMS, it is only necessary to define the segment as being long enough to contain all the application data to be stored. The only fields you must define to IMS are those you wish to use for identifying and searching for segments. Specifying non-search fields (field level sensitivity) is optional.

One occurrence of a root segment, with all its dependent segments, is referred to as a single database record.

In addition to the application data, each segment will also contain control information used by IMS. This is at the start of the segment, in a segment prefix. Figure 26 shows the layout of a segment with the prefix and application data portions. This is automatically maintained by IMS, and is transparent, and not accessible to, the application. This control information consists of various flags and descriptive fields (segment type and delete byte) and pointers to implement the hierarchical structure and access paths. The contents of the prefix will vary, depending on the IMS access method and options chosen when the database is defined.

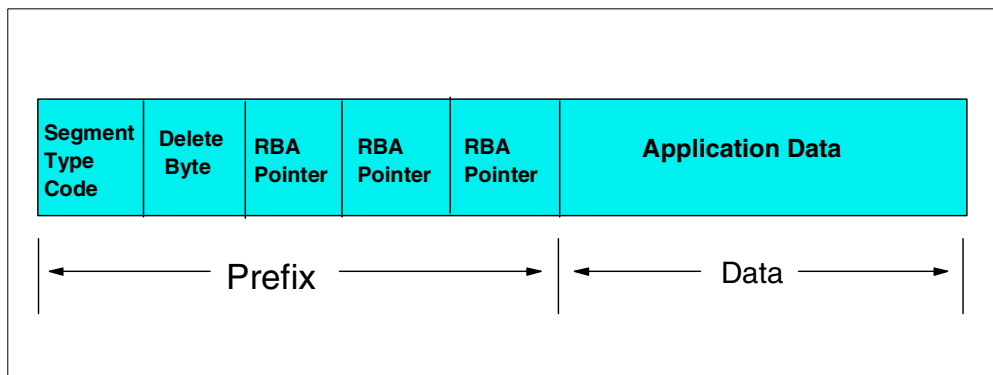


Figure 26. Segment Layout

These pointers consist of the relative offset (number of bytes) of the segment being pointed at, from the start of the data set being used to store the data. This is commonly referred to as the relative byte address (RBA). For example, a root segment would contain pointer fields in the prefix for, at a minimum all of the dependent segment types under the root. IMS will automatically define the minimum set of pointers to maintain the hierarchical structure. The database designer has the option to specify additional pre-defined types of pointers above those necessary for the minimum hierarchical structure. This pointer selection can influence the performance of applications using the databases. Figure 27 contains a diagram of database segments with their pointers.

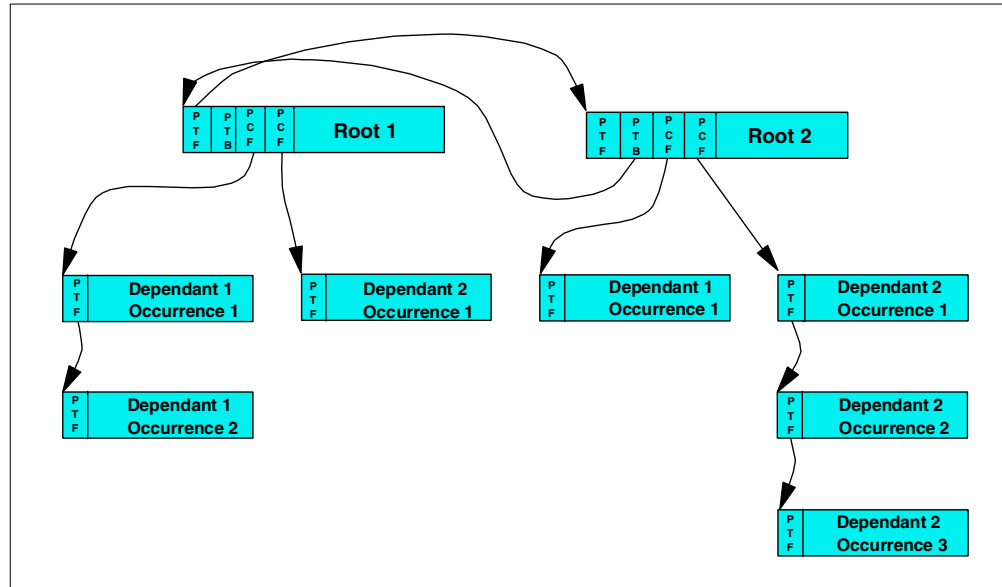


Figure 27. Database record and pointers

12.2 Physical storage of the data

There are a number of different IMS access methods used to organize and store the data segments and records. The choice of which access method to use should be made after a careful analysis of the access requirements of the applications. The choice of access method will affect the functionality available to the application, the order in which segments are returned to the application, and will influence database performance. Table 5 contains a list of the most commonly used database organizations.

Table 5. Database organization types

Organization	Database Type
Hierarchical Direct Access Method	HDAM
Hierarchical Index Direct Access Method	HIDAM
Simple Hierarchical Index Sequential Access Method	SHISAM
Hierarchical Index Sequential Access Method	HISAM
Generalized Sequential Access Method	GSAM
Data Entry Database	DEDB

The operating system access methods that underlays the IMS access methods are mentioned in this section, but are discussed in more detail in the following section.

The three major IMS access methods are:

- Hierarchical Direct — Consisting of the Hierarchical Direct Access Method (HDAM) and the Hierarchical Indexed Direct Access Method (HIDAM). Both of these methods are described in detail below.

- **Hierarchical Sequential** — Consisting of the Hierarchical Sequential Access Method (HSAM) and the Hierarchical Indexed Sequential Access Method (HISAM). These are less used today, as the HD access methods have a number of advantages. A short description of them, together with their limitations, is given below.
- **Data Entry DataBase (DEDB)** — This was originally part of the separately orderable IMS Fast Path feature, but is now delivered as part of the IMS base product. It has characteristics that make it suitable for high performance and/or high availability applications. However, the application must be specifically designed and written to make use of these characteristics. It is described in detail below.

The Hierarchical Direct (HD) and Hierarchical Sequential (HS) databases are often referred to as *Full Function* (FF) databases, while the DEDB databases are referred to as *Fast Path*. Because of its original development as a separately orderable feature, *Fast Path* functions are normally described in separate sections/chapters in the manuals. There is a second Fast Path database access method, Main Storage Database (MSDB), but its functionality has been superseded by the Virtual Storage Option (VSO) of the DEDB, so it is not described in this book, and you are advised not to use it.

In addition, there are two more IMS access methods that provide additional functionality:

- **Index Databases** — These are used to physically implement secondary indices and HIDAM primary indices.
- **Generalized Sequential Access Method (GSAM)** — This is used to extend the restart/recovery facilities of IMS Database Manager to non-IMS sequential files being processed by IMS batch programs and BMPs. These files can also be accessed directly via MVS access methods. Table 6 contains a list of database organizations and which application regions can access each type.

Table 6. IMS access method availability by application address space type

IMS Access Method	MPP	IFP	BMP	Batch	CICS
HDAM	Y		Y	Y	Y
HIDAM	Y		Y	Y	Y
HSAM	Y		Y	Y	Y
HISAM	Y		Y	Y	Y
DEDB	Y		Y	N	Y
GSAM	N		Y	Y	N
Secondary Index	Y		Y	Y	Y

12.2.1 HDAM

Refer to Figure 28 for the following discussion. An HDAM database normally consists of one VSAM ESDS or OSAM data set. To access the data in an HDAM database, DL/I uses a randomizing module. The randomizing module is used by DL/I to compute the address for the root segment in the database. This address consists of the relative number of a VSAM Control Interval (CI) or OSAM block within the data set and the number of an anchor point within that block. Anchor point (s) are located at the beginning of the CI/blocks. They are used for the chaining of root segments which randomize to that CI/block. All chaining of segments is done using a 4 byte address, this address is the byte the segment starts at, relative to the start of the data set (Relative Byte Address/RBA).

A general randomizing module, DFSHDC40, is supplied with IMS. This is suitable for most applications. The *IMS/ESA Customization Guide*, SC26-8020 describes this module. It also gives details if you wish to modify this module, or develop your own randomizing routines. The VSAM ESDS or OSAM data set is divided into two areas:

- The root addressable area. This is the first n control intervals/blocks in the data set. You define it in your DBD.
- The overflow area is the remaining portion of the data set. The overflow area is not explicitly defined, but is the remaining space in the data set after space is allocated for the root addressable area.

The root addressable area (RAA) is used as the primary storage area for segments in each database record. IMS will always attempt to put new/updated segments in the RAA. The overflow area is used when IMS is unable to find suitable space for a segment being inserted in the RAA.

IMS employs a number of techniques to distribute free space within the RAA, to allow future segments to be inserted in the most desirable block. Since database records will vary in length a parameter (in the DBD) is used to control the amount of space used for each database record in the root addressable area (note that this limitation only applies if the segments in the record are inserted at the same time, see below). This parameter, “bytes” in the RMNAME= keyword, limits the number of segments of a database record that can be consecutively inserted into the root addressable area. When consecutively inserting a root and its dependents, each segment is stored in the root addressable area until the next segment to be stored will cause the total space used to exceed the specified number of bytes.

The total space used for a segment is the combined lengths of the prefix and data portions of the segment. When exceeded, that segment and all remaining segments in the database record are stored in the overflow area. It should be noted that the “bytes” value only controls segments consecutively inserted in one database record. Consecutive inserts are inserts to one database record without an intervening call to process a segment in a different database record.

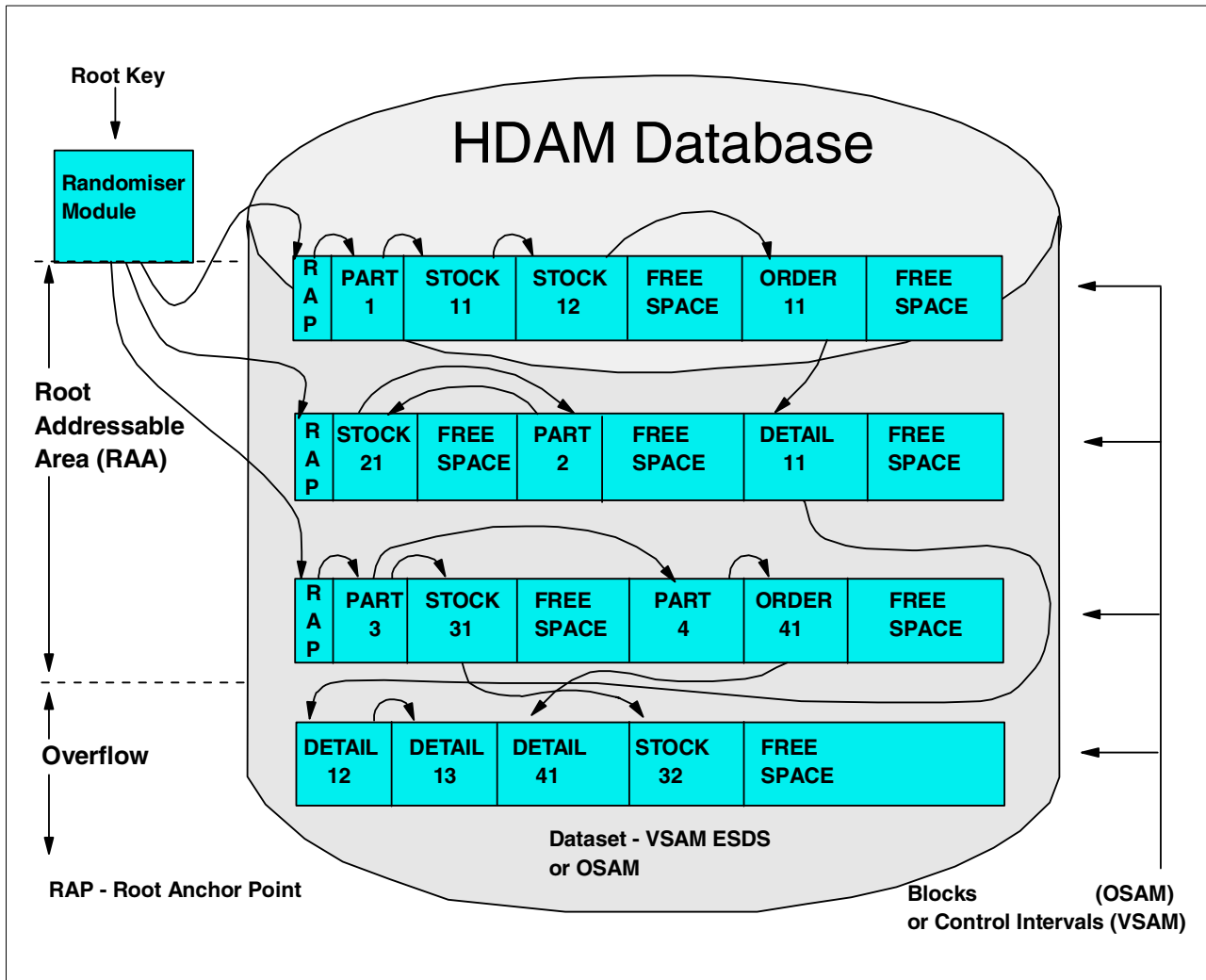


Figure 28. HDAM — database in physical storage

When you initially load HDAM databases, you can specify that a percentage of the space in each block should be left for subsequent segments to be inserted. This freespace will allow subsequent segments to be inserted close to the database record they belong to. This freespace percentage is specified on the DBD. You can also specify in the DBD that a percentage of blocks in the data set are left empty, but you should not do this with HDAM databases, as this will result in IMS randomizing segments to a free block, then placing them in another block. This would result in additional I/O (the block they randomize to, plus the block they are in) each time the segment is retrieved. You should analyze the potential growth of the database to enable you to arrive at a figure for this free space.

When IMS is inserting segments, it uses the HD space search algorithm to determine which CI/block to put the segment in. This attempts to minimize physical I/Os while accessing segments in a database record by placing the segment in a CI/block as physically close as possible to other segments in the database record. The HD space search algorithm is described in the chapter on designing Full Function databases, in *IMS/ESA Administration Guide: Database Manager*, SC26-8012.

In addition to organizing the application data segments in an HDAM database, IMS also manages the freespace in the data set. As segments are inserted and deleted, areas in the CI/blocks become free (in addition to the freespace defined when the database is initially loaded). IMS space management allows this free space to be re-used for subsequent segment insertion. To enable IMS to quickly determine which CI/blocks have space available, IMS maintains a table (bit map) that indicates which CI/blocks have a large enough area of contiguous free space to contain the largest segment type in the database. Note that if a database has segment types with widely varying segment sizes, even if the CI/block has room for the smaller segment types, it would be marked as having no free space if it cannot contain the largest segment type. The bit map consists of one bit for each CI/block, set on (1) if space is available in the CI/block, set off (0) if space is not available in the CI/block. The bit map is in the first (OSAM) or second (VSAM) CI/block of the data set and occupies the whole of that CI/block. Figure 29 illustrates the free space management.

Within the CI/block itself, IMS maintains a chain of pointers to the areas of freespace. These are anchored off a Free Space Element Anchor Point (FSEAP). This contains the offset, in bytes from the start of the CI/Block, to the first Free Space Element (FSE), if freespace exists. Each area of freespace greater than 8 bytes contains a FSE containing the length of the freespace, together with the offset from start of CI/block to the next FSE.

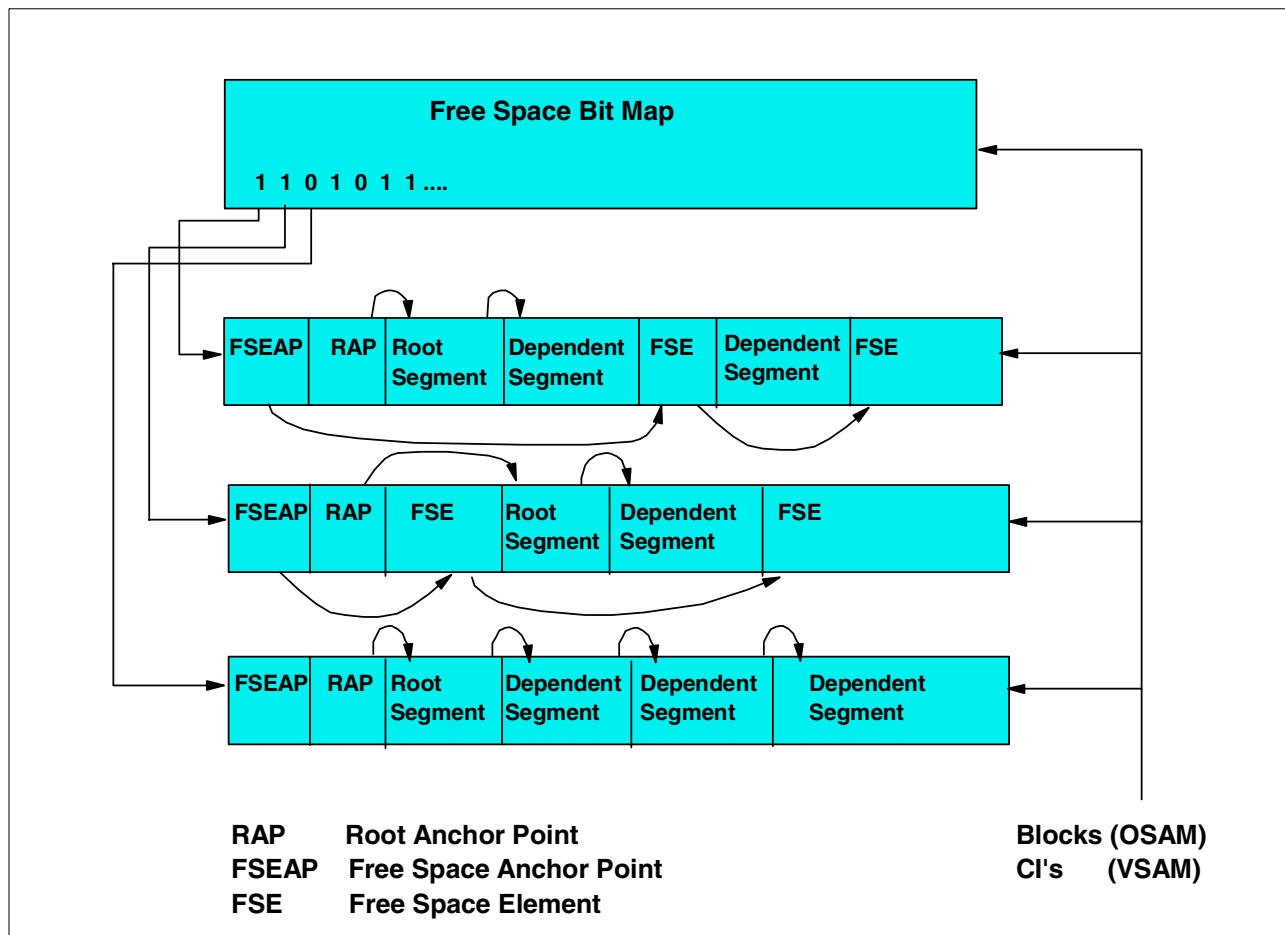


Figure 29. HDAM database — free space management

All management of free space and application segments in the data sets is done automatically by IMS and is transparent to the application. You only need to be aware of these because of the performance and space usage implications.

A full description of the HDAM internal organization is given in the chapter on Choosing a Database Type in *IMS/ESA Administration Guide: Database Manager*, SC26-8012.

The principle features of the HDAM access method are:

- Fast random access to the root segments, via the randomizer.
- Quick access to segments in a database record, as IMS attempts to store them in the same, or physically near, CI/block
- Automatic re-use of space after segment deletions
- Can have non-unique root segment keys

The principle weaknesses of the HDAM access method are:

- It is not possible to access the root segments sequentially, unless you create a randomizing module that randomizes into key sequence, or incur the overhead of creating and maintaining a secondary index.
- It is slower to load than HIDAM, unless you sort the segments into randomizer sequence (for example by writing user exits for the sort utility that call the randomizing module).
- It is possible to get poor performance if too many keys randomize to the same anchor point.

Overall, if there are no requirements to regularly process the database in root segment key sequence, and you do not require the special features of a DEDB, choose HDAM.

Further details on the reasons for choosing the HDAM access method, and choosing the options you can define for it, are given in the section on Database design.

12.2.2 HIDAM

A HIDAM database in DASD is actually comprised of two physical databases that are normally referred to collectively as a HIDAM database, see Figure 30. When defining each through the DBD, one is defined as the HIDAM primary index database and the other is defined as the main HIDAM database. In the following discussion the term “HIDAM database” refers to the main HIDAM database defined through DBD.

The main HIDAM database is similar to an HDAM database. The main difference is in the way root segments are accessed. In HIDAM, there is no randomizing module, and normally no RAPs. Instead, the HIDAM primary index database takes the place of the randomizer in providing access to the root segments. The HIDAM primary index is an indexed sequential file (VSAM KSDS) that contains one record for each root segment, keyed on the root key. This record also contains the pointer (RBA) to the root segment in the main HIDAM database.

The HIDAM primary index database is used to locate the database records stored in a HIDAM database. When a HIDAM database is defined through the DBD, a unique sequence field must be defined for the root segment type. The value of this sequence field is used by DL/I to create an index segment for each root segment (record in the KSDS). This segment in the HIDAM primary index database contains, in its prefix, a pointer to the root segment in the main HIDAM database.

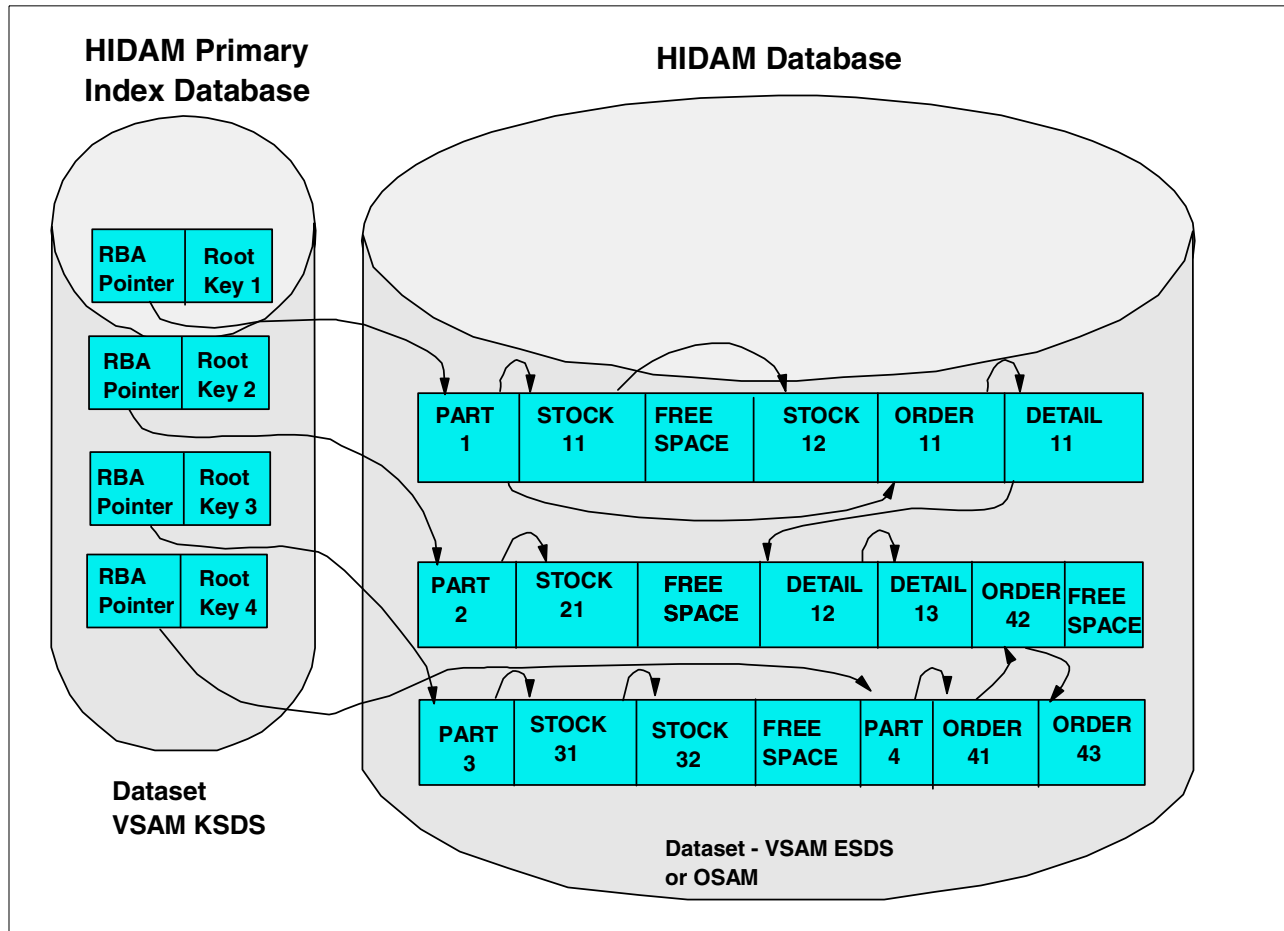


Figure 30. HIDAM database in physical storage

When the HIDAM database is initially loaded, the database records are loaded into the data set in root key sequence. Providing root anchor points are not specified, reading the database in root key sequence will also read through the database in the physical sequence the records are stored in on the DASD. If you are processing the databases in key sequence like this, and regularly inserting segments and new database records, you should specify sufficient freespace when the database is initially loaded so that the new segments/records can be physically inserted adjacent to other records in the key sequence.

A full description of the HIDAM internal organization is given in the chapter on “Choosing a Database Type” in *IMS/ESA Administration Guide: Database Manager*, SC26-8012.

Free space in the main HIDAM database is managed in the same way as in an HDAM database. IMS keeps track of the free space using Free space elements anchor points. When segments are inserted, the HD free space search algorithm is used to locate space for the segment. The HIDAM primary index database is processed as a normal VSAM KSDS, and, consequently, a high level of insert/delete activity will result in CI/CS splits, which may require the index to be reorganized.

When the HIDAM database is initially loaded, free space can be specified as a percentage of the CI/blocks to be left free, and as a percentage of each CI/block to be left free. This is specified on the DBD.

The principle advantages of the HIDAM access method are:

- Ability to process the root segments and database records in root key sequence.
- Quick access to segments in a database record, as IMS attempts to store them in the same, or physically near, CI/block.
- Automatic re-use of space after segment deletions.
- Ability to reorganize the HIDAM primary index database in isolation from the main HIDAM database (but NOT the other way round).

The principle weaknesses of the HIDAM access method are:

- Longer access path, compared to HDAM, when reading root segments randomly by key. There will be at least one additional I/O to get the HIDAM primary index record, before reading the block containing the root segment (excluding any buffering considerations).
- Extra DASD space for the HIDAM primary index.
- If there is frequent segment insert/delete activity, the HIDAM primary database will require periodic reorganization to get all database records back to their root key sequence in physical storage.

Overall, only choose HIDAM if there are requirements to regularly process the database in root segment key sequence. If there are also requirements for fast random access to roots (from online systems), look at alternatives for the sequential access, such as unload/sort or secondary indices.

12.2.3 Index databases

Index databases are used to implement secondary indices, and the primary index of a HIDAM database. The index database is always associated with another HD database. It cannot have an existence by itself. It can, however, be processed separately by an application program.

The Index database consists of a single VSAM KSDS (Key Sequenced Data Set). Unlike the VSAM ESDSs used by IMS, which are processed at block (Control Interval) level, the index database is processed as a normal indexed file. IMS uses the normal VSAM access method macros to access it.

The records in the KSDS contain the fields that make up the key, and a pointer to the target segment. For a secondary index, the pointer can be direct (relative byte address of the target segment) or symbolic (the complete, concatenated key of the target segment). For a HIDAM primary index, it is always direct.

As the indices are a normal VSAM KSDS (and no relative address are used for data in the index database) they can be processed using the normal VSAM Access Method Services (IDCAMS). For example you could use the REPRO function to copy the database and remove CI/CA splits or resize the data set, without having to perform any other IMS reorganization.

12.2.4 DEDB

Data Entry Databases (DEDBs) were originally part of the separately priced IMS Fast Path functions. These functions are now delivered as part of the IMS base product, though some of the documentation may still reflect this separation. The Fast Path functions were originally developed for application that required higher performance, capacity and/or reliability than was available with the normal IMS functions. There is another database access method that was available with Fast Path, Main Storage Databases (MSDB), but as this is now been superseded by DEDBs using the virtual storage option described below, we will not describe it further, and recommend that you do not use it. DEDBs are frequently referred to in the documentation as Fast Path databases, as distinct from the HD and HS databases, which are referred to as Full Function databases (the main restrictions on Fast Path DEDBs are described below).

The DEDB's implementation of the IMS hierarchical database model is broadly the same as the IMS HDAM access method. However:

- The implementation of the IMS access method onto the operating system access method data sets is different (and appreciably more complicated) than with HDAM. This is done to provide the additional features offered by DEDBs.
- There are various restrictions on facilities available with this access method, again a trade-off for the additional features provided.

The hierarchical structure of a database record within a DEDB is the same as HDAM, except for an additional dependent segment type. There is one root segment in each database record and from zero to 126 dependent segment types. One of these segment types can, optionally, be a sequential dependent segment type (See below for more details). As with HDAM, a randomizing module is used to provide initial access to the database data set(s) containing the DEDB database.

The highest level in the structure used to implement a DEDB is the area. A DEDB can consist of from 1 to 240 areas. Each area is implemented as one VSAM ESDS data set.

Each DEDB Area data set is divided into:

- A root addressable part — This contains VSAM CIs that are addressable by the randomizing middle.
- An independent overflow part.
- A sequential dependent part — This is optional, and is only defined if the DEDB has a sequential dependent segment defined in the hierarchical structure.

The root addressable part is further subdivided into units of work (UOWs). These should not be confused with the unit of work that encompasses an application's minimum set of updates to maintain application consistency. The DEDB UOW is similar, however, as it is the smallest unit that some Fast Path utilities (for example, reorganization) work with, and lock, preventing other transactions accessing them. Each unit of work consists of from 2 to 32767 CIs, divided into a base section of 1 or more CIs and a dependent overflow section, consisting of the remaining CIs.

Figure 31 shows segments stored in a DEDB area data set.

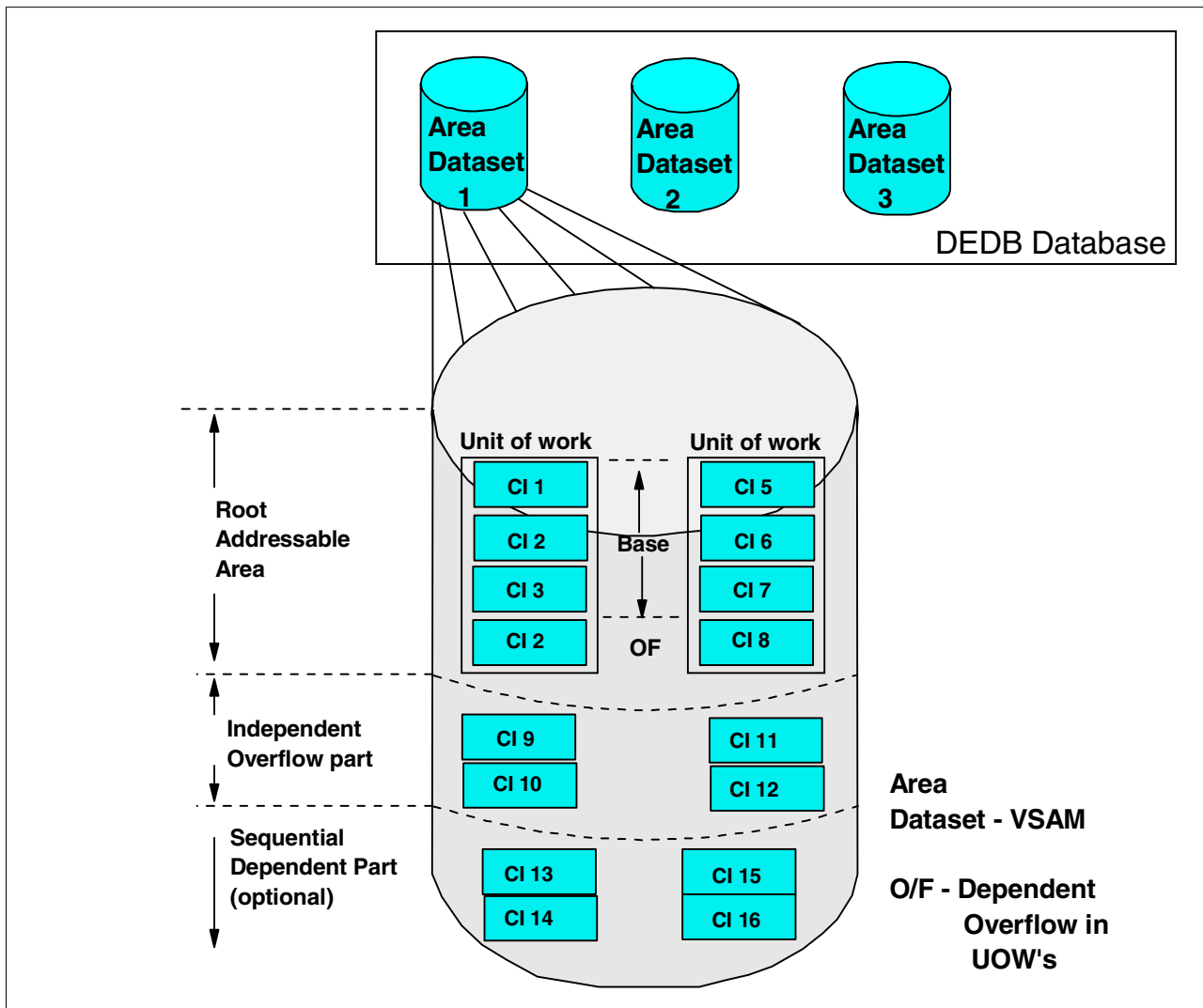


Figure 31. Overall structure of Fast Path DEDB

The randomizing module works in a similar way to an HDAM database. It takes the key value of the root segment and performs calculations on it to arrive at a value for a root anchor point. However, for a DEDB this is the root anchor point within the Area data set. The randomizer must also provide the value of the area data set that contains the RAP. Again, there is a sample randomizer provided with IMS, although due to DEDB's unique characteristics, you should look closely at whether you need to code your own.

The randomizer will produce the value of a root anchor point in the base section of a unit of work. IMS will attempt to store all dependent segments (except sequential dependents) of the root in the same UOW as the root. If more than one root randomizes to the same RAP, then they are chained off the UOW in key sequence. If there is insufficient space in the base section, then root and non-sequential dependent segments are placed in the overflow section of that UOW. If there is no space in the dependent overflow section in the UOW, a CI in the independent overflow part of the DEDB Area is allocated to that UOW and the segment stored there. This CI in the independent overflow part is then used exclusively by that UOW, and is processed with that UOW by the DEDB reorganization utility.

The free space between the data segments in the CIs in the root addressable part and Independent overflow part of a DEDB area data set are managed in the same way as in an HDAM data set, with a free space element anchor point at the start of the CI pointing to a chain of free space elements. As with HDAM, space from deleted segments is automatically re-used, and the UOW can be reorganized to consolidate fragmented free space (without making the database unavailable). Unlike an HDAM database, there is no free space map. The segments for a database record can only be allocated in the same UOW (or attached segments in dependent overflow) as the root segment. An out of space condition results if insufficient free space is available in the UOW or Independent overflow and the database is then unavailable while lengthy recovery action is taken.

The following, optional, features can also be used with a DEDB:

- **Virtual Storage Option (VSO)** — This stores the CIs of a DEDB in OS/390 data spaces, eliminating I/O to the DASD system. The data can either be loaded (partially or completely) when the database is opened, or loaded into the dataspace as it is referenced.
- **Multiple Area Data Sets** — You can define DEDB areas so that IMS will automatically maintain up to seven copies of each area. This can be used to provide a backup if I/O errors occur, allow data sets to be re-defined on a different device without taking the database offline, or to provide parallelism in I/O access for very busy applications.
- **High Speed Sequential Processing** — This provides a facility is a function provided by Fast Path to enhance the performance of programs that are processing segments sequentially in a database. IMS issues a single I/O request that reads 10 CIs at a time, to reduce the overhead of multiple I/O request, and stores the CIs in a separate buffer pool. It also issues the read request in advance of the program asking for the data, to provide parallel processing. In this way, the segments in the database are available to the program without any delays to wait for I/O processing. The overall runtime can be significantly reduced, as long as the database is being read sequentially.
- **Sequential Dependent Segments** — A DEDB database can have one sequential dependent segment type defined in the database record. This is processed completely separately to the other dependent segments. Normal application programs can only Insert new sequential dependent segments or read existing sequential dependent segments. All other processing of these sequential dependents is performed by IBM supplied utility programs. The sequential dependents are stored in the Sequential dependent part of the area data set in chronological sequence, and processed by the IMS utilities, to read or delete them, in the same sequence.

The main situations where you might consider using Fast Path DEDBs are:

1. Where you have very high volumes of data to store. The DEDB can be spread over up to 240 VSAM ESDS data sets, each with a maximum capacity of 4 GB. However not all this space is available for application data as some space is needed for IMS and VSAM overhead and free space.
2. Where you have a small to medium database that needs extremely fast access. you could use the DEDB VSO option and have the data held in an OS/390 dataspace, making a major reduction in the physical I/O associated with the database.
3. If you needed a database with very high availability. The use of multiple area data sets, the ability to reorganize online and the DEDBs tolerance to I/O errors mean the database can be kept available for extended periods.
4. Where an application needs to record large amounts of data very quickly (for example journalling details of online financial transactions) but does not require to update this data, except at specified times (for example, an overnight process), then a DEDB with a sequential dependent segment could provide the solution.

The principal disadvantages that have to be weighed against OSAM DEDBs to see if they are a cost effective solution are:

- This method is more complicated than other IMS access methods. Consequently, it requires a higher degree of support both for initial setup and running.
- The person designing the application must understand the restrictions and special features of DEDBs and design the application accordingly.
- The DEDBs are only available for applications running against an IMS control region (MPP, IFP, BMP and CICS applications). There is no batch support except indirectly via the IMS supplied utilities to extract the data.
- Fast Path DEDBs do not support logical relationships or secondary indices, so these functions must be implemented in the application.

For more details on using DEDBs, together with samples of their use, refer to the ITSO publication *IMS Fast Path Solutions Guide*, SG24-4301

The features of DEDBs are described in detail in chapters on designing a Fast Path database in *IMS/ESA Administration Guide: Database Manager*, SC26-8012.

The utilities used with DEDB are described in *IMS/ESA Utilities Reference: Database Manager*, SC26-8034 and the randomizer and other Fast Path exits are in *IMS/ESA Customization Guide*, SC26-8020.

12.2.5 GSAM

An OS/390 sequential file being used as an interface to or from an IMS application can be defined to DL/I as a GSAM database. However, the normal concepts of hierarchical structures do not apply to GSAM, as it just contains the normal data records, with no IMS information.

These files can be OS/390 Sequential files, or VSAM ESDSs. Before or after the IMS application processes them, other applications can process them using the normal BSAM, QSAM and VSAM access methods.

When using GSAM for sequential input and output files, DL/I will control the physical access and position of those files. This is necessary for the repositioning of such files in case of program restart. When using GSAM, DL/I will, at restart time, reposition the GSAM files in synchronization with the database contents in your application program's working storage. To control this, the application program should use the restart (XRST) and checkpoint (CHKP) calls. These calls will be discussed in 19.8, "Batch checkpoint/restart" on page 200. Note that IMS can not re-position VSAM ESDS files on restart. There are also some other restrictions on restarting, detailed in the Designing Full Function Databases chapter of *IMS/ESA Administration Guide: Database Manager*, SC26-8012.

Whenever you want your program to be restartable, you should use GSAM for its sequential input and output files. There are two reasons why you should want to do this. The first is to save time if a program rerun is required in case of program system failure. This is normally only done for long-running update programs (one or more hours). The other reason stems from a planned online usage of the databases.

12.2.6 Sequential

The two hierarchical Sequential (HS) access methods, HSAM and HISAM have now been superseded by the HD access methods. The HD access methods have a number of features that would almost always make them a better choice.

The HSAM access method will not allow updates to a database after it was initially loaded and the database can only be read sequentially. I was used in the past to process operating system sequential files, but GSAM is now a better choice.

The HISAM access method offers similar functionality to HIDAM, but has poorer internal space management than the HD access methods that would normally result in more I/O to retrieve data, and the need to reorganize the databases much more frequently.

The HS access methods are described in *IMS/ESA Administration Guide: Database Manager*, SC26-8012

12.3 Selecting database organization

Access methods can, in general, be changed during reorganization without affecting application programs. Still, because the access method is one of the most critical performance factors, it should be carefully selected. First we will discuss selection of the DL/I access method, HDAM, HIDAM, or HISAM.

12.3.1 When to choose HISAM

HISAM is not a very efficient database organization. All HISAM databases can easily be converted to HIDAM. The application should receive significant performance improvements as a result. The only situation where HISAM may be desirable over a HIDAM database is when it is a root-segment-only database.

Even so, segments are not deleted and free space reclaimed after a segment is deleted until the next database reorganization.

12.3.2 When to choose HDAM

HDAM is recognized, in practice, to be the most efficient storage organization of the DL/I. It should be considered first. After looking at all the required access to the database, if there are not requirements to process a large section of the database in key sequence, then HDAM should be chosen. If sequential access of the root keys is required, the process can retrieve the data in physical sequence and sort the output.

HDAM's prime advantages are:

1. Fast direct access (no index accesses) with few I/O operations
2. Single data associated control blocks
3. Small working set in main storage for DL/I
4. Good physical sequential access

Some disadvantages of HDAM are:

1. You need a randomizing module.
2. Sequential access in root key order is not possible if the physical sequence of database records in storage is not the same as the root key sequence. This is dependent on the randomizing module and root key characteristics.
3. If the database exceeds the space in the RAA (root addressable area) it will extend into overflow. Once in overflow, the performance of the access to these segments can increase drastically.
4. To increase the space of the database, a DBDGEN is required to increase the number of blocks in the RAA. This will also require an ACBGEN to rebuild the online ACBs for use in the online system. This will require more time to complete and to coordinate the change.

In many cases, the disadvantages for HDAM do not apply or can be circumvented. The effort needed to circumvent should be weighed against the savings in terms of main storage and CPU usage. There is no doubt, however, that an application with only HDAM databases is the most compact one. Some possible solutions for the above HDAM disadvantages are:

1. The IMS provides a general randomizing module, DFSHDC40, which can be used for any key range.
2. If heavy sequential processing is required and a randomizing module which maintains key sequence cannot be designed, then sort techniques can be used:
 - If the program is non-input driven, as is the case with many report programs, simple get-next processing presents all the database records in physical sequential order. The output could then be sorted in the desired order. Also, in many instances, only certain selected segments are required. So the output file of the extract can be a fairly small file.

- If there are input transactions which would normally be sorted in root key sequence. This can be readily done with an E61 sort exit routine which passes each root key to the randomizing module for address calculation and then sorts on the generated addresses plus the root key instead of the root key itself.
3. A secondary index could be built with the root key as index search argument. The cost of this should be weighed against the cost of sorting as in 2 above. The secondary index provides full generic key search capability, however. A secondary index on the root segment should never be used to process the whole database, as this will cost a lot more I/Os than to process the database in physical sequence.

12.3.3 When to choose HIDAM

HIDAM is the most common type of database organization. It has the advantages of space usage like HDAM but also keeps the root keys available in sequence. These days, with the speed of DASD the extra read of the primary index database can be incurred without much overhead. The most effective way to do this is to specify specific buffer pools for use by the primary index database, thus reducing the actual IO to use the index pointer segments.

HIDAM does not need to be monitored as closely as HDAM.

12.4 Physical segment design

In the final steps of segment design, we must look at the physical parameters more closely:

- The segment length:

IMS will use the segment length as defined in the DBD to store each segment. If you have left free space at the end of the segment for future use, that space will be physically hold space on DASD unless you have compressed the segment. If the application is likely to have additional requirements later, it can be easier to make use of this free space than to increase the segment length later. You have to balance the cost of making the change to the databases and programs against the cost of wasted DASD space.
- The number of occurrences per segment per parent:

Try to avoid long twin chains, that is, many occurrences of a particular segment type under one parent. Chain lengths should be estimated in terms of blocks needed to store each such segment.
- Location of segments in the hierarchy:

Try to locate the segments most often used together with the root segment into one control interval/block. The segments are initially physically stored in hierarchical sequence, so the most frequently used segments should be on the left of the structure (low segment codes).
- Average database record size:

The average database record is calculated by the total bytes of all segments under the root segment. Small segments with more twins than larger segments with fewer twins, although having almost the same number of bytes, results in better performance and space usage.

12.5 Operating system access methods

To underpin the IMS access methods, IMS uses two operating system access methods to store the data on disk storage, and move the data between the disk storage and the buffers in the IMS address space. These are:

- Virtual Sequential Access Method (VSAM). Two of the available VSAM access methods are used, Key Sequenced Data Sets (KSDS) for Index databases, and Entry Sequenced Data Sets (ESDS) form the primary data sets for HDAM, HIDAM, etc. The data sets are defined using the VSAM Access Method Services (AMS) utility program.
- Overflow Sequential Access Method (OSAM) — This access method is unique to IMS and is delivered as part of the IMS product. It consists of a series of channel programs that IMS executes to use the standard operating system channel I/O interface. The data sets are defined using JCL statements. As far as the operating system is concerned, an OSAM data set is described as a physical sequential data set (DSORG=PS)

There are two types of data sets defined using these access methods:

- Indexed sequential data sets. These are all defined and accessed as VSAM KSDSs, and are used to implement primary and secondary index databases. These databases are processed using the standard record level instructions of VSAM. A catalogue listing (VSAM LISTCAT) will show all current details of the files. They are susceptible to the normal performance degradation of VSAM KSDSs from CI/CS splits caused by insert/delete activity. They can, if necessary, be processed using AMS utilities such as REPRO.
- Sequential data sets. These are defined and accessed either as VSAM ESDSs or using OSAM. It is important to note that, while these data sets appear as sequential data sets to the operating system, IMS accesses them randomly. The data sets do not contain records as such. IMS always processes them at the CI (VSAM) or block (OSAM) level. The internal structure within each CI/block is arranged as described in the previous section on IMS access methods. Interpreting catalogue listings of these files as if they were sequential files can, at times, be misleading.

12.5.1 VSAM or OSAM

While most physical databases are implemented over a single VSAM ESDS or OSAM data set, IMS provides facilities to spread an HDAM or HIDAM physical database over up to nine additional data sets (multiple data set groups). The facility is restricted as, with the current release of IMS, the 1st, primary data set group, that is always defined, must contain the root segments, and can contain any dependent segment type. The other (secondary) data set groups can each contain any dependent (non-root) segment type. However, each dependent segment type can only be defined in one data set group. This is, aside from performance implications, transparent to applications. If the database needs to be reorganized, then all data sets that make up the physical database have to be reorganized at the same time.

Reasons why you may wish to use secondary data set groups are:

- To separate little used segments from the main data set, to leave more space for frequently used segments. This will increase the chance the all regularly accessed segments are in the same block with the root, and enhance performance. For example, you might have a segment type that has a new occurrence inserted each month, say month end audit totals. This is only rarely accessed after insertion. Placing in this segment type in a secondary data set group, while imposing an overhead on the program that inserted it, could improve performance of all other programs as there is an increased chance segments they access are in the same block as the root, and more database records can be packed into one CI/block.
- If you have a database with one very large segment type, and a number of other small segment types than, as described above, this can result in unusable space as IMS space management only regards a CI/block within a data set as having freespace if it can accommodate the largest segment type stored in that data set. Putting this large segment type in a secondary data set group means that the other data set groups will now only be regarded as full if they could not contain the second largest segment type.
- You can specify different freespace parameters on the different data set groups, so you could place non-volatile segment types in a data set group with little free space, to increase packing in a CI/block, and consequently the chances of having several segments a program is retrieving in the same block. Volatile segment types (that is, frequent insert/delete) could be placed in a data set group with a large freespace specification, allowing segments to be inserted near related segments.
- For very large databases, you may be approaching the structural limit of the data set access method (4 GB of data). If you have one or two segment types that occur very frequently, the each of these segment types could be placed in one or more secondary data set groups to provide more space. But in this case, see also the additional features of OSAM below, and also look closely at DEDBs, which can be spread over many more data sets.

When performing space calculations, you need to be aware that, in addition to the overhead for IMS control information (pointers, etc.), VSAM data sets will also contain a suffix area at the end of the CI that contains VSAM control information. This makes the space available in the CI for IMS data slightly less than the VSAM CI size.

The choice between OSAM and VSAM ESDS for the primary database data sets will depend, to some extent, on whether your site already uses VSAM and whether you need to make use of the additional features described below. The choice between VSAM ESDS and OSAM is not final, as a database can be changed from one access method to the other by unloading the database, changing and regenerating the DBD, then re-loading the database.

As the OSAM access method is specific to IMS, it has been optimized for use by IMS. Reasons you may want to use OSAM are:

- Sequential Buffering (SB) — With this feature, IMS will detect when an application is processing data sequentially and pre-fetch blocks it expects the application to request from DASD, so they will already be in the buffers in the IMS address space when the application requests segments in the block. This is manually activated for specific IMS databases/programs. It can appreciably decrease run times for applications processing databases sequentially. It is similar to the sequential prefetch available with some DASD controllers, but has the advantage that the data is fetched into the address space buffer in main memory, rather than the DASD controller cache at the other end of the channel. Refer to the “Full Function DB Design Considerations” chapter in *IMS/ESA Administration Guide: Database Manager*, SC26-8012 for details.
- The structural limit on the amount of data that IMS can store in a VSAM ESDS is 4 GB of data. This also used to be the limit on OSAM. However, changes have been made to OSAM to allow it to process a data set up to 8 GB in size. This is part of the base IMS product, however it was retrofitted to IMS version 5 as maintenance. As this feature was retrofitted, it is not mentioned in the IMS Version 5 manuals. Providing the maintenance has been applied to the system, this feature can be used by simplifying defining the data set between 4 GB and 8 GB in size and, for HDAM adjusting the number of RAA to make use of the increased space. If you have written your own HDAM randomizer module(s) you should also review these.
- Overall, OSAM is regarded as more efficient as it is more efficient, buffering, shorter instruction path

12.5.2 IMS and system managed storage (SMS)

Most of the IMS data sets can be managed by SMS. The only concern would be OLDS data sets. If they should get migrated (not very likely in most installations) they may be recalled with different attributes.

OLDS data sets must be allocated in contiguous space. It could also be possible for both the primary and secondary OLDS data sets to be on the same volume. This is a major problem if that volume becomes unreadable. You should use management classes to avoid this.

WADS data sets have very high write rate and are very sensitive to slow response. These data sets should be placed with some care. SMS may not provide a good place to allocate them.

If any OLDS, RLDS or SLDS or image copy data sets are SMS managed, the CATDS parameter must be set for the RECON. This will tell DBRC to use the system catalog to find data sets and not be concerned if they are not on the same volumes which they were originally allocated.

12.6 IMS checkpoints: preserving application data integrity

A database management system, such as IMS, provides facilities to keep all the application data stored in the databases in a consistent state. This discussion is principally concerned with keeping the application data consistent, from an applications point of view. It relies on the application using the facilities provided by IMS. However, the facilities to consistently update the database also ensure that all internal IMS information (pointers, free space elements, etc.) are kept consistent, though this is transparent to the application program.

An application program may make updates to several IMS databases. If a problem is encountered part of the way through these updates, either the program fails, or application logic dictates it cannot continue with the processing, then it will need to restore the data in the databases to the state when it started updating them. For example, a program adds a detail to the order, in the order database, and then needs to update the parts database to reduce the quantity of the part available for ordering. If the program updates the order database, but then fails before updating the parts database, the order is recorded, but the quantity of the part is still shown as available for ordering on the parts database. The update to the order database and the update to the parts database make up a single *unit of work* (UOW). For the application data to be consistent, either all the updates in a unit of work must be written to the database successfully (committed) or none of the updates in the UOW must be committed.

To maintain database consistency, IMS uses the concept of the *application checkpoint*. You should not confuse the application checkpoint, which applies to the single execution of an application program, with the system checkpoints IMS subsystems take. System checkpoints are taken to allow the IMS subsystem to recover from a failure of the complete IMS subsystem. The application checkpoint indicates to IMS the end of the applications unit of work and causes IMS to commit all updates made in that UOW.

An applications UOW commences when the application program starts running. By default, IMS takes an application checkpoint, and commits all updates when the application terminates normally. You can also explicitly request a checkpoint, using the CHKP function of the DL/I API. The CHKP call is also taken as starting another UOW. If an application program terminates abnormally, then all database changes are backed out to the last commit point (start of program or last CHKP call). The application can also explicitly back out all updates within the current UOW by using the ROLB, ROLL or ROLS functions of the DL/I API (the difference between the calls relate to action taken by the Transaction Manager component, if applicable, and whether the application regains control after the call). The use of these functions is described fully in the section on maintaining database integrity in *IMS/ESA Application Programming: Database Manager*, SC26-8015.

For long running batch and BMP application programs, you should issue explicit checkpoint calls at regular intervals. As the programs read database records, details of these database records (internal IMS addresses) are stored by the IMS subsystem until the application reaches a commit point (issues a CHKP or terminates). This is done to prevent other application programs updating these database records while the application is working with them. These details are stored in an internal buffer in the IMS address space. Failure to issues regular checkpoints can cause the following problems:

- The IMS address space has insufficient storage to contain all the buffers needed to contain these details, resulting in the application program being terminated.
- If the application fails, or issues a ROLL, ROLB or ROLS call, IMS will have to back out all the updates performed by the application. If it has been running for a long time without checkpointing, it may well take the same time to back out all the updates as it took to apply them. Equally, if you then correct the problem and re-start the program, it will take the same time again to re-process the updates.

- For BMPs, other applications processing the databases via the IMS control region are prevented from accessing these database records. This can cause severe response time problems if the other applications are being used by online users. For Batch jobs, you can get similar problems if block level data sharing is being used.

Long running programs should issue checkpoints based on the number of database calls made. As a rule of thumb, initially issue batch checkpoints at about every 500 database calls. You do not want to checkpoint too frequently, as there is an overhead in writing out all updates and your application re-positioning itself in all the IMS databases after the CHKP call. Obviously you cannot CHKP more frequently than the number of calls in one UOW. As you may need to tune the checkpoint frequency, it is recommended that you code the program so it can be easily changed. It is best to code it in the program as a variable, possibly input as a parameter at run time.

The functions described in the previous paragraphs are referred to as *basic checkpoint*. For applications running in Batch and BMP address spaces, there is also extended checkpoint functionality available. This is referred to as *symbolic checkpointing*. Symbolic checkpointing provides the following additional facilities that enable application programs running in batch or BMP address spaces to be re-started.

- The XRST function call is made at the start of the program, and indicates to IMS that the application is using symbolic checkpointing.
- The CHKP function is extended to allow the application to pass up to seven areas of program storage to IMS. These areas are saved by IMS and returned to the program if it is restarted. This can be used for any variables, (for example, accumulated totals, parameters) that the application would need to resume processing.
- Each CHKP call is identified by a unique ID. This is displayed in an IMS message output to the operating system log when the checkpoint is successfully complete,
- If the program fails, after correcting the problem, it can be restarted from either the last, or any previous successful checkpoint in that run. IMS will re-position databases (including non-VSAM sequential files accessed as GSAM) to the position they were at when the checkpoint was taken. When the XRST call is made on re-start, the program will receive the ID of the checkpoint it is re-starting from, together with any user areas passed to IMS when that CHKP call was issued.

Full details of symbolic checkpointing, along with various restrictions on what can be done, are in the chapter on maintaining database integrity in *IMS/ESA Application Programming: Database Manager*, SC26-8015.

12.7 Locking: sharing IMS data between multiple tasks

The other main facility a Database Management System (as distinct from the use of a database) provides, is the ability for more than one application to simultaneously access the database for update, while preserving database integrity.

This prevents situations such as in the following example: Application A reads a record. While it is processing it (waiting for a user to respond at a terminal), application B reads the same record. While application B is processing the record, application A writes back the updated record. The user of application B now responds, and application B writes back the updated record, overwriting the update to the record made by application A.

The mechanism used to prevent this is to lock (enqueue) the database segments/records until the application has finished processing them successfully, that is reached the end of a unit of work, as described above. As in the previous section, while this discussion is mainly concerned with ensuring application data is updated consistently, the mechanisms used by IMS also ensure that IMS's internal information in the databases (pointers, etc.) remains consistent.

One problem that can occur from this enqueueing of database segments, is a deadlock between two application programs. For example, application A reads database record 1. While A is doing other processing, application B reads database record 2, then tries to read database record 1, and is suspended waiting for it, as it is enqueued by application A. Application A now attempts to read database record 2, and is suspended, as it is enqueued by application B. Both applications are now suspended waiting for a record enqueued by the other — a deadlock. IMS detects this, and will abnormally terminate (abend) the application it assesses has done the least work, backing out its updates to the last commit point. The mechanism IMS uses to detect the deadlock depends on what method of data sharing is being used (see below). This is either direct detection of the deadlock from the details enqueued, or by timeout; that is, terminating a task after a (parameter specified) period of waiting for a database record.

If the application is accessing DB2 tables, DB2 also detects deadlocks by timeouts and will instruct IMS to abend the program. The abend code issued is the same as for an IMS database deadlock. What IMS cannot detect is a deadlock between two applications where the two different resources the applications are trying to get are being managed by two separate resource managers. This is most common with CICS applications using IMS/DB databases. For example, CICS task A reads, and enqueues a database record. CICS task B then issues a CICS ENQ for a resource, for example to serialize on the use of a TDQ. CICS task B then attempts to read the database record held by task A, and is suspended, waiting for it. CICS task A then attempts to serialize on the resource held by task B and is suspended. We now have a deadlock between task A and B. But neither IMS or CICS is aware of the problem, as both can only see the half of the deadlock they are managing. Unless IMS was using one of the data sharing techniques that timed out application that wait for the database, or CICS was set up to abend tasks after a very short time suspended, this deadlock would have to be resolved manually.

The person designing an application that uses IMS databases needs to be aware of the potential problems with database deadlocks, and design the application to avoid them. If the application also locks resources managed by another product, they also need to be aware of the potential for a deadlock developing between the IMS database records and the resources managed by the other product. Unfortunately, deadlocks often only occur when the application processes very large volumes, as they often require very precise timing to occur. This means that they are often not detected during testing with small volumes.

IMS supports three methods of sharing data between a number of application tasks:

- Program isolation (PI) — This can be used where all applications are accessing the IMS databases via a single IMS control region. IMS maintains tables of all database records enqueued by the tasks in buffers in the control region address space. This provides the lowest level of granularity for the locking, and the minimum chance of a deadlock occurring. Deadlocks are resolved by IMS checking the tables of database records enqueued to ensure there is not a deadlock situation, and abending one of the tasks if there is.
- Block level data sharing — This allows any IMS control region or batch address space running on an OS/390 system to share access to the same databases. It uses a separate feature, the Internal Resource Lock Manager, IRLM. This is delivered as part of the IMS product, but needs to be separately installed. It runs in its own address space in the OS/390 system and maintains tables of the locks in this address space. With block level data sharing IMS locks the databases for the application at the block level. This locking is at a higher level than with program isolation (that is, all database records in a block are locked). Because of this coarser level of locking, there is an increased risk of deadlocks and contention between tasks for database records. Deadlocks are resolved by a timeout limit specified to the IRLM. If the disk storage the databases are on is shared between two OS/390 systems, it is also possible to share the databases between IMS applications running on the two OS/390 images, by running an IRLM address space on each of the two OS/390 images. The IRLMs communicate using VTAM but maintain lock tables in each IRLM address space. IRLM is also used as the lock manager for DB2 but, because of the different tuning requirements, you should use separate IRLM address spaces for DB2 and IMS. IRLM was originally developed for IMS, before adoption for use with DB2. It was originally known as the IMS Resource Lock Manager (IRLM) and you may find it referred to by this name in older publications.
- Sysplex data sharing — Where a number of OS/390 systems are connected together in a sysplex, with databases on DASD shared by the sysplex, it is possible for IMS control regions and batch jobs to run on any of these OS/390 images and share access to the databases. To do this, an IRLM address space, running version 2 of IRLM, must be running on each OS/390 image the IMS address spaces are running on. The IRLMs perform the locking at block level, as in the previous case. However, instead of holding details of the locks in the IRLM address space, the lock tables are stored in shared structures in the sysplex coupling facility. Deadlocks are resolved by a timeout limit specified to IRLM.

For further details on data sharing, refer to the chapter on administering a data sharing environment in *IMS/ESA Administration Guide: System*, SC26-8013.

If you believe your application volumes may justify use of sysplex data sharing, then there are two ITSO publications that may help you, *IMS/ESA Data Sharing in a Parallel Sysplex*, SG24-4303, and *IMS/ESA Sysplex Data Sharing: An Implementation Case Study*, SG24-4831.

Chapter 13. Choosing the correct type of database

This chapter is intended to assist you in choosing the best type of database for your applications.

13.1 Applications suitable for Full Function (DL/I)

DL/I is a general access database manager, thus it is suitable for a wide variety of applications. There are as many types of applications as there are companies using IMS. Unless you have specific application requirements which warrant the use of IMS Fast Path databases, Full Function databases are generally the best choice.

13.2 Applications suitable for Fast Path (DEDB)

Obviously, the application area for which the DEDB was originally designed — the management of customer accounts in a retail bank — is an ideal candidate for that database implementation, but it is far from the only one, and some of the more recent additions to the functions of the DEDB (notably VSO, which has been available since IMS/ESA V5) extend the application areas for which you should consider using a DEDB.

Many users have not realized the dramatic operational and performance benefits available with DEDBs and have, for various reasons, not familiarized themselves with that database implementation. In one example, a customer who preferred to use only DB2 for new databases was convinced to use a DEDB with a saving of some 65% in the processor requirements for that very large application. Initially, it may seem daunting to introduce a DEDB to an organization where the users are unfamiliar with that technology, but practical experience has shown that user education is really a small, easily contained issue, and the benefits of the DEDB for well suited applications, greatly outweigh the additional effort for the introduction of this new type of database.

Several different requirements should be considered as indicators as to whether the DEDB is likely to be a suitable mechanism:

- 13.2.1, “Very large databases” on page 104
- 13.2.2, “High availability requirements” on page 104
- 13.2.3, “Highly active databases” on page 105
- 13.2.4, “Limited data lifetime” on page 105
- 13.2.5, “Extreme performance levels” on page 105
- 13.2.6, “DEDB: reduced I/O usage” on page 106
- 13.2.7, “DEDB: CPU utilization” on page 106

13.2.1 Very large databases

The structure of the DEDB was designed to facilitate handling of very large databases by implementing each database as 1-240 areas, each of which may be as large as 4 GB. This relieves the size constraints of conventional DL/I databases, and provides an effective mechanism for processing and managing large databases as multiple units.

The areas are relatively independent of each other — and for batch-style processing, multiple areas can easily be processed in parallel, which dramatically reduces run times for such things as overnight update and report runs (executing as bumps), image copy jobs, and similar tasks that involve processing entire databases. If the area breakup can be in processing units, then individual areas can be processed independently. For example, if an area is dedicated to one subsidiary within a conglomerate business, then the processing for that subsidiary can be optimized and performed independently of other subsidiaries.

The algorithm by which data records are assigned to area is entirely under the user control, so data and application requirements can readily exploit the area structures by using separate areas for groupings of data that have different characteristics (and so require different space definitions for optimal performance) or are processed on different schedules. For example, separate areas could be used for records representing different business units, or different regions for which processing is done on different cycles.

The high performance characteristics of the DEDB, discussed below, are particularly important for large databases, as in many instances, the sheer size of a database may impose a requirement for high performance, particularly in batch or “whole of database” processing.

13.2.2 High availability requirements

Since the implementation of a DEDB is designed so that almost all maintenance such as image copying or database reorganization can be done while the database is online, the requirement for extended outages for planned maintenance is dramatically reduced. During a database reorganization, only a small part of the data, one unit of work (UOW), which might typically be a few tens of control Intervals, is locked at any particular time. Thus online processing can generally proceed with minimal impact during a reorganization.

Additionally, the scheduling of a PSB to access the DEDB does not depend on the availability of all areas, so even when one area is not available for access, say a database recovery is in progress, then all other areas are accessible and transaction and BMP scheduling can occur. In one customer’s retail banking DEDB, 20 areas located on 20 separate DASD device were used, so that even if a single area or the DASD device on which it was located were unavailable, 95% of the data should still be accessible.

The DL/I programming interface to the DEDB provides for an application that attempts to access data in an area that is currently not available to be given the same DL/I status code as for an I/O error, which now generalizes the meaning of that status code to be: "The data you requested is temporarily not available". This can be meaningfully handled by most existing programs. The net result of this is that, when one area of database is unavailable, processing for other areas can proceed normally, which is in contrast with an IMS Full Function database, where unavailability of any part of the database precludes all scheduling.

13.2.3 Highly active databases

If the Virtual Storage Option (VSO) of the DEDB is exploited for one or more areas, then all records in those areas are held in virtual storage during database processing. Updates are logged for recoverability and written to DASD periodically in an asynchronous process. If the DEDB is participating in Parallel Sysplex data sharing, then all database updates are written to structure in the coupling facility to be shared with other IMSs. These mechanisms avoid I/O for most database accesses.

13.2.4 Limited data lifetime

A user can define that one segment within a DEDB is stored in a form called the sequential dependent segment. This is managed by IMS in a very different way from other data segments in the DEDB (where the storage mechanisms are rather similar to those used in a Full Function database). The data entry segment type is designed to optimize the interim storage and retrieval of data (as the name suggests) for which only a short lifetime is normal before the data is reprocessed by some form of batch processing. The sequential dependent data storage mechanism is therefore ideally suited to data entry style applications where data may be inserted progressively over a period, is not accessed heavily by online transactions, and is extracted for reprocessing in bulk at intervals, and deleted in bulk at some time after that. This suits such applications as the maintenance of an audit trail or the collection of transactions for batch reprocessing, sometimes involving very high rates of data insertion into the database.

13.2.5 Extreme performance levels

There are several different aspects of the DEDB that are designed to minimize the number of I/Os necessary for data access and update, to minimize the path length of instructions used for a DEDB activity, and to ensure parallelism between multiple nearly simultaneous applications. These improve the performance of online and BMP processing, thus allowing either higher workloads on any given processor, or reduced processing costs for a given application workload.

This capacity to handle extreme workloads has been amply demonstrated by various Fast Path benchmarks showing the capability to exceed 1,000 transactions per second even in 1993. More recent work has far exceeded even that performance level. Note that these benchmarks were achieved on processors that are quite small by today's standards.

13.2.6 DEDB: reduced I/O usage

The space search and usage algorithms for the root and direct dependent segment data in a DEDB are markedly simpler than other database implementations, while usually providing good locality of data, thus reducing the number of I/Os required for a given process compared to say a DL/I database implementation.

If the sequential dependent segment type is used, the total number of I/Os required for insertion of data and deletion is substantially less than for other segment types for the typical insert-retrieve-delete sequence of processing.

When DEDB database control intervals are written as the result of add/update/delete calls, the I/Os are asynchronous to the transaction or BMP unit of work. The I/Os are done after the sync point is complete — which results in improved transaction response times, and improved BMP elapsed times

If the high speed sequential processing (HSSP) functions of the DEDB are employed, many of the Read I/Os to access data are also done asynchronously, which can again greatly reduce BMP elapsed times.

DEDB updates are logged in a slightly different manner from Full Function database updates. During each Sync interval (an online transaction or BMP Checkpoint), changed data is written to the database only after the sync point processing has committed the changes. There is no requirement for “before” image data to be logged as would happen for Full Function database updates, thus substantially reducing the volumes of log data and thus reducing the total I/O workload.

13.2.7 DEDB: CPU utilization

Since the DEDB implementation uses simpler algorithms for most functions than IMS Full Function, the CPU utilization for similar processing workloads is typically approximately one half that of IMS Full Function. It is also notable that almost all processing for an online transaction, or for a BMP, takes place under the TCB of the region processing that transaction or BMP, thus allowing a very high degree of transaction parallelism.

All the mechanisms mentioned above to reduce I/Os have a secondary effect that the CPU utilization to perform those I/Os is similarly reduced.

13.3 Applications suitable for Fast Path

Obviously, the application for which the DEDB was originally built — retail banking — is a suitable environment where the database designer should consider using a DEDB, but there are many other circumstances where the particular characteristics of the DEDB may be beneficial. The following examples are drawn from many industries and show that, especially since the introduction of VSO (IMS V5), the DEDB may be very effective.

- **Account database — retail bank:**

This application exploits the characteristic effectiveness of the sequential dependent to collect transactions for reprocessing (posting) at the end of the business day. The low cost of deletion of the sequential dependent reduces the overheads for very large numbers of transactions. The DEDB also allows near-continuous operation and portioning of the data to ensure manageability of the large databases involved.

Access to the account by account number requires only one I/O and almost all processing can be done, with one read I/O and one write I/O since we can practically ignore the I/Os for the sequential dependents.

One disadvantage is that the DEDB requires all access to the account be via the account number, so a second database is necessary to access the account record from another key. This would be the credit card database mentioned below, and so access via the credit card would require one I/O to the credit card database and one to the account database.

- **Credit card database — retail bank:**

To provide access to an account DEDB from a credit card number, a cross-reference database is required and must be maintained by user programming (unlike a secondary index). This is usually a root-only database with little data in each segment, primarily the relevant account number to which the credit card transactions are to be posted, and the status of the card itself.

- **Teller control database — retail bank:**

Teller transaction journals can be readily kept as sequential dependents, provided they are not usually required for online access. If online access is necessary then a direct dependent segment would be more suitable.

- **Account database — utility company:**

A utility company (telephone, electricity, gas or water) requires very similar processing to that for a retail bank and a similar data structure is very suitable. All the remarks above for the banking application are relevant, though there is one interesting difference. In the case of a telephone utility, there is a need to refer to a meter database that is similar to a credit card database for the banking environment.

- **Meter database — utility company:**

This database provides a cross reference from a meter identifier to the account that is to be billed for usage recorded by that meter. Since meter readings are input and processed in batches that tend to be quite predictable, then it could be very effective to exploit this predictability, to put the meter data into areas that correspond to the batches of data that are processed in one BMP execution and to switch an area into VSO prior to each batch run, and out of VSO afterwards.

- **Audit and history database:**

This database illustrates a good use of the sequential dependent segment type to provide a historical journal of activities. For a non-shared DEDB, the sequential dependents will be in absolute time sequence, and if the DEDB is shared (IMS V6 onwards), then the segments can be readily sorted into time sequence.

- **Status report database:**

This database was designed to hold a few tens of lines of report data for each of several thousand destinations. Each report was generated daily, and access was required to each report for typically three days before it could be purged. Access to the reports was occasional, and a high percentage of the data was accessed either once or not at all.

By using the sequential dependent to store each report, the detail lines of each report were kept in the same or adjacent CIs (as they were inserted within one sync interval), so that online access to them was quite efficient.

As each report was generated, a summary was placed in the report summary segment so that it could be accessed at the same time as the root segment.

- **Bet status database — gambling system:**

This database is designed to support an online totalizer system where the total of all bets placed on a given horse is required to be kept up to date with very high concurrency, and sometimes there are high transaction rates against a few records for a relatively short period. Here, the judicious use of VSO can allow records that are currently active to be held in VSO, while less active records will stay on DASD. Note that there is an option to restrict this database to a root-only design, obeying the constraints of the old main storage database, and thus allowing the use of FLD calls that can reduce the scope and duration of data locking. This should substantially increase the level of maximum concurrency that can be achieved.

Chapter 14. Database reorganization processing

In this chapter, we provide an overview of the database reorganization tasks that will need to be performed by the IMS database administrator function. We start with general background information regarding IMS database reorganization, then look in more detail at reorganizing HD databases.

Specifically, this chapter:

- Introduces the function of database reorganization in a DL/I environment. It is a first-time general introduction into the requirements for, and the process of, IMS database reorganization.
- Gives a formal description of the available DL/I utilities for reorganizing HD databases.
- Introduces the use of the utilities for particular situations. It describes what needs to be run to reorganize an HD database with and without logical relationships or secondary indices. It also looks at partial reorganization of HD databases. Finally, there is a short discussion on initial loading of databases with logical relationships/secondary indices, as this also requires the reorganization utilities to build the logical relationships/secondary indices.

14.1 Why is reorganization necessary ?

Reorganization is the process of changing the physical storage and/or structure of a database to better achieve the application's performance requirements. We distinguish between the following two types: physical reorganization, to optimize the physical storage of the database; and restructuring, to alter the database structure.

The most common reasons a database will need reorganizing are:

- To reclaim and consolidate free space that has become fragmented due to repeated insertion and deletion of segments.
- To optimize the physical storage of the database segments for maximum performance (get dependent segments that are in distant blocks, increasing physical I/O, back in the same block as the parent and/or root). This situation is normally the result of high update activity on the database.
- To alter the structure of the database, change the size of the database data sets, alter the HDAM root addressable area, add or delete segment types.

The first two reasons would be described as reorganization, the last one as restructuring. The need for reorganization is always due to change, either setting up a new database, amending the structure of the database as application requirements change, or as a result of update activity against the database. If you do not update a database, then once you have gotten it to an optimum state for performance, there is no further need to reorganize it.

Reorganizing and restructuring the databases is only part of the process of tuning and monitoring access to IMS databases. There are also many things that can be done to tune the database manager component in the IMS subsystem and the applications accessing of the databases. This is covered in detail in Chapters 11 and 12 of the ITSO publication *IMS Version 5 Performance Guide*, SG24-4637.

14.2 When to reorganize

There are no fixed rules about when to reorganize. There are two approaches to deciding when to reorganize, reactive and proactive. You will probably do a mixture of both. When you initially install the application and set up the databases, a lot of the reorganization will be done reactively, as performance and space problems manifest themselves (while you can reduce this by careful analysis of the databases and application access to them, there will normally be things that only come to light after implementation). As you develop a history of the behavior of the application and the databases, the scheduling of reorganization should become more proactive.

Reactive scheduling of reorganization will normally be a result of perceived problems with the performance of the application, or problems with shortage of freespace in the database.

Where there are perceived application performance problems, you need to monitor closely what the application is doing. The initial thing to look at is, what the average and maximum online response times and batch run times are. Are they excessive for the amount of work the application is doing? The ITSO publication *IMS Version 5 Performance Guide*, SG24-4637 covers in great detail monitoring and investigating performance of IMS application and subsystems. If there are performance problems, then go through the process described in the document to monitor the performance and identify where the problems are.

Only once you have gone through the procedures detailed in this document and identified potential problems with the databases should you start to look at reorganizing the database. Do not look only at the total time that the application program takes for database processing, but also look at the amount of database calls it is processing. For example, if an online application is taking 10 seconds for database processing, but is reading 3-4000 database segments, then there may be little room for database tuning. However, you may want to look more closely at why (and whether) the application really needs to read all these segments. The solution to performance problems is normally an interactive process involving the database administrator, application support function, and the operating system support function, as all three control areas that affect performance.

When you encounter problems due to shortage of space in database data sets, there is little you can do but schedule a database reorganization to increase the database size. However, you should then pursue the growth rate with the application support function (this is where it is useful to have a history of the volume of the application data stored in the database over time). Questions to ask are whether growth will continue at the current rate, or at a different rate, and whether this data all needs to be online. Remember there are finite architectural limits to the size of the databases which vary depending on the IMS and operating system access methods.

The proactive approach to scheduling database reorganization relies on regular monitoring of the databases. Some products for monitoring the databases are covered in more detail in the next section. In addition, you should maintain a history of the monitoring information you collect, so you can analyze this for trends and schedule database reorganization and restructuring before any problems occur. When you decide to make a change to a database, only change one thing at a time, if possible, and then monitor application performance before and after the change so you can see what effect this one change had.

The main things you will be doing when you look at the monitoring data will be to try to minimize the physical I/O for each database access, and optimize the free space available in the database so it is not excessive, but sufficient for normal update access on the databases.

The physical I/O from the disk storage into the buffers in the IMS subsystem is the major component of the elapsed time for database access. You will want to minimize this by:

1. Making the best use of buffers in the IMS subsystem; the more requests for database access you satisfy from the buffers, the fewer physical I/Os are necessary. This is covered in the *IMS Version 5 Performance Guide*, SG24-4637.
2. Minimizing the number of physical I/Os when a segment does have to be retrieved from disk. For example, trying to place as many dependents as possible in the same block/CI as its parent, ensuring HDAM root segments are in the same block/CI as the RAP. This is where database reorganization and restructuring is used.

While there are no fixed guidelines for when to reorganize an IMS database, the following guidelines were used successfully with a medium-sized commercial application using IMS HD databases stored in VSAM files. You may wish to use them as a starting point for scheduling database reorganization and, when you have monitored the effects of the reorganization, adjust these parameters accordingly.

- For HDAM databases, less than 75% of root segments in the root addressable area (RAA). Recalculate the RAA (as described in the database design section). reorganize database, if calculation of RAA showed it needed to be larger, then restructure at same time.
- For HD databases, less than 50% of database records have all segments making up the record (root and dependents) in the same block/CI.
- For HDAM databases, less than 50% of root anchor points (RAP's) point to root segments in the same block/CI. (that is, the RAP points to a root that has been placed in another block/CI because there is not room in this block/CI) This causes two I/Os, one to the RAP block, and one to the block that the root is in, instead of one I/O.
- Less than 20% freespace in an HD database. You may want to increase this limit if you have volatile data or infrequent windows for reorganization.
- VSAM or OSAM file in secondary extents. You may wish to resize the file, if this is caused by growth.
- VSAM KSDS (index) with CA splits or more than 15 CI splits.

- VSAM KSDS (index) with less than 20% free space (as IMS manages freespace in VSAM ESDS, this only applies to a KSDS).

14.3 Monitoring the databases

The database monitoring divides in to two categories. Monitoring program and subsystem access to the databases, and monitoring the structure, space usage and pointer chains in the actual database data sets.

The first type of monitoring is described in detail in the ITSO publication *IMS Version 5 Performance Guide*, SG24-4637. The principle tools used to monitor database access are:

- The IMS monitor, to gather details of buffer usage and database calls over a specified time period in an IMS subsystem.
- The //DFSSTAT DD card, used in batch JCL to provide a summary of buffer usage and database calls. As there is very little overhead in including this (the details printed to the DD at region termination are accumulated by the IMS region controller whether they are output or not), it is normally worthwhile putting this in all batch jobs.
- Running the DB monitor on a batch job, to collect similar details to the IMS monitor in an online system. As there is an overhead on running this, it would normally only be turned on when specific problems are being investigated.

There are a number of products available to let you monitor the databases, and the data sets in which they are stored. These include the following.

IMS Database Tools (DBT) V2, 5685-093, which consists of a collection of utility programs that are very useful to the database administrator. The utilities include pointer checkers for HD and DEDB databases that, besides confirming the integrity of the databases, collect information that can be used for performance analysis. This can also be fed into the historic reporting component of DBT to provide trend analysis. There are two new redbooks related to the IMS database tools: *IMS/ESA Database Tools, Volume I: Database Manager Tools*, SG24-5166; and *IMS/ESA Database Tools, Volume II: System Extension Tools*, SG24-5242.

The VSAM Access Method Services (IDCAMS) utility can be used to list the catalog details (LISTCAT command) for VSAM data sets. This can show space usage, extents and CI/CA splits. Remember that IMS processes the ESDS's at the CI level, so some figures in the catalogue listing (for example, CI freespace) will not be relevant.

There are various facilities in MVS (ISPF option 3.2) to get details of the OSAM data sets from the catalogue.

The IMS database surveyor, DFSPRSUR, can provide some information about the internal state of HD databases. See *IMS/ESA Utilities Reference: Database Manager*, SC26-8034 for further details.

14.4 Reorganization processing overview

The database reorganization process can vary from very simple to very complex, depending on the databases involved. If the databases involved do not have IMS logical relationships or secondary indices, then the process is very simple. When logical relationships and secondary indices are involved the process becomes more involved.

14.5 The reorganization process description

The process, in its simplest form, is to unload the database, delete and redefine the physical data set, and then reload it. If the database is not involved in any logical relationships and does not have any secondary indices, then that is the complete process. Database reorganization of HD databases would normally take the following steps if both logical relationships and secondary indices are involved:

1. Back up the databases (both the data and, if you are changing them, the appropriate control blocks, for example, DBDs) so you have a fallback point if there are any problems during the reorganization. See Chapter 15, “Database recovery processing” on page 123 for more information.
2. Unload the existing database data sets to sequential files using the IMS utilities. The process is discussed in 14.5.1, “Database unload processing” on page 114.
3. Delete the database data sets. If you are making any changes to the definitions of the database data sets, make them now, remembering to save the old definitions as a fallback.
4. Redefine the database data sets.
5. This step is only necessary if you are making any changes to the database structure by altering the DBD. Make the changes to the DBD and reassemble it by running the DBDGEN utility. Then run the ACBGEN utility with DBD= parameter to ensure all appropriate control blocks are regenerated. It cannot be overemphasized that you must make sure all programs/utilities use the new versions of the control blocks if you change the DBD; otherwise, database corruption will result.
6. Run the IMS utilities to reload the database. If you have altered the DBD, the utility, and any subsequent programs/utilities, should use the new DBD.
7. If the database has secondary indices, or participates in logical relationships, then you will need to run additional utilities to rebuild these connections. These connections (unless using symbolic pointers) rely on the database segments relative position in the database, which has been altered by the reorganization. The utilities will determine the new positions and amend the direct pointers in the indices and logically related databases.
8. If your databases are registered with DBRC (and they **should** be), then you will need to take an image copy of the reorganized databases. This is for the same reason as above. IMS database forward recovery, using changes recorded in IMS logs, relies on the position of the segments relative to the start of the data set, which is altered by the reorganization. You need to take the image copies to establish a new base from which the databases can be rolled forward.

14.5.1 Database unload processing

The unload processing for HD databases is very simple. The HD unload utility will unload the main database and the primary index data set if the database is HIDAM. The output of the utility is a sequential data set which is input to the HD reload utility. If the database is a HIDAM database, then the primary index database must also be present. The utility can only unload a single database at a time. If there are logically related databases which are to be reorganized at the same time then the step should be executed once for each database. Figure 32 shows a diagram of the utility.

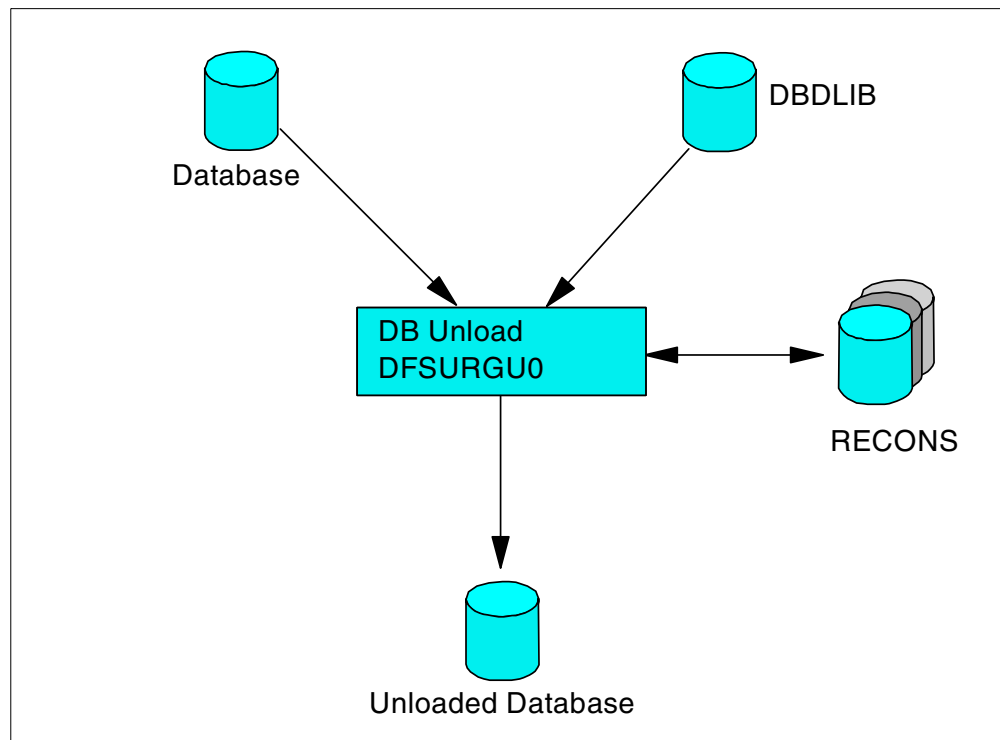


Figure 32. Database unload processing

There are some considerations to be kept in mind:

- The database is allocated by the DD name(s) in the DBD. Dynamic allocation cannot be used; the database DD card(s) must be present.
- If a HIDAM database is being unloaded, the primary index database DD card must also be present.
- The utility will check with DBRC for database registration. If the database is registered, then the utility will request RD access authorization. It will be allowed to authorize the database even if the **PROHIBIT AUTH** flag is set on.
- If the database is being reorganized to change the structure of the database, then the old DBD definition should be used.
- Regardless of how many database data set groups the database is divided into, there is only one output data set.
- The reload utility can only unload one database per job step. To unload multiple databases, you must use multiple job steps.

14.5.2 Defining databases

If the access method used for the database is VSAM, then an IDCAMS job step is required to delete and redefine the VSAM cluster. The reload utility will fail if the data sets are not empty. If OSAM is used, the a DISP=OLD can be used to overwrite the data set. However, if the database is on more than a single DASD volume, it is highly recommend that you delete the data set and redefine it (IEFBR14) to ensure that the correct end-of-file marker is placed.

14.5.3 Database reload processing

The reload processing can be more complex then the unload processing. If the database is does not have any secondary indices and is not involved in a logical relationship, then the database can simply be reloaded. The following sections will deal with each combination of reload processing required.

- Reload only
- Reload with secondary indices only
- Reload with logical relationship only
- Reload with both secondary indices and logical relationships

The reloading of the database itself is the same, however, there are additional utility programs that need to be run before and after the database is reorganized to rebuild logical relationships and secondary indices so they reflect the new physical positions of the segments in the reorganized database. Until all this processing is complete, the logical relationships and secondary indices are not usable. If you attempt to use them before completing this process, the applications will fail with IMS abend codes indicating that there are invalid IMS pointers.

14.5.3.1 Reload only

The reload processing for a HD database without any logical relationships or secondary indices is shown in Figure 33.

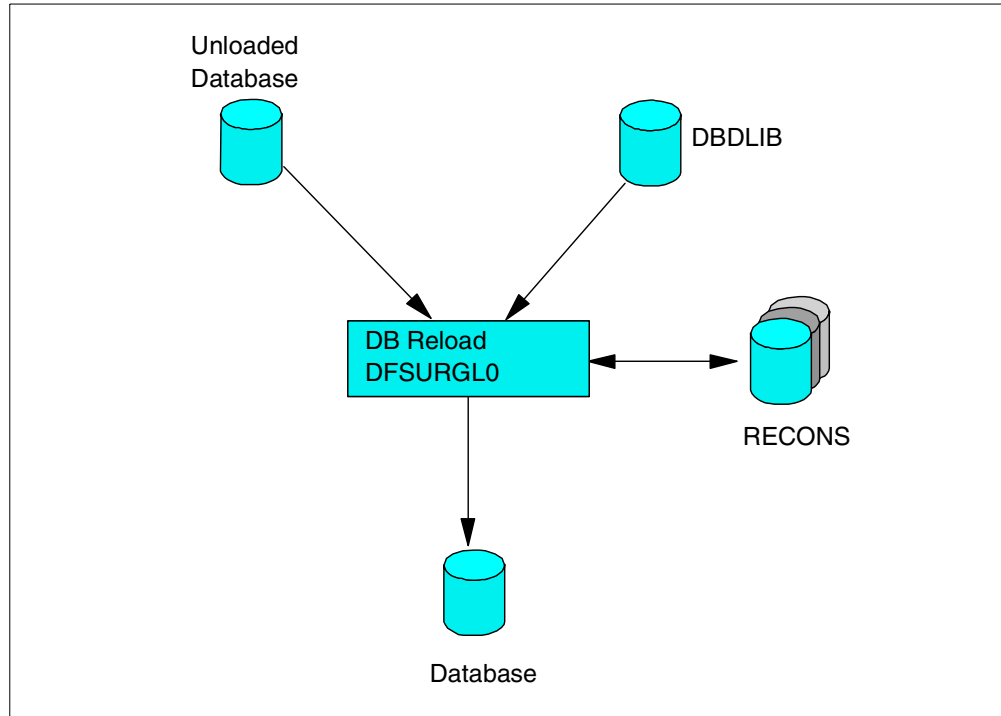


Figure 33. Database reload only

There are some considerations to be kept in mind:

- The database is allocated by the DD name(s) in the DBD. Dynamic allocation cannot be used; the database DD card(s) must be present.
- If a HIDAM database is being reloaded, the primary index database DD card must also be present.
- The utility will check with DBRC for database registration. If the database is registered, then the utility will request EX access authorization. It will be allowed to authorize the database even if the PROHIBIT AUTH flag is set on.
- If the database is being reorganized to change the structure of the database, then the new DBD definition should be used.
- Regardless of how many database data set groups the database is divided into, there is only one input data set.
- The reload utility can only reload one database per job step. To reload multiple databases you must use multiple job steps.
- The DFSURWF1 DD statement can be specified as DUMMY.

14.5.3.2 Reload with secondary indices

The reload processing for an HD database but with secondary indices requires the use of the prereorganization utility. It is used to define which databases are involved in the secondary index relationship. A control file is created with this information and passed to the subsequent utilities.

The HISAM unload utility will read the DFSURIDX data set which contains the unload secondary index segments and creates load files for each secondary index. The secondary index database themselves can be empty.

The HISAM reload utility can reload all the secondary index database unloaded by the HISAM unload utility in one JOB step. Figure 34 illustrates the reload process.

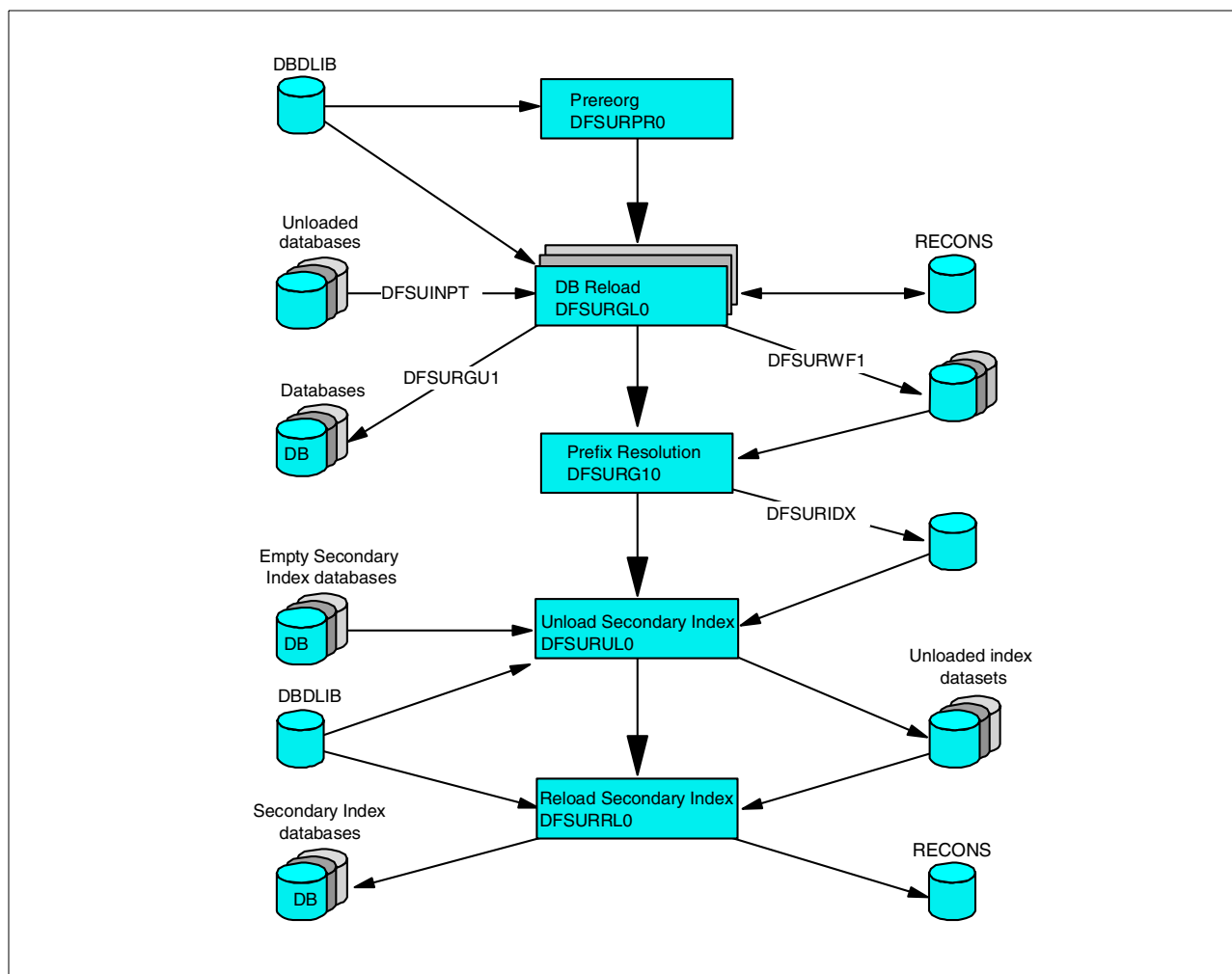


Figure 34. Reload processing with secondary indices

There are some considerations to be kept in mind:

- The database is allocated by the DD name(s) in the DBD. Dynamic allocation cannot be used, the database DD card(s) must be present.
- If a HIDAM database is being reloaded, the primary index database DD card must also be present.
- The utility will check with DBRC for database registration. If the database is registered, then the utility will request EX access authorization. It will be allowed to authorize the database even if the PROHIBIT AUTH flag is set on.
- If the database is being reorganized to change the structure of the database, then the new DBD definition should be used.
- Regardless of how many database data set groups the database is divided into, there is only one input data set.
- The reload utility can only reload one database per job step. To reload multiple databases, you must use multiple job steps.

- The DFSURWF1 DD statement can be specified as DUMMY, but it must be present.

14.5.3.3 Reload with logical relationships

The reload processing for a HD database but with logical relationships requires the use of the prereorganization utility. It is used to define which databases are involved in the logical relationship. A control file is created with this information and passed to the subsequent utilities. If all the databases logically related to each other are being reloaded then the DBIL option on the control card should be used. These will reset all the pointers and logical parent counters. If not then the DBR option should be used.

All databases involved in the logical relationships should normally be reloaded. The DFSURWF1 work files from all steps should be passed to the prefix update utility as illustrated in Figure 35. The HISAM unload utility will read the DFSURIDX data set, which contains the unload secondary index segments and creates load files for each secondary index. The secondary index database themselves can be empty.

The prefix resolution utility will extract the RBAs from the required segments and sort them. This file will be passed the prefix update utility to update the database segment prefixes.

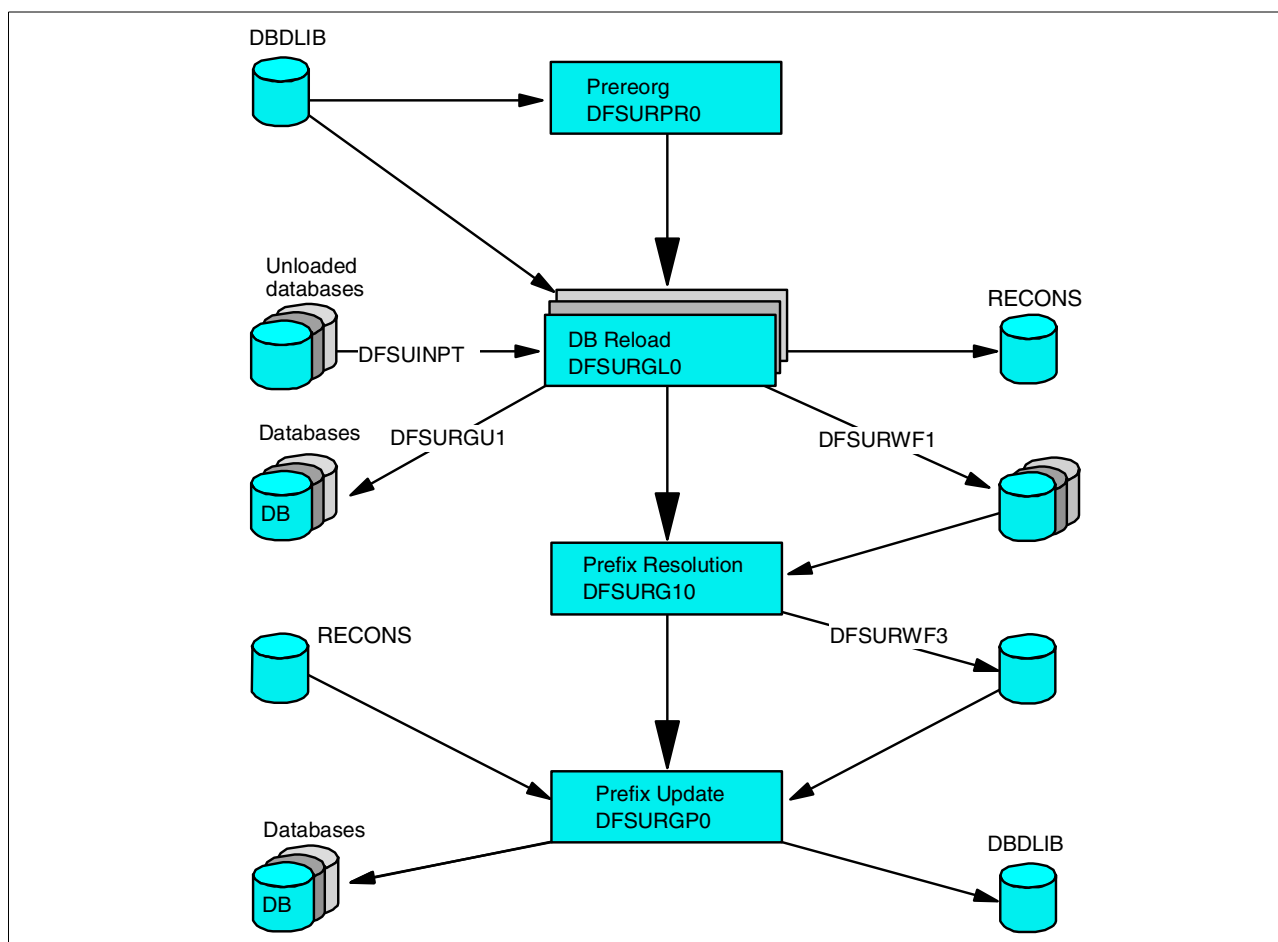


Figure 35. Database reload with logical relationships

There are some considerations to be kept in mind:

- The database is allocated by the DD name(s) in the DBD. Dynamic allocation cannot be used, the database DD card(s) must be present.
- If a HIDAM database is being reloaded, the primary index database DD card must also be present.
- The utility will check with DBRC for database registration. If the database is registered then the utility will request **EX** access authorization. It will be allowed to authorize the database even if the **PROHIBIT AUTH** flag is set on.
- If the database is being reorganized to change the structure of the database, then the new DBD definition should be used.
- The reload utility can only reload one database per job step. To reload multiple databases, you must use multiple job steps.
- The DFSURWF1 DD statement must be present.
- The prefix update utility will acquire **EX** access to the databases being updated.
- The **IMAGE COPY NEEDED** flag will be set on by the reload utility.

14.5.3.4 Reload with logical relationship and secondary indices

The reload processing for both secondary indices and logical relationships is a combination of both the individual processes described above.

The reload processing for a HD database but with secondary indices and logical relationships requires the use of the prereorganization utility. It is used to define which databases are involved in the relationships. A control file is created with this information and passed to the subsequent utilities.

The prefix resolution utility will extract the RBAs from the required segments and sort them. This file will be passed the prefix update utility to update the database segment prefixes. It will also create a file with the secondary index information to be passed the HISAM unload utility.

The HISAM unload utility will read the DFSURIDX data set which contains the unload secondary index segments and creates load files for each secondary index. The secondary index database themselves can be empty.

The HISAM reload utility can reload all the secondary index database unloaded by the HISAM unload utility in one JOB step. Figure 36 illustrates the reload process.

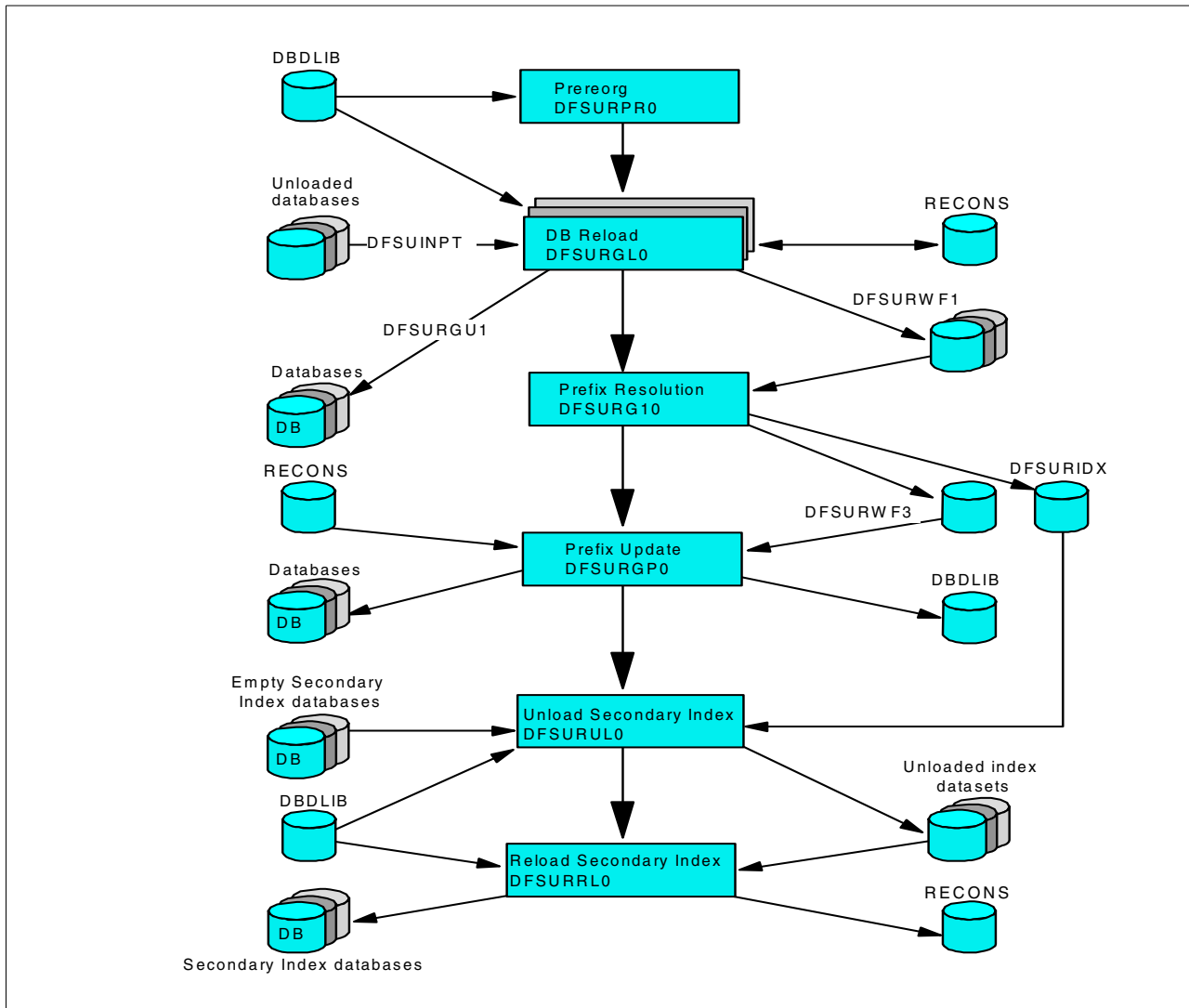


Figure 36. Database reload with secondary indices and logical relationships

There are some considerations to be kept in mind:

- The database is allocated by the DD name(s) in the DBD. Dynamic allocation cannot be used, the database DD card(s) must be present.
- If a HIDAM database is being reloaded, the primary index database DD card must also be present.
- The utility will check with DBRC for database registration. If the database is registered then the utility will request **EX** access authorization. It will be allowed to authorize the database even if the **PROHIBIT AUTH** flag is set on.
- If the database is being reorganized to change the structure of the database, then the new DBD definition should be used.
- The reload utility can only reload one database per job step. To reload multiple databases you must use multiple job steps.
- The DFSURWF1 DD statement must be present.

- The prefix update utility will acquire **EX** access to the databases being updated.

The **IMAGE COPY NEEDED** flag will be set on by the reload utility.

14.6 Fast Path reorganization

The process for reorganizing a Fast Path DEDB can be appreciably different.

If you are only reorganizing to reclaim fragmented free space and/or get the best placement of the segments for performance (that is, DBD/data set definitions not being changed), then you can run the high speed DEDB direct reorganization utility DBFUHDR0. This can be run without making the database unavailable (that is, no service outage). See *IMS/ESA Utilities Reference: Database Manager*, SC26-8034 for further details.

If you are reorganizing a DEDB to alter the structure, then you need to have your own user-written programs to unload and reload the database data set at the appropriate points, or use the DEDB unload/reload utility programs from the separately priced *IMS Database Tools (DBT) V2*, 5685-093. You also need to run the DEDB initialization utility, provided with the IMS base product, immediately prior to reloading the database. However, as the DEDB does not support secondary indices and logical relationships, you do not have to worry about running further utilities after the database is reloaded.

More information about the database reorganization process, and what steps you have to take to alter specific attributes of the structure of the database are in the chapter on monitoring and tuning the databases in *IMS/ESA Administration Guide: Database Manager*, SC26-8012.

The IMS utilities available for database reorganization are described in *IMS/ESA Utilities Reference: Database Manager*, SC26-8034.

Chapter 15. Database recovery processing

This chapter provides an overview of backup and recovery tasks that need to be performed by the IMS database administrator function. It gives a general background on IMS database backup and recovery, then looks in more detail at the sample application.

15.1 About this chapter

This chapter discusses:

1. Overview of database recovery
2. Overview of database backup and recovery utilities:
 - Image copy utility
 - Recovery utility
 - Batch backout utility
 - Change accumulation utility
3. Implementing backup and recovery procedures

15.2 Overview of database recovery

Database recovery, in its simplest form, is the restoration of a database after its (partial) destruction due to some failure. In order to facilitate this process, some forward planning needs to be done.

Periodically, a copy of the data in the database is saved. This copy is normally referred to as a backup or image copy. These image copies can reside on DASD or cartridges. Though this process can be done anytime, it is normally done when there is no other database activity at the same time. This creates a complete backup. There are other strategies for taking a database backup, but they will not be discussed in this book.

In addition to taking an image copy of the database(s), all changes made to the data in the database can be logged and saved, at least until the next image copy. These changes are contained in data sets called log data sets. This provides a complete recovery environment so that no data is lost in the event of a system or application failure.

There is an IMS facility called database recovery control (DBRC) which provides database integrity and can be used to help ensure that there is always a recovery process available. The use of DBRC to control database backup and recovery is not mandatory, but is **highly recommended**.

15.2.1 When is recovery needed ?

Database recovery is normally on done when there has been a failure of some sort. Most of the time it is done as a result of a system, hardware, or application failure. However, it can be used to return a database to a point-in-time to recover out of application logic failures.

In general, a database may need to be recovered under the following circumstances:

1. A DLI batch update job fails after making at least one database update.
2. A failure has occurred on a physical DASD device.
3. A failure has occurred in a database recovery utility.
4. A failure of dynamic backout or batch backout utility has occurred.
5. An IMS online system failure and emergency restart has not been completed.

15.2.2 Online programs

IMS online transactions use dynamic backout to “undo” updates done in any incomplete unit of work. Abending online programs are automatically backed out by the online system using the log records. In addition, if the system should fail while an application program is active, any updates made by that program will be automatically backed out when the system is restarted.

If the program was a BMP, the updates are automatically backed out to its most recent checkpoint. Because of this automatic backout, the recovery of individual databases will not be needed.

At IMS restart time, if the emergency restart cannot complete the backout for any individual transactions, then the databases affected by those updates are stopped, and DBRC is requested to set the recovery needed flag to ensure that a correct recovery is completed before the database is opened for more updates. In the case of dynamic backout failure, a batch backout or database recovery needs to be performed, depending on the reason for the backout failure.

15.2.3 DLI batch update programs

DLI Batch update programs can make use of dynamic backout like BMP, provided the following JCL changes are done:

1. The BKO=Y parameter is set in the EXEC statement
2. A DASD log data set is provided in the IEFRDER DD statement
3. A ROLB Call is issued in the program code for non-system abends

The dynamic backout will then back out the updates to the last checkpoint found on the log data set.

15.3 The database utilities

DL/I provides four utilities for recovering a database. The diagram in Figure 37 illustrates the relationship between these utilities.

A description of these utilities and their basic function follows:

1. Database image copy utility for creation of image copies of databases.
2. Database change accumulation utility for accumulation of database changes from DL/I log tapes since the last complete image copy.
3. Database recovery utility for restoration of the database, using a prior database image copy and the accumulated changes from DL/I log tapes.

4. Database backout utility for removal of changes made to databases by a specific application program.

A fifth utility program, the system log recovery utility (DFSULTRO), is used to close a log data set in the event of an operating system or hardware failure, thus enabling use of the log by the four principal programs of the recovery system.

For those databases which consist of multiple data sets, recovery is done by individual data set. To recover a complete database composed of multiple data sets, database recovery must be performed for each of its component data sets.

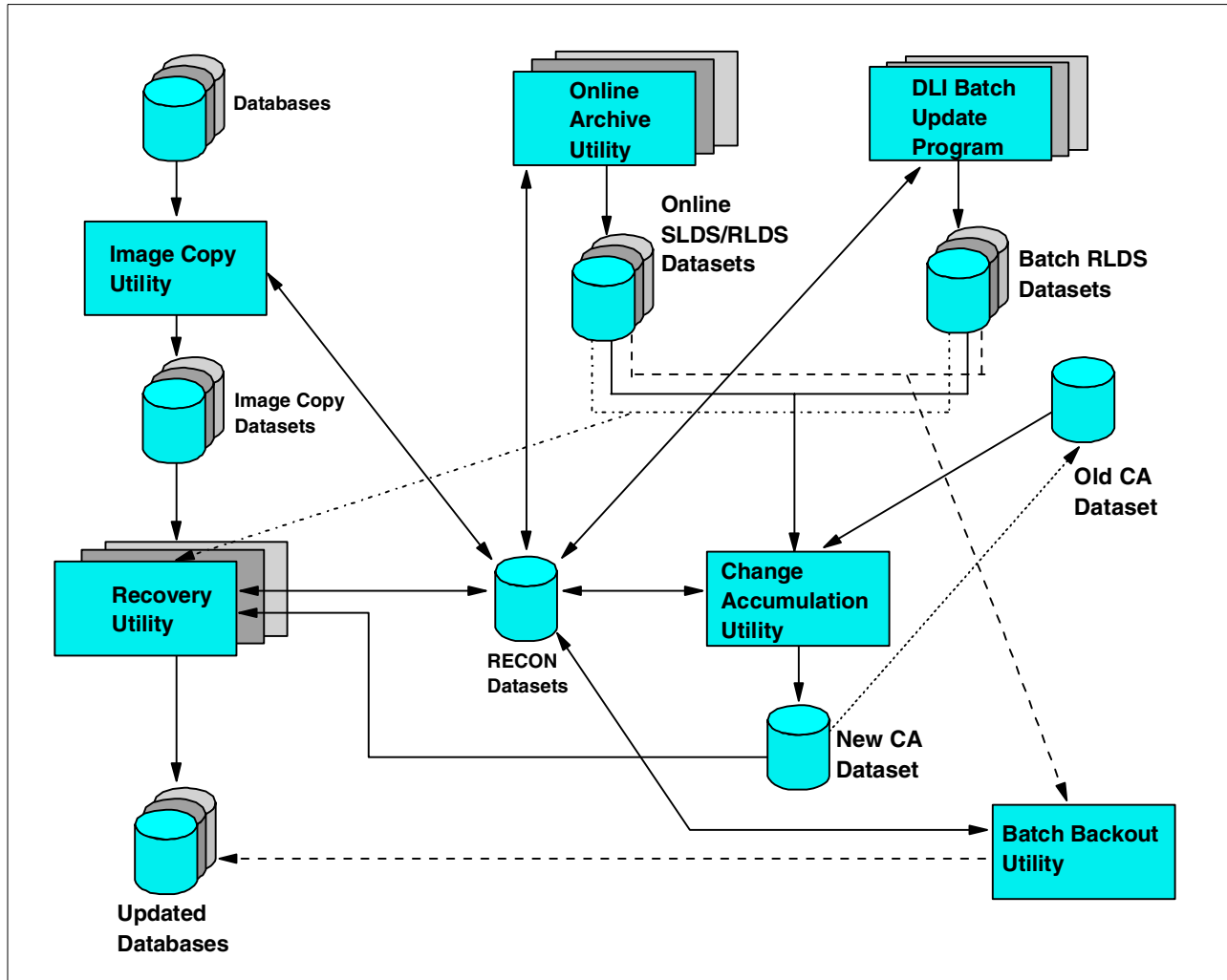


Figure 37. Overview of recovery utilities

15.4 Overview of backup/recovery utilities

This section will give a brief description of the four major utilities used in database recovery.

15.4.1 Database image copy utility (DFSUDMP0)

The database image copy utility creates a copy of the data sets within the databases. the output data sets is called an IMAGE COPY. It is a sequential data set and can only be used as input to the database Recovery utility. The IMAGE copy utility does not use DLI to process the database. Track I/O is used. There is no internal checking to determine if all the IMS internal pointer are correct. There are tools available to run as part of the image copy utility to do this checking. It is recommended that at least periodic checking of these internal pointers is done.

There can be no changes to the DBD when this databases is recovered using the IMS recovery utility. In order to make changes to the DBD, a database reorganization is needed to implement those changes.

Multiple databases and data sets can be copied with one execution of the image copy utility. All data sets of a database should be copied at the same time. In our subset, we presume that all database data sets are dumped at the same time, that is, no intervening database processing.

DBRC can be used to generate this utility if required. A redbook on DBRC, *Database Recovery Control (DBRC) Examples and Usage Hints*, SG24-3333, gives examples and usage hints, and contains an example of how to set up a DBRC-generated image copy JCL.

A flow diagram of the database image copy utility is shown in Figure 38.

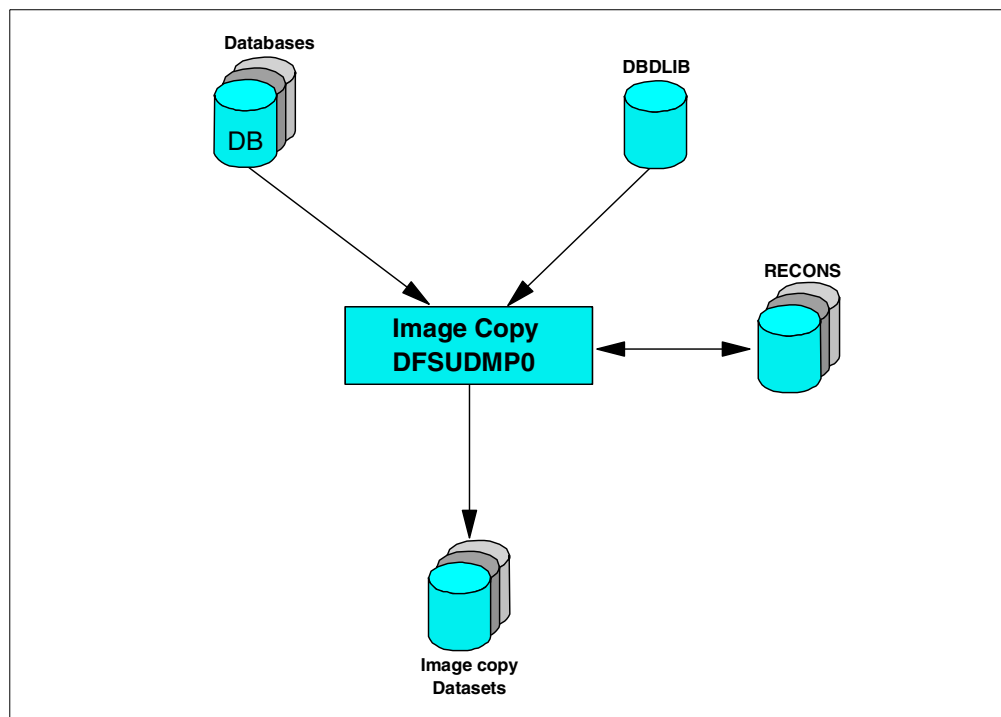


Figure 38. Image copy utility

15.4.2 Database change accumulation utility (DFSUCUM0)

The function of the database change accumulation utility is to create a sequential data set that contains only that database log records from all the log data sets which are necessary for recovery. This accumulation log data set is to be used by the database recovery utility. This accumulation is done by sorting only the required log records in physical record within data set sequence. This provides efficient database recovery whenever needed. The number of log data sets which need to be kept will be significantly reduced.

The change accumulation utility can be run independently of DL/I application programs. The new output database recovery utility.

It is highly recommended that DBRC be used to create the JCL for each execution of this utility. DBRC will ensure that a complete set of log data sets is used to create the change accumulation data set. The logs records must be supplied to in the correct sequence.

A flow diagram of the change accumulation utility is shown in Figure 39.

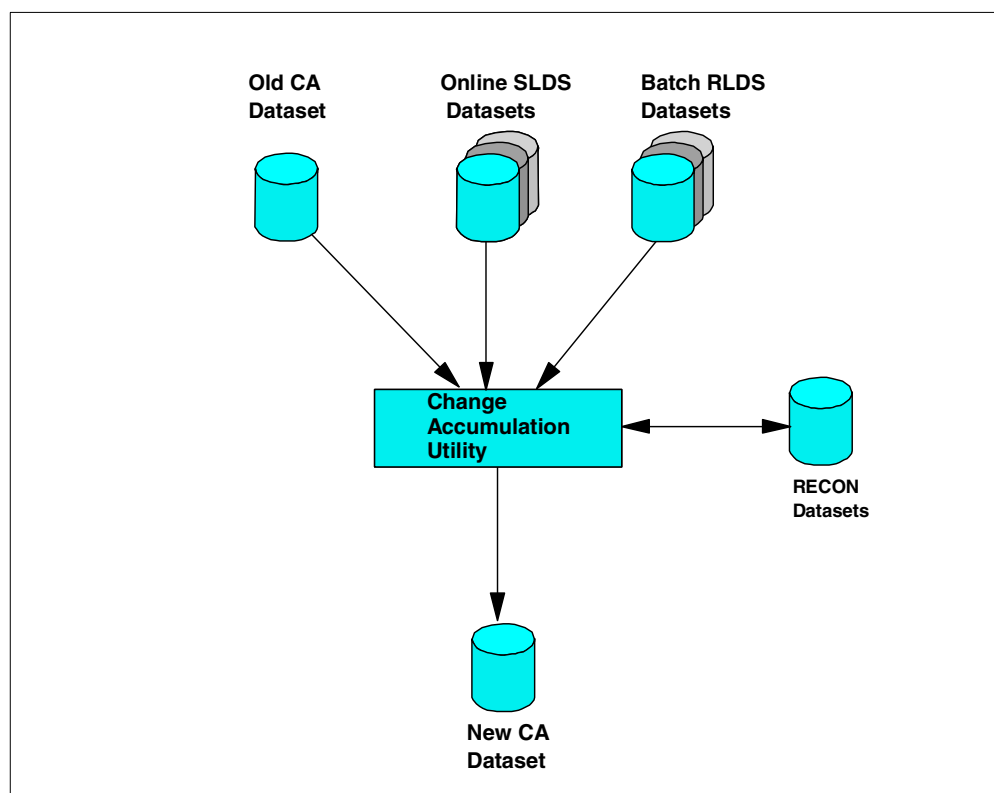


Figure 39. Change accumulation utility

The input to the database change accumulation utility consists of:

1. All log data sets created since either the last image copy utility execution or the last execution of this utility.
2. The previous change accumulation data set. This would be the output from the last execution of this utility. The first change accumulation run after a new image copy must not include any old change accumulation data set, that is, those created during the previous period.

3. An optional control statement (ID).

Output from the database change accumulation utility consists of a new change accumulation data set. This is a sequential data set containing the combined database records for all database data sets.

15.4.3 Database recovery utility (DFSURDB0)

The database recovery utility program will restore a database data set. This utility does not provide a means of recovery from application logic errors: it is the user's responsibility to ensure the integrity of the data in the database.

Unlike the image copy utility, the recovery utility recovers one database data set per job step. Thus to recover multiple data sets for a database the utility must be run once for each data set.

It is highly recommend that DBRC be used to create each execution of this utility. DBRC will ensure that all the correct inputs are supplied.

The recovery utility can be run in a number of ways depending on what input is required. Generally the main input to the recovery utility is the image copy data set. Other input can consist of any log data sets or change accumulation data sets which might be needed. The utility can be run with only the log information as input, in this case the database already existing would be used.

The input to the recovery utility consists of an image copy data set and, optionally, an accumulated change data set and any log data sets not included in the change accumulation data set.

A flow diagram is shown in Figure 40.

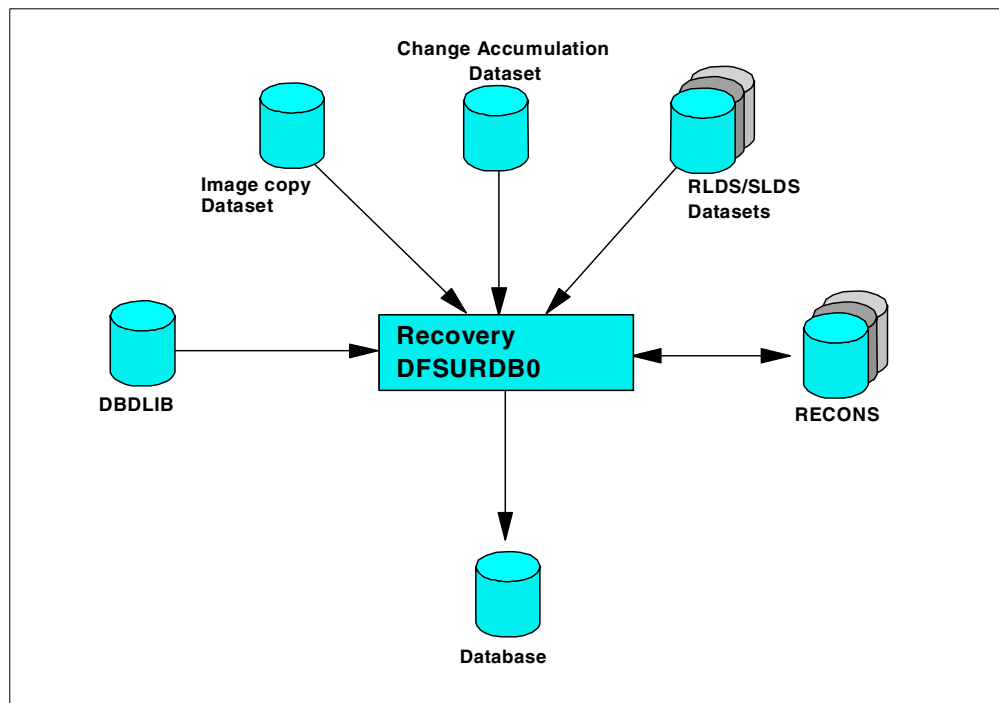


Figure 40. Recovery utility

The database recovery utility program is executed in a DL/I batch region. It will allocate the database in exclusive mode so that there can be no other database activity at the time.

15.4.4 Database batch backout utility (DFSBB000)

Batch backout, in its simplest form, is the reading of log data set(s) to back out all database updates. This is done by using the “before image data” in the log record to re-update the database segments. It has the effect of undoing the previous updates.

The database backout utility removes changes in the database which were made by a specific failing program. The following limitations apply:

- The log data set of the failing program must be on DASD.
- No other update programs should have been executed against the same database (s) between the time of the failure and the backout.

The program operates as a normal DL/I batch job. It uses the PSB used by the program whose effects are to be backed out. All databases updated by the program must be available to the backout utility. Figure 41.

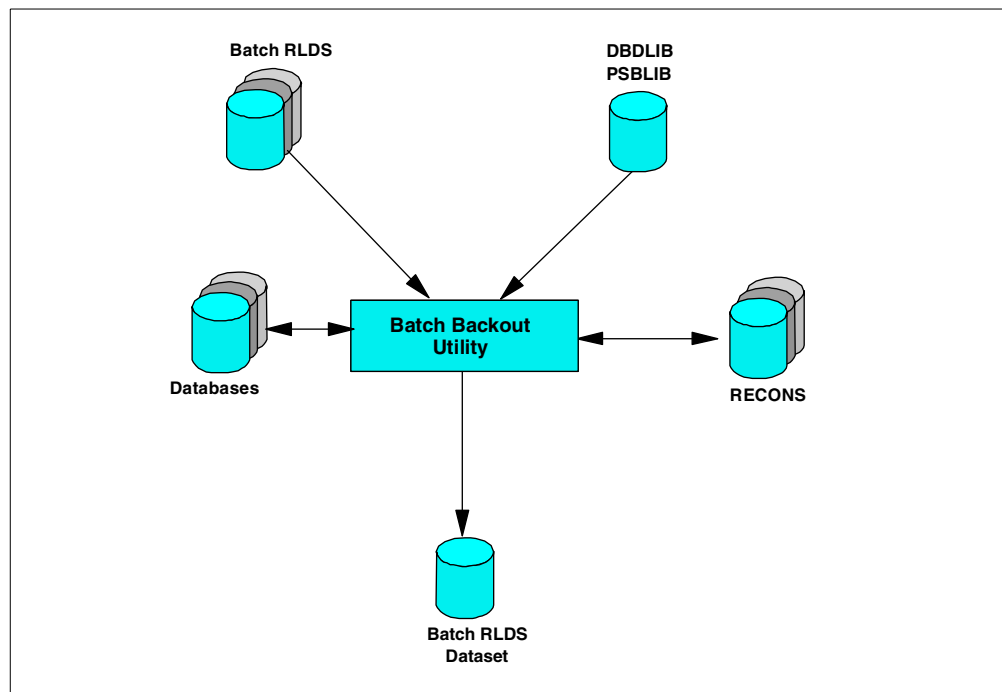


Figure 41. Batch backout utility

A log data set is created during the backout process. This data set, preceded by the log data set produced for the failing job, must be included in the next change accumulation run, as any other log data set. This data set must not be used as input to any subsequent backout attempt.

Notes:

1. If checkpoint/restart is not used, then backout always backs out all the database changes of the program.
2. If checkpoint/ restart is used (program uses XRST and CHKP-ID calls), then backout will only do backout if the specified CHKP-ID is found on the log data set during read forward. If no CHKP-ID is specified, then the last one on the log data set is used (the first one encountered during read backward).
3. If, when using checkpoint/restart, you want to be able to completely back out a job (steps), you must issue a CHKP call immediately after the XRST call, that is, before any real database activity. The CHKP-ID of this call can then be used for a full backout operation.
4. To run batch backout for a DLI batch which had completed successfully, the DBRC="C" parameter must be added to the EXEC PARM keyword.

Part 4. IMS application development

This part contains four chapters:

- An application programming overview. Refer to Chapter 16, “Application programming overview” on page 133.
- A discussion of application programming for the IMS Transaction Manager. Refer to Chapter 17, “Application coding for IMS Transaction Manager” on page 147.
- A discussion of the message formatting services that are used by application programs running in the IMS Transaction Manager. Refer to Chapter 18, “IMS message format service” on page 157.
- A discussion of application programming for the IMS Database Manager. Refer to Chapter 19, “Application coding for IMS Database Manager” on page 173.

Chapter 16. Application programming overview

This chapter explains the basics for any programming running in an IMS environment. It consists of three sections:

- Overview of application programs
- Application program structure
- IMS control blocks

16.1 Overview

IMS programs (online and batch) have a different structure than non-IMS programs. An IMS program is always called as a subroutine of the IMS region controller. It also has to have a program specification block (PSB) associated with it. The PSB provides an interface from the program to IMS services which the program needs to make use of. These services can be:

- Sending or receiving messages from online user terminals
- Accessing database records
- Issuing IMS commands
- Issuing IMS service calls
 - Checkpoint calls
 - Sync call

The IMS services available to any program are determined by the IMS environment in which the application is running.

16.2 Program structure

During initialization, both the application program and its associated PSB are loaded from their respective libraries by the IMS system. The IMS modules interpret and execute database CALL requests issued by the program. These modules may reside in the same or different MVS address spaces depending on the environment in which the application program is executing.

Application programs executing in an online transaction environment are executed in a dependent region called the message processing region (MPR). The programs are often called message processing programs (MPP). The IMS modules which execute online services will execute in the control region (CTL) while the database services will execute in the DLI separate address space (DLISAS). The association of the application program and the PSB is defined at IMS system generation time via the APPLTN and TRANSACTION macros.

Batch application programs can execute in two different types of regions.

1. Application programs which need to make use of message processing services or databases being used by online systems are executed in a batch message processing region (BMP).
2. Application programs which can execute without messages services execute in a DLI batch region.

For both these types of batch application programs, the association of the application program to the PSB is done on the PARM keyword on the EXEC statement.

The application program interfaces with IMS via the following program elements:

- An ENTRY statement specifying the PCBs utilized by the program
- A PCB-mask which corresponds to the information maintained in the pre-constructed PCB and which receives return information from IMS
- An I/O for passing data segments to and from the databases
- Calls to DL/I specifying processing functions
- A termination statement

The PCB mask (s) and I/O areas are described in the program's data declaration portion. Program entry, calls to IMS processing, and program termination are described in the program's procedural portion. Calls to IMS, processing statements, and program termination may reference PCB mask(s) and/or I/O areas. In addition, IMS may reference these data areas. Figure 42 illustrates how these elements are functionally structured in a program and how they relate to IMS. The elements are discussed in the text that follows:

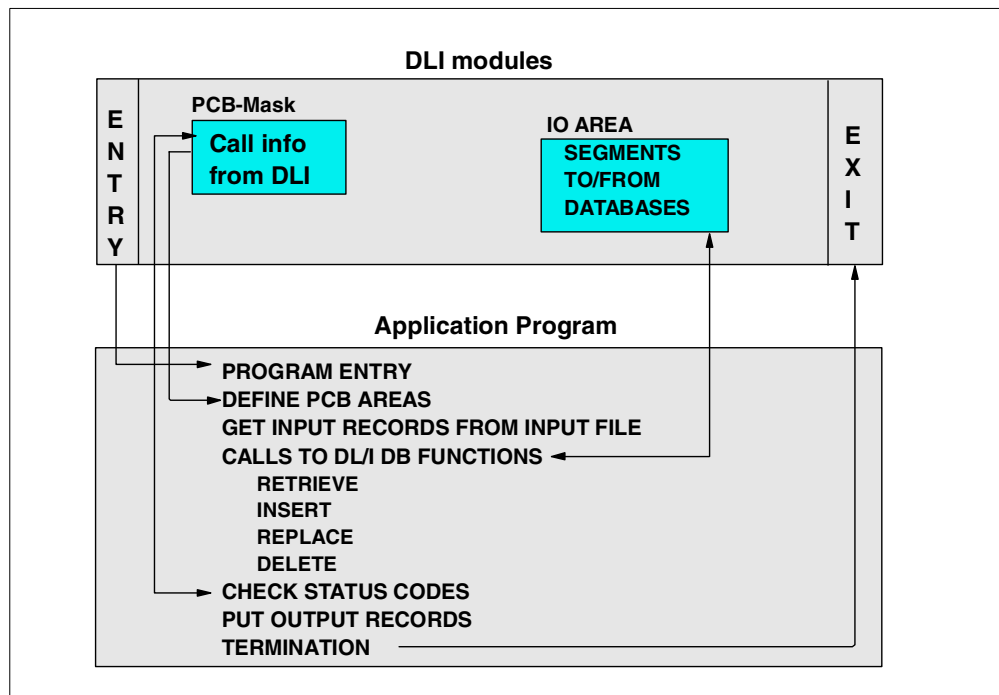


Figure 42. Structure of an application program

16.2.1 Entry to application program

Referring to Figure 42, when the operating system gives control to the IMS control facility, the IMS control program in turn passes control to the application program (through the entry point as defined below). At entry, all the PCB-names used by the application program are specified. The order of the PCB-names in the entry statement must be the same as in the PSB for this application program. The sequence of PCBs in the linkage section or declaration portion of the application program need not be the same as in the entry statement.

Notes:

1. Batch DL/I programs cannot be passed parameter information via the PARM field from the EXEC statement.
2. Online PCBs must proceed database PCBs in the PSB.

16.2.2 Termination

At the end of the processing of the application program, control must be returned to the IMS control program. Table 7 shows examples of the termination statements.

Table 7. Program return statements

Language	Return Statement
COBOL	GOBACK.
PL/I	RETURN;
ASSEMBLER	RETURN(14,12),RC=0

Warning: Since IMS links to your application program, return to IMS causes storage occupied by your program to be released. Therefore you should close all non-DL/I data sets for COBOL and Assembler before return, to prevent abnormal termination during close processing by MVS. PL/I automatically causes all files to be closed upon return.

16.2.3 Calls to IMS

Actual processing of IMS messages, commands, databases and services are accomplished using a set of input/output functional call requests. A call request is composed of a CALL statement with an argument list. The argument list will vary depending on the type of call to be made. The argument list will consist of the following parameters:

- Function call
- PCB name
- IOAREA
- Segment search argument (SSA) (database calls only)

Table 8 shows a brief explanation of the argument list items. The argument list items for database processing are discussed in more detail in Chapter 19, "Application coding for IMS Database Manager" on page 173. The online services and commands argument list items are discussed in more detail in the Chapter 17, "Application coding for IMS Transaction Manager" on page 147.

Table 8. IMS call argument list

Component	Description
Function	Identifies the DL/I function to be performed. This argument is the name of the four character field which describes I/O operation. The DL/I functions are described in the individual chapters.
PCB-name	Is the name of the database program communication block (PCB). It is the name of the PCB within the PSB that identifies which specific data structure the application program wishes to process. The PCB is defined in more detail in "PCB mask" on page 136
I/O Area	Is the name of a I/O work area. This is an area of the application program into which DL/I puts a requested segment, or from which DL/I takes a designed segment. If this a common area is used to process multiple calls it must be long enough to hold the longest path of segments to be processed.
SSA1,... SSAn	Are the names of the Segment Search Arguments (SSA). These are optional depending on the type of call issued.
IOAREA	An area of storage defined in the program for the call to use and input or output. In the case of a database call the segments would be written from or retrieved into this area.
Segment Search Argument	This is only used for database calls. It provides information to define the segment to be retrieved or written.

16.2.4 PCB mask

A mask or skeleton database PCB must provide in the application program. One PCB is required for each view of a database or online service. The program views a hierarchical data structure via this mask.

One PCB is required for each data structure. A database PCB mask is shown in Figure 45. An online PCB is shown in Figure 44.

As the PCB does not actually reside in the application program, care must be taken to define the PCB mask as an assembler dsect, a COBOL linkage section entry, or a PL/I based variable.

The PCB provides specific areas used by IMS to inform the application program of the results of its calls. At execution time, all PCB entries are controlled by IMS. Access to the PCB entries by the application program is for read-only purposes. The PCB masks for an online PCB and a database PCB are different. An example of both are shown in Figure 43.

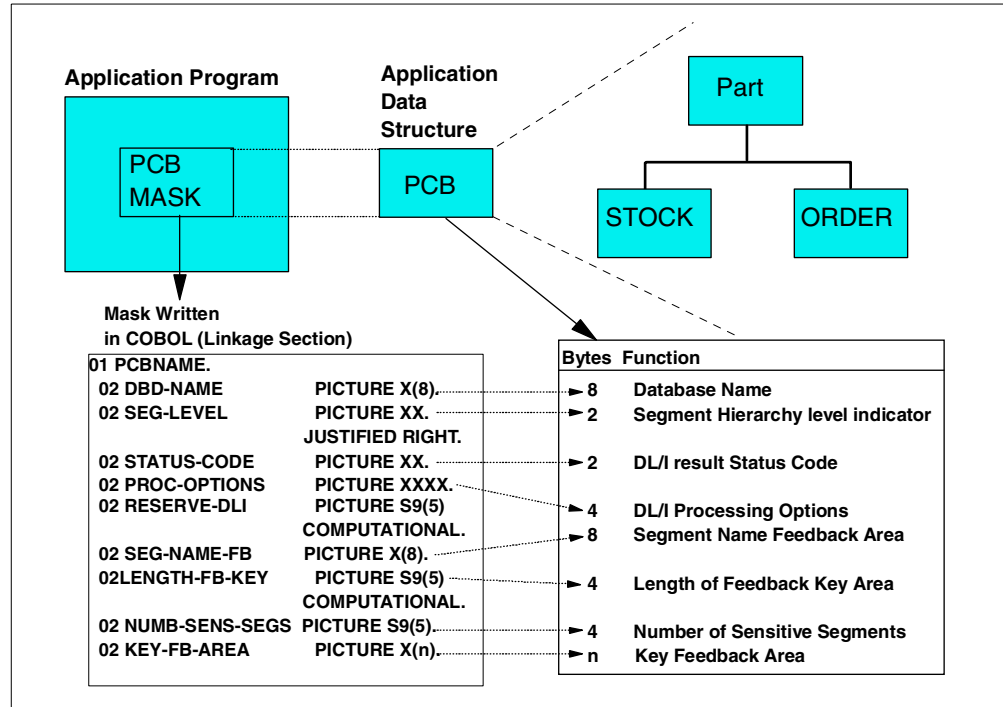


Figure 43. Application PSB structure

16.2.4.1 Online PCB mask

Figure 44 shows an example of an online program's PCB mask, which defines the PCB area used by IMS to return the results of the call.

01	PCBNAME.	
02	DBD-NAME	PICTURE X(8) .
02	SEG-LEVEL	PICTURE XX.
02	STATUS-CODE	PICTURE XX.
02	PROC-OPTIONS	PICTURE XXXX.
02	RESERVED-DLI	PICTURE S9(5) .
02	SEG-NAME	PICTURE X(8) .
02	LENGTH-FB-KEY	PICTURE S9(5) .
02	NUMB-SENS-SEGS	PICTURE S9(5) .
02	KEY-FB-AREA	PICTURE X(n) .

Figure 44. On-line application PCB mask

16.2.4.2 Database PCB mask

Figure 45 shows an example of a DLI program's PCB mask, which defines the PCB area used by IMS to return the results of the call.

01	PCBNAME.	
02	DBD-NAME	PICTURE X(8) .
02	SEG-LEVEL	PICTURE XX.
02	STATUS-CODE	PICTURE XX.
02	PROC-OPTIONS	PICTURE XXXX.
02	RESERVED-DLI	PICTURE S9(5) .
02	SEG-NAME	PICTURE X(8) .
02	LENGTH-FB-KEY	PICTURE S9(5) .
02	NUMB-SENS-SEGS	PICTURE S9(5) .
02	KEY-FB-AREA	PICTURE X(n) .

Figure 45. DLI application PCB mask

The following items comprise a PCB for a hierarchical data structure from a database.

1. Name of the PCB — This is the name of the area which refers to the entire structure of PCB fields. It is used in program statements. This name is not a field in the PCB. It is the 01 level name in the COBOL mask in Figure 45.
2. Name of the database — This is the first field in the PCB and provides the DBD name from the library of database descriptions associated with a particular database. It contains character data and is eight bytes long.
3. Segment hierarchy level indicator — IMS uses this area to identify the level number of the last segment encountered which satisfied a level of the call. When a retrieve is successfully completed, the level number of the retrieved segment is placed here. If the retrieve is unsuccessful, the level number returned is that of the last segment that satisfied the search criteria along the path from the root (the root segment level being '01') to the desired segment. If the call is completely unsatisfied, the level returned is '00'. This field contains character data: it is two bytes long and is a right-justified numeric value.
4. DL/I status code — A status code indicating the results of the DL/I call is placed in this field and remains here until another DL/I call uses this PCB. This field contains two bytes of character data. When a successful call is executed, DL/I sets this field to blanks or to an informative status indication. A complete list of DL/I status codes can be found in the IMS application programming manuals sextets.
5. DL/I proccession options — This area contains a character code which tells DL/I the "processing intent" of the program against this database (that is, the kinds of calls that may be used by the program for processing data in this database). This field is four bytes long. It is left-justified. It does not change from call to call. It gives the default value coded in the PCB PROCOPT parameter, although this value may be different for each segment. DL/I will not allow the application to change this field, nor any other field in the PCB.
6. Reserved area for IMS — IMS uses this area for its own internal linkage related to an application program. This field is one fullword (4 bytes), binary.

7. Segment name feedback area — IMS fills this area with the name of the last segment encountered which satisfied a level of the call. When a retrieve call is successful, the name of the retrieved segment is placed here. If a retrieve is unsuccessful, the name returned is that of the last segment, along the path to the desired segment, that satisfied the search criteria. This field contains eight bytes of character data. This field may be useful in GN calls. If the status code is 'AI' (data management open error), the DD name of the related data set is returned in this area.
8. Length of key feedback area — This entry specifies the current active length of the key feedback area described below. This field is one fullword (4 bytes), binary.
9. Number of sensitive segments — This entry specifies the number of segment types in the database to which the application program is sensitive. This would represent a count of the number of segments in the logical data structure viewed through this PCB. This field is one fullword (4 bytes), binary.
10. Key feedback area — IMS places in this area the concatenated key of the last segment encountered which satisfied a level of the call. When a retrieve is successful, the key of the requested segment and the key field of each segment along the path to the requested segment are concatenated and placed in this area. The key fields are positioned from left to right, beginning with the root segment key and following the hierarchical path. When a retrieve is unsuccessful, the keys of all segments along the path to the requested segment, for which the search was successful, are placed in this area. Segments without sequence fields are not represented in this area.

Note: This area is never cleared, so it should not be used after a completely unsuccessful call. It will contain information from a previous call. See Figure 46 for an illustration of concatenated keys.

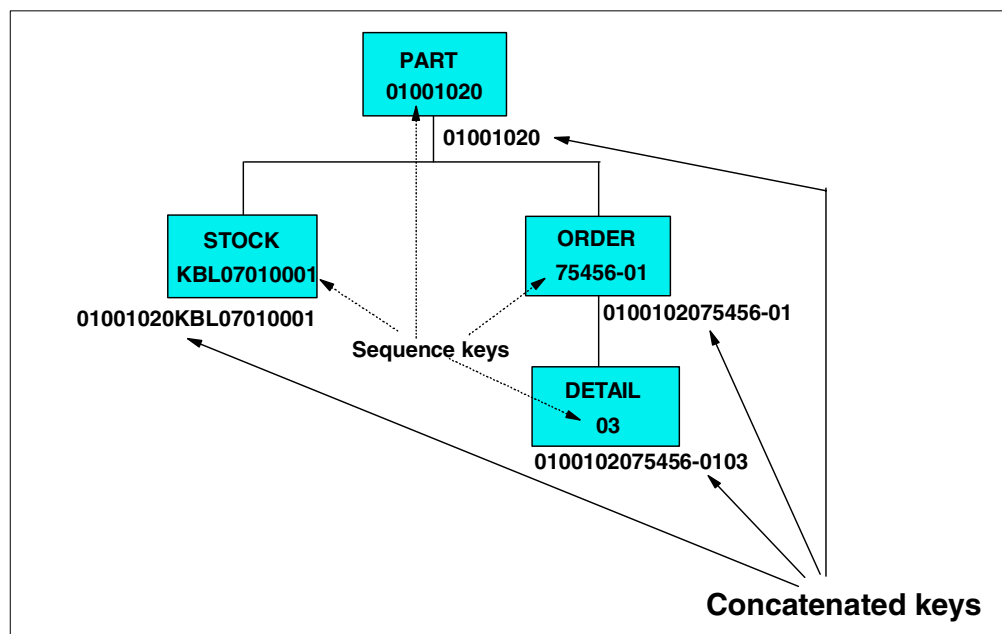


Figure 46. Concatenated keys

16.2.5 Status code handling

After each IMS call, a two-byte status code is returned in the PCB which is used for that call. We distinguish between three categories of status code:

- The blank status code, indicating a successful call
- Exceptional conditions and warning status codes from an application point of view
- Error status codes, specifying an error condition in the application program and/or IMS

The grouping of status codes in the above categories is somewhat installation dependent. We will, however, give a basic recommendation after each specific call function discussion. It is also recommended that you use a standard procedure for status code checking and the handling of error status code. The first two categories should be handled by the application program after each single call. Figure 47 gives an example using COBOL.

```
CALL 'CBLTDLI' USING ....  
IF PCB-STATUS EQ 'GE' PERFORM PRINT-NOT-FOUND.  
IF PCB STATUS NE 'bb' PERFORM STATUS-ERROR.  
everything okay, proceed....
```

Figure 47. Testing Status Codes

Notice that it is more convenient to directly test the regular exceptions in-line instead of branching to a status code check routine. In this way, you clearly see the processing of conditions that you wish to handle from an application point of view, leaving the real error situations to central status code error routine.

16.2.6 IMS control blocks

Before you execute an application program, a program specification block generation (PSBGEN) must be performed to create the program specification block (PSB) for the program. The PSB contains one PCB for each DL/I database (logical or physical) the application program will access. The PCBs specify which segments the program will use and the kind of access (retrieve, update, insert, delete) the program is authorized to. The PSBs are maintained in one or more IMS system libraries called a PSBLIB library.

All IMS databases require a database descriptor block (DBD) created to have access to any IMS databases. The details of these control blocks are describe in 16.2.7, "Generation of IMS control blocks" on page 141. The database DBD is assembled into a system library called a DBDLIB.

The IMS system needs to combine the PSB and DBD control blocks for an application program into a access control blocks (ACB) to be executable. This combination is called a ACB gen. In a batch DLI environment the ACB gen is done dynamically at step initialization time. In an online environment the ACB gen needs to be done before an application can be executed. The ACB gen is run offline and the resulting control blocks are placed in an ACB library.

The IMS system needs to access these control blocks (DBDs and PSBs) in order to define the applications use of the varies IMS resources required. Depending on which environment the application program is executed in will determine how IMS accesses those control blocks. See Figure 48 to see a overview of the processing.

16.2.7 Generation of IMS control blocks

In addition to database PCBs, a PSB for MPP/BMP contains one or more data communication PCBs.

- The order of the PCBs in the PSB must be:
 1. Data communication PCB's
 2. Database PCBs,
 3. GSAM PCBs (not allowed for MPPs)
- One data communication PCB is always automatically included by IMS at the beginning of each PSB of an MPP or BMP. This default data communication PSB is used to insert output messages back to the originating LTERM or USERID.

Note: By use of the COMPAT=YES keyword on the batch PSBGEN statement, we already provided this to PCB to the batch program. In this way, a batch program can be run as a BMP without change. The relative positions of the database PCBs remain the same.

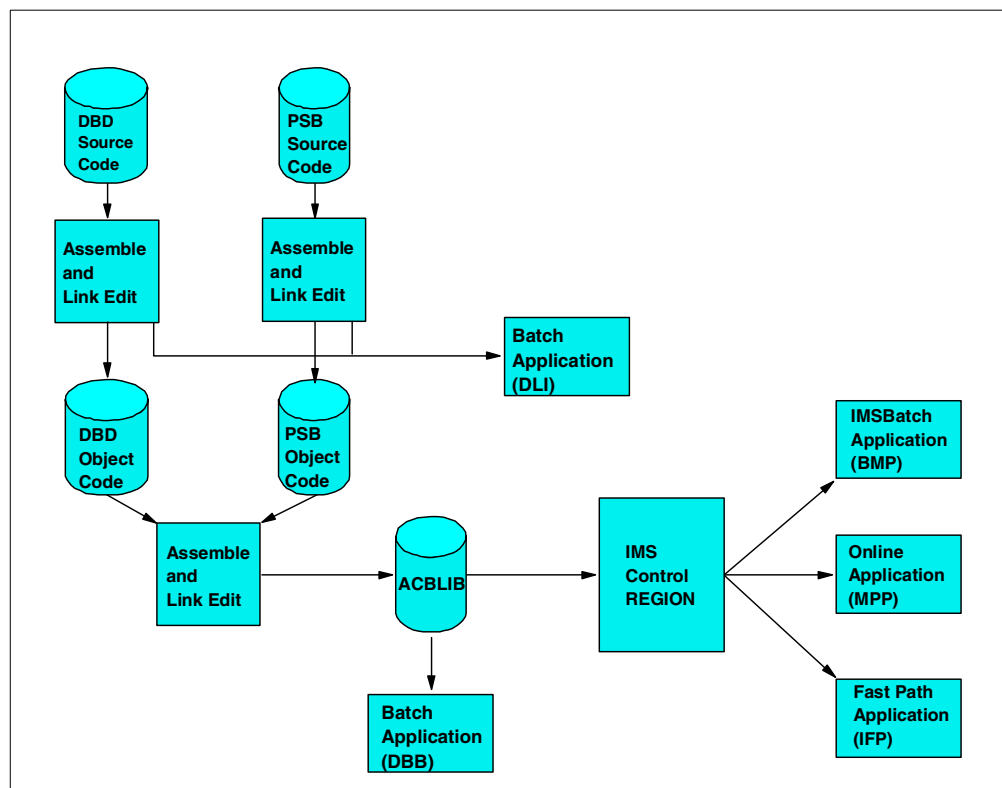


Figure 48. IMS control block generation

Note: Multiple BUILD statements can be coded for both DBDs and PSBs, but the ones for DBDs must be first.

16.3 The IMS database application programming interface (API)

IMS provides a standard set of functions to allow applications to access and manipulate data managed by the IMS Database Manager. These functions also allow applications to access and process messages managed by the IMS Transaction Manager and to perform certain system functions.

Calls to these functions can be made in a number of ways:

- A language specific call interface. There is one for each programming language that IMS applications can be written in.
- A language independent call interface for applications written in any language that supports IBM's language environment product.
- The application interface block (AIB) call interface.
- For CICS applications that access IMS databases, the application can use the CICS command level interface to provide IMS/DB support.
- REXX EXEC's can invoke IMS functions via the IMS adaptor for REXX.

16.3.1 Get unique (GU)

The GU (get unique) call is used to retrieve a specific segment or path of segments from a database. At the same time it establishes a position in a database from which additional segments can be processed in a forward direction.

16.3.2 Get next (GN)

The GN (get next) call is used to retrieve the next or path of segments from the database. The get next call normally moves forward in the hierarchy of a database from the current position. It can be modified to start at an earlier position than current position in the database through a command code, but its normal function is to move forward from a given segment to the next desired segment in a database.

16.3.3 Hold form of get calls

GHU (get hold unique), or GHN (get hold next), indicates the intent of the user to issue a subsequent delete or replace call. A get hold call must be issued to retrieve the segment before issuing a delete or replace call.

16.3.4 Insert

The ISRT (insert) call is used to insert a segment or a path of segments into a database. It is used to initially load segments in databases, and to add segments in existing databases.

To control where occurrences of a segment type are inserted into a database, the user normally defines a unique sequence field in each segment. When a unique sequence field is defined in a root segment type, the sequence field of each occurrence of the root segment type must contain a unique value. When defined for a dependent segment type, the sequence field of each occurrence under a given physical parent must contain a unique value. If no sequence field is defined, a new occurrence is inserted after the last existing one.

16.3.5 Delete

The DLET (delete) call is used to delete a segment from a database. When a segment is deleted from a DL/I database, its physical dependents, if any are also deleted.

16.3.6 Replace

The REPL (replace) call is used to replace the data in the data portion of a segment or path of segments in a database. Sequence fields cannot be changed with a replace call.

16.3.7 System service calls

In addition to the functions above, used to manipulate the data, there are a number of system service calls provided to allow the application to make use of other facilities provided by the IMS Database Manager, such as checkpoint/restart processing (CHKP, XRST, ROLB, ROLL, ROLS), described below.

Refer to the sections on writing DL/I calls for database management and system services in *IMS/ESA Application Programming: Database Manager*, SC26-8015 for full details of available functions.

16.4 The data communication PCB

Besides the default data communication PCB, which does not require PCB statement, additional PCBs can be coded. These PCBs are used to insert output messages to:

- LTERMs other than the LTERM which originated the input message. A typical use of an alternate PCB is to send output to a 3270 printer terminal.
- A non-conversational transaction.
- Another USERID.

The destination of the output LTERM can be set in two ways:

- During PSBGEN by specifying the LTERM/TRANNAME in a alternate PCB.
- Dynamically by the MPP during execution, by using a change call against a modifiable alternate PCB.

The method used depends on the PCB statement.

16.4.0.1 The PCB statement

This is the only statement required to generate an alternate PCB (multiple occurrences are allowed). Its format is:

```
PCB TYPE=TP,LTERM=name,MODIFY=YES
```

See Table 9 for a description of the parameters.

Table 9. PCB statement

Keyword	Description
TYPE=TP	Is required for all alternative PCBs.
LTERM=name	Specifies this PCB is pointing at a know LTERM defined in the IMS system. The name is optional.
MODIFY=YES	<p>If the modify is specified then the LTERM name may be changed by a CHANGE call within the application program.</p> <p>Note: If MODIFY=YES is specified, the MPP must specify a valid alternate output LTERM with a change call before inserting any message via this PCB.</p>

16.4.1 The database PCB

The database PCB for an MPP or BPP is basically the same discussed above. Two additional processing intent options can be specified with the PROCOPT=keyword of the PCB and/or SENSEG statement.

16.4.2 Additional processing intent options

The PROCOPT=keyword is extended with two additional processing intent options, "O" AND "E".

Their meanings are:

O — Read only: no dynamic enqueue is done by program isolation for calls against this database. Can be specified with only the G intent option, as GO or GOP. This option is only valid for the PCB statement.

E — Forces exclusive use of this database or segment by the MPP/BMP. No other program which references this database/segment will be scheduled in parallel. No dynamic enqueue by program isolation is done, but dynamic logging of database updates will be done. E can be specified with G, I, D, B, and A.

CAUTION: If the 'O' option (read-only) is used for a PCB, IMS does not check the ownership of the segments returned. This means that the read-only user might get a segment that had been updated by another user. If the updating user should then abnormal terminate, and he backed out, the read-only user would have a segment that did not (and never did) exist in the database. Therefore, the 'O' option user should not perform updates based on data read with that option. An ABEND can occur with PROCOPT=GO if another program updates pointers when this program is following the pointers. Pointers are updated during insert, delete and backout functions.

16.4.2.1 The PSBGEN statement

This is basically the same as for a database PCB. The IOEROPN= parameter must be omitted, the COMPAT=YES parameter is ignored.

16.4.3 Application control block generation (ACBGEN)

Before PSBs and DBDs can be used by the CTL region, they must be expanded to an internal control block format. This expansion is done by the application control block generation (ACBGEN) utility. The expended control blocks are maintained in the IMSVS.ACBLIB. This is a standard MVS partitioned data set. JCL Requirements

An ACBGEN procedure is placed in IMSVS.PROCLIB during IMS system definition.

Note: Multiple BUILD statements can be coded for both DBDs and PSBs, but the ones for DBDs must be first.

16.4.4 IMS/DB2 resource translate table (RTT) assembly

When an IMS transaction accesses DB2, the plan name used is, by default, the same as the PSB/APPLCTN name.

It is, however, possible to set up a translation table, the RTT, that translates an APPLCTN to a different DB2 plan name.

This is described in the DB2 (not IMS) documentation for attaching DB2 to IMS. Refer to Defining DB2 Plans for IMS Applications in *DB2 for OS/390 V5 Installation Guide*, GC26-8970. It is simply a table of macros, associating APPLCTNs with DB2 plan names. This is assembled in a CSECT (with the name the same as the label of the 1st macro in the table). This must then be placed in an APF authorized library in the RESLIB concatenation of the IMS control region. The RTT is pointed to in the PROCLIB member that defines the DB2 attachment. If the RTT parameter is null, the RTT is not used.

The re-assembled table will be picked up the next time IMS is stopped/started or when a stop (/STO SUBSYS xxxx) and restart (/STA SUBSYS xxxx) of the DB2 connection.

Chapter 17. Application coding for IMS Transaction Manager

This chapter, which deals with writing application programs in the IMS Transaction Manager environment, is divided into two major sections:

- Basic application processing requirements
- Designing application programs in an IMS online environment

17.1 Application Program Processing

Basically, the MPP processing can be divided into five phases. See Figure 49.

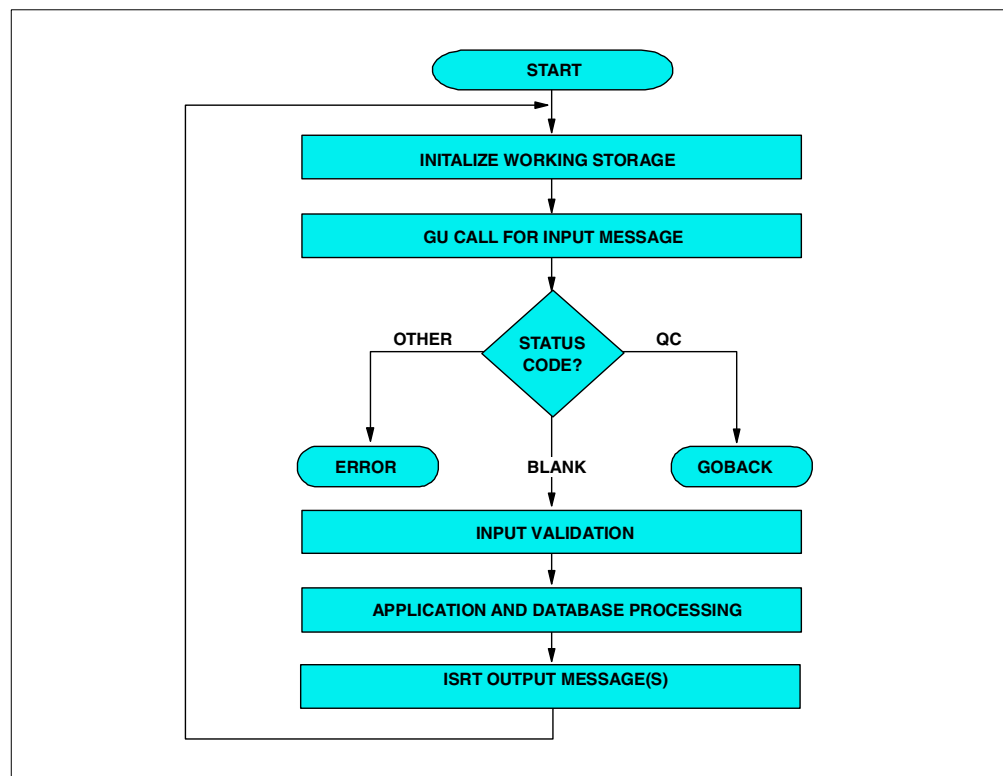


Figure 49. General MPP Structure and Flow

1. Initialization. The clearing of working storage, which may contain data left-over by the processing of a message from another terminal.
2. Retrieval of the SPA (optional) and the input message.
3. Input syntax check. All checks which can be done without accessing the database, including a consistency check with the status of the conversation as maintained in the SPA.
4. Database processing, preferably in one phase. This means that the retrieval of a database segment is immediately followed by its update. Compare this to an initial retrieve of all required segments followed by a second retrieve and then update.
5. Output processing. The output message is built and inserted together with the SPA (only for conversational transactions).

Note: After finishing the processing of one input message, the program should go back to step 1 and request a new input message. If there are no more input messages, IMS will return a status code indicating that. At that time, the MPP must return control to IMS.

17.1.1 Role of the PSB

The program specification block (PSB) for an MPP or a BMP contains, besides database PCBs, one or more PCB (s) for logical terminal linkage. The very first PCB always identifies the originating logical terminal. This PCB must be referenced in the get unique and get next message calls. It must also be used when inserting output messages to that LTERM. In addition, one or more alternate output PCBs can be defined. Their LTERM destinations can be defined in the PCBs or set dynamically with change destination calls.

17.1.2 DL/I message calls

The same DL/I language interface which is used for the access of databases is used to access the message queues.

The principal DL/I message call function codes are:

- GU, get unique. This call must be used to retrieve the first, or only, segment of the input message.
- GN, get next. This call must be used to retrieve second and subsequent message segments.
- ISRT, insert. This call must be used to insert an output message segment into the output message queue. Note: these output message (s) will not be sent until the MPP terminates or requests another input message via a get unique.
- CHNG. change destination. This call can be used to set the output destination for subsequent insert calls.

For a detailed description of the DL/I database calls and guidelines for their use, see Chapter 19, "Application coding for IMS Database Manager" on page 173.

17.1.3 Application program abnormal termination

Upon abnormal termination of a message or batch-message processing application program for other reasons than deadlock resolution, internal commands are issued to prevent rescheduling. These commands are the equivalent of a /STOP command. They prevent continued use of the program and the transaction code in process at the time of abnormal termination. The master terminal operator can restart either or both stopped resources. At the time abnormal termination occurs, a message is used to the master terminal and to the input terminal that identifies the application program, transaction code, and input terminal. It also contains the system and user completion codes. In addition, the first segment of the input transaction, in process by the application at abnormal termination, is displayed on the master terminal. The database changes of a failing program are dynamically backed-out. Also, its output messages inserted in the message queue since the last synchronization point are cancelled.

17.1.4 Conversational processing

A transaction code can be defined as belonging to a conversational transaction during IMS system definition. If so, an application program that processes that transaction, can interrelate messages from a given terminal. The vehicle to accomplish this is the scratchpad area (SPA). A unique scratchpad area is created for each physical terminal which starts a conversational transaction. Each time an input message is entered from a physical terminal in conversational mode, its SPA is presented to the application program as the first message segment (the actual input being the second segment). Before terminating or retrieving another message (from another terminal), the program must return the SPA to the control region with a message ISRT call. The first time a SPA is presented to the application program when a conversational transaction is started from a terminal, IMS will format the SPA with binary zero's (X'00). If the program wishes to terminate the conversation, it can indicate this by inserting the SPA with a blank transaction code.

17.1.5 Output message processing

As soon as an application reaches a synchronization point, its output messages in the message queue become eligible for output processing. A synchronization point is reached whenever the application program terminates or requests a new message from the input queue via a GU call.

In general, output messages are processed by message format service before they are transmitted via the telecommunications access method.

Different output queues can exist for a given LTERM, depending on the message origin. They are, in transmission priority:

1. Response messages, that is, messages generated as a direct response (same PCB) to an input message from this terminal.
2. Command responses.
3. Alternate output messages, messages generated via an alternate PCB.

17.1.6 Logging and checkpoint/restart

To ensure the integrity of its databases and message processing IMS uses logging and checkpoint/restart. In case of system failure, either software or hardware, IMS can be restarted. This restart includes the repositioning of users' terminals, transactions, and databases.

17.1.6.1 Logging

During IMS execution all information necessary to restart the system in the event of hardware or software failure, is recorded on a online log data sets (OLDS).

The following critical system information is recorded on the OLDS:

- The receipt of an input message in the input queue
- The start of an MPP/BMP
- The receipt of a message by the MPP for processing
- Before and after images of database updates by the MPP/BMP
- The insert of a message into the queue by the MPP

- The termination of an MPP/BMP
- The successful receipt of an output message by the terminal

In addition to the above logging, all previous database record unchanged data is written to the log data set. This log information is only used for dynamic back-out processing of a failing MPP/BMP. as soon as the MPP/BMP reaches a synchronization point, the dynamic log information of this program is discarded.

17.1.6.2 Emergency restart

In case of failure, IMS is restarted with the log data set active at the time of failure. Restart processing will back-out the database changes of incomplete MPPs and BMPs. The output messages inserted by these incomplete MPPs will be deleted.

After back-out, the input messages are re-enqueued, the MPPs restarted and the pending output messages are (re) transmitted. If a BMP was active at the time of failure, it must be resubmitted via MVS job management. If the BMP uses the XRST/CHKP calls, it must be restarted from its last successful checkpoint. In this way missing or inconsistent output is avoided.

17.2 The data communication design process

We will distinguish between the following areas in the IMS database/data communication design process:

- Program design
- Message format service design
- Database design

In the program design section, we will concentrate on the design of message processing programs (MPPs).

The MFS design will discuss the 3270 screen layouts and operator interaction.

Although we will cover each of the above areas in separate sections, it should be realized that they are largely dependent upon each other. Therefore, an overall system design must be performed initially and an overall system review must follow the design phase of each section.

17.2.1 Concepts of online transaction processing

In an IMS online environment, one can view a transaction from three different points:

- The application, that is, its processing characteristics and database accesses.
- The terminal user.
- The IMS system.

Each of the above constitutes a set of characteristics. A description of each set follows.

17.2.2 Application characteristics

From an application point of view, we can identify:

- Data collection with no previous database access). This is not a typical IMS application but can be part of an IMS application system.
- Update. This normally involves database reference and the subsequent updating of the database. This is the environment of most IMS applications.

In typical IMS multi-application environment, the above characteristics are often combined. However, a single transaction normally has only one of the above characteristics.

17.2.3 Terminal user characteristics

From the terminal user's point of view, we distinguish:

- Single-interaction transactions.
- Multi-interaction transactions.

The single interaction transaction does not impose any dependency between any input message and its corresponding output, and the next input message. The multi-interaction transaction constitutes a dialogue between the terminal and the message processing program (s). Both the terminal user and the message processing rely on a previous interaction for the interpretation/processing of a subsequent interaction.

17.2.4 IMS characteristics

From the IMS system point of view, we distinguish:

- Non-response transactions.
- Response transactions.
- Conversational transactions.

Note: These IMS transaction characteristics are defined for each transaction during IMS system definition.

With non-response transactions, IMS accepts multiple input messages (each being a transaction) from a terminal without a need for the terminal to first accept the corresponding output message, if any. These non-response transactions will not be further considered in our sample.

With response transactions, IMS will not accept further transaction input from the terminal before the corresponding output message is sent and interpreted by the user.

For conversational transactions, which are always response transactions IMS provides a unique scratch pad area (SPA) for each user to store vital information across successive input messages.

17.2.5 Transaction response time considerations

In addition to the above characteristics, the transaction response time is often an important factor in the design of online systems. The response time is the elapsed time between the entering of an input message by the terminal operator and the receipt of the corresponding output message at the terminal.

Two main factors, in general, constitute the response time:

1. The telecommunication transmission time, which is dependent on such factors as:
 - Terminal network configuration
 - Data communication access method and data communication line procedure
 - Amount of data transmitted, both input and output
 - Data communication line utilization
2. The internal IMS processing time, which is mainly determined by the MPP service time. The MPP service time is the elapsed time required for the processing of the transaction in the MPP region.

17.2.6 Choosing the right characteristics

Each transaction in IMS can and should be categorized by one characteristic of each of the previously discussed three sets.

Some combinations of characteristics are more likely to occur than others, but all of them are valid.

In general, it is the designer's choice as to which combination is attributed to a given transaction. Therefore, it is essential that this selection of characteristics is a deliberate part of the design process, rather than determined after implementation.

Following are some examples:

1. Assume an inquiry for the customer name and address with the customer number as input. The most straightforward way to implement this is clearly a non-conversational response-type transaction.
2. The entry of new customer orders could be done by a single response transaction. The order number, customer number, detail information, part number, quantity etc., could all be entered at the same time. The order would be processed completely with one interaction. This is most efficient for the system, but it may be cumbersome for the terminal user because she or he has to re-enter the complete order in the case of an error.

Quite often, different solutions are available for a single application. Which one to choose should be based on a trade-off between system cost, functions, and user convenience. The following sections will highlight this for the different design areas.

17.2.7 Online program design

This area is second in importance to database design. We will limit the discussion of this broad topic to the typical IMS environment. We will first discuss a number of considerations so that you become familiar with them. Next, we will discuss the design of the two online sample programs. You will notice that some discussions are quite arbitrary and may not have to be adjusted for your own environment. Do remember, however, that our prime objective is to make you aware of the factors which influence these decisions.

17.2.7.1 Single versus multiple passes

A transaction can be handled with one interaction or pass, or with two or more passes (a pass is one message in and one message out). Each pass bears a certain cost in line time and in IMS and MPP processing time. So, in general, you should use a few passes as possible. Whenever possible you should use the current output screen to enter the next input. This is generally easy to accomplish for inquiry transactions, where the lower part of the screen can be used for input and the upper part for output. (See 17.2.8, "Basic screen design" on page 154.

For update transactions, the choice is more difficult. The basic alternatives are:

One-pass update:

After input validation, the database updates are all performed in the same pass. This is the most efficient way from the system point of view. However, correcting errors after the update confirmation is received on the terminal requires additional passes or re-entering of data. An evaluation of the expected error rate is required.

Two-pass update:

On the first pass, the input is validated, including database access. A status message is sent to the terminal. If the terminal operator agrees, the database will be updated in the second pass. With this approach, making corrections is generally much simpler, especially when a scratch pad area is used. However, the database is accessed twice.

You should realize, that, except for the SPA, no correlation exists between successive interactions from a terminal. So, the database can be updated by somebody else and the MPP may process a message for another terminal between two successive passes.

Multi-pass update:

In this case, each pass does a partial database update. The status of the database and screen is maintained in the SPA. This approach should only be taken for complex transactions. Also, remember that the terminal operator experiences response times for each interaction. You also must consider the impact on database integrity. IMS will only back-out the database changes of the current interaction in the case of project or system failure.

Notes:

1. IMS emergency restart with a complete log data set will reposition the conversation. The terminal operator can proceed from the point where he or she was at the time of failure.
2. When a conversational application program terminates abnormally, only the last interaction is backed out.

The application must reposition the conversation after correction. For complex situations, IMS provides an abnormal transaction exit routine. This is not covered in our subset.

17.2.7.2 Conversational versus non-conversational

Conversational transactions are generally more expensive in terms of system cost than non-conversational ones. However, they give better terminal operator service. You should only use conversational transactions when you really need them. Also, with the proper use of MFS, the terminal operator procedures sometimes can be enhanced to almost the level of conversational processing. This will be discussed in the section about MFS Design.

17.2.7.3 Transaction/program grouping

It is the designer's choice how much application function will be implemented by one transaction and/or program. The following considerations apply:

- Inquiry-only. transactions should be simple transactions. These should be normally implemented as non-conversational transactions. Also, they can be defined as "non-recoverable inquiry-only". If in addition, the associated MPPs specify PROCOPT= GO in all their database PCB's, no dynamic enqueue and logging will be done for these transactions.
- Limited-function MPPs are smaller and easier to maintain. However, a very large number of MPPs costs more in terms of IMS resources (control blocks and path lengths).
- Transactions with a long MPP service time (many database accesses). should be handled by separate programs.

Note: IMS provides a program-to-program message switch capability. This is not part of our subset. With this facility, you can split the transaction processing in two (or more) phases. The first (foreground) MPP does the checking and switches a message (and, optionally, the SPA) to a (background) MPP in a lower priority partition which performs the lengthy part of the transaction processing. In this way the foreground MPP is more readily available for servicing other terminals. Also, if no immediate response is required from the background MPP and the SPA is not switched, the terminal is more readily available for entering another transaction.

17.2.8 Basic screen design

Generally, a screen can be divided into five areas, top to bottom:

1. Primary output area, contains general, fixed information for the current transaction. The fields in this area should generally be protected.
2. Detail input/output area, used to enter and/or display the more variable part of the transaction data. Accepted fields should be protected (under program control): fields in error can be displayed with high intensity and unprotected to allow for corrections.
3. MPP error message area. In general, one line is sufficient. This can be the same line as 5 below.
4. Primary input, that is requested action and/or transaction code for next input, and primary database access information.
5. System message field, used by IMS to display system messages and by the terminal operator to enter commands.

For readability, the above areas should be separated by at least one blank line. The above screen layout is a general one, and should be evaluated for each individual application. It is recommended to develop a general screen layout and set of formats to be used by incidental programs and programs in their initial test.

This can significantly reduce the number of format blocks needed and maintenance. In any case, installation standards should be defined for a multi-application environment.

17.2.8.1 MFS subset restriction

1. The maximum output length of a message segment is 1388 bytes: this is related to our long message record length of 1500 bytes.
2. A format is designated for one screen size. This can be later changed via additional MFS statements to support both screens and other devices with the same set of format blocks. A 1920 character format can be displayed on the top part of a 2560 or 3440 character display, and 480 character format can be displayed on the top of a 960 character display.
3. A segment is one physical page, which is one logical page.

17.2.8.2 General screen layout guidelines

The following performance guidelines should be observed when making screen layouts:

1. Avoid full-format operations. IMS knows what format is on the screen. So if the format for the current output is the same as the one on the screen, IMS need not retransmit all the literals and unused fields.
2. Avoid unused fields, for example, undefined areas on the screen. Use the attribute byte (non-displayed) of the next field as a delimiter, or expand a literal with blanks. Each unused field causes additional control characters (5) to be transmitted across the line during a full-format operation.

Note: This has to be weighed against user convenience. For example, our sample customer name inquiry format does not have consecutive fields but it is user convenient. Also, this application rarely needs a new format so we are not so much concerned with unused fields.

17.2.8.3 Including the transaction code in the format

IMS requires a transaction code as the first part of an input message. With MFS, this transaction code can be defined as a literal. In doing so, the terminal operator always enters data on a preformatted screen. The initial format is retrieved with the /FORMAT command. To allow for multiple transaction codes on one format, part of the transaction code can be defined as a literal in the MID. The rest of the transaction code can then be entered via a DFLD. This method is very convenient for the terminal operator because the actual transaction codes are not of his concern. Any example of such a procedure is shown in our sample customer order entry application.

17.2.8.4 Miscellaneous design considerations

The following design considerations should also be noted:

- The conversation will be terminated (insert blank transaction code in SPA) after each successful order entry. This is transparent to the terminal operator, because the output format is linked to a MID which contains the transaction code, so the operator need not re-enter it.
- Each output message should contain all the data (except the MOD-defined literals) to be displayed. You should never rely on already existing data on the screen, because a clear or (re) start operation may have destroyed it.
- Using secondary indexing can significantly increase the accessibility of online databases. Therefore, a wider use of this facility is discussed in 11.5, “Secondary indexing” on page 76.

Chapter 18. IMS message format service

The chapter contains an overview of the message format service. This the function of IMS which describes the screen input and output interaction with IMS online programs.

18.1 Message format service overview

Through the message format service (MFS), a comprehensive facility is provided for IMS users of 3270 and other terminals/devices. MFS allows application programmers to deal with simple logical messages instead of device dependent data. This simplifies application development. The same application program may deal with different device types using a single set of editing logic while device input and output are varied to suit a specific device. The presentation of data on the device or operator input may be changed without changing the application program. Full paging capability is provided for display devices. This allows the application program to write a large amount of data that will be divided into multiple screens for display on the terminal. The capability to page forward and backward to different screens within the message is provided for the terminal operator. The conceptual view of the formatting operations for messages originating from or going to an MFS-supported device is shown in Figure 50.

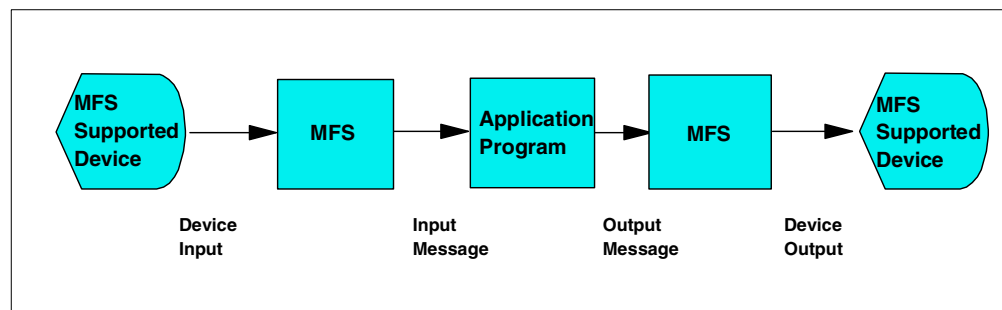


Figure 50. Message formatting using MFS

MFS has three major components

1. MFS language utility
2. MFS pool manger
3. MFS editor

The MFS language utility is executed offline to generate control blocks and place them in a format control block data set named IMSVS.FORMAT. The control blocks describe the message formatting that is to take place during message input or output operations. They are generated according to a set of utility control statements. There are four types of format control blocks:

1. Message input descriptor (MID)
2. Message output descriptor (MOD)
3. Device input format (DIF)
4. Device output format (DOF)

The MID and MOD blocks relate to application program input and output message segment formats, and the DIF and DOF blocks relate to terminal I/O formats. The MID and DIF blocks control the formatting of input messages, while the MOD and DOF blocks control output message formatting.

Notes:

- The DIF and the DOF control blocks are generated as a result of the format (FMT) statement.
- The MID and the MOD are generated as a result of the various message (MSG) statements.
- The initial formatting of a 3270 display is done via the “/FORMAT modname” command. This will format the screen with the specified MOD, as if a null message was sent.

Figure 51 provides an overview of the MFS operations.

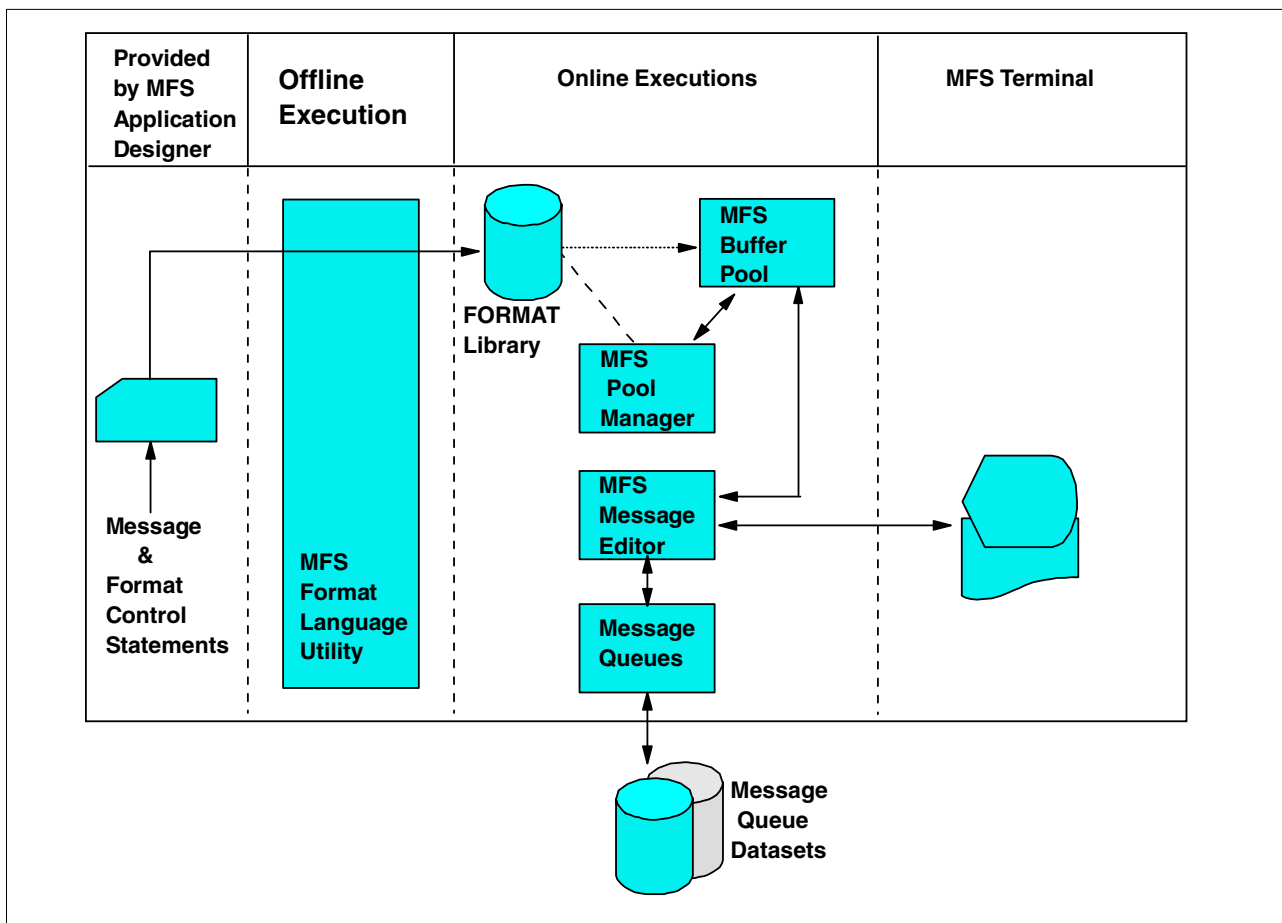


Figure 51. Overview of message format service

18.2 MFS and the 3270

The IMS message format service (MFS), described in the previous section, is always used to format data transmitted between IMS and the devices of the 3270 information display system. MFS provides a high level of device independence for the application programmers and a means for the application system designer to make full use of the 3270 device capabilities in terminal operations. Although our subset only considers the 3270, its use of MFS is such that it is open-ended to the use of other MFS supported terminals when required.

18.3 Relationship between MFS control blocks

Several levels of linkage exist between MFS control blocks, as described in the following sections.

18.3.1 MFS control block chaining

Figure 52 shows the highest-level linkage, that of chained control blocks.

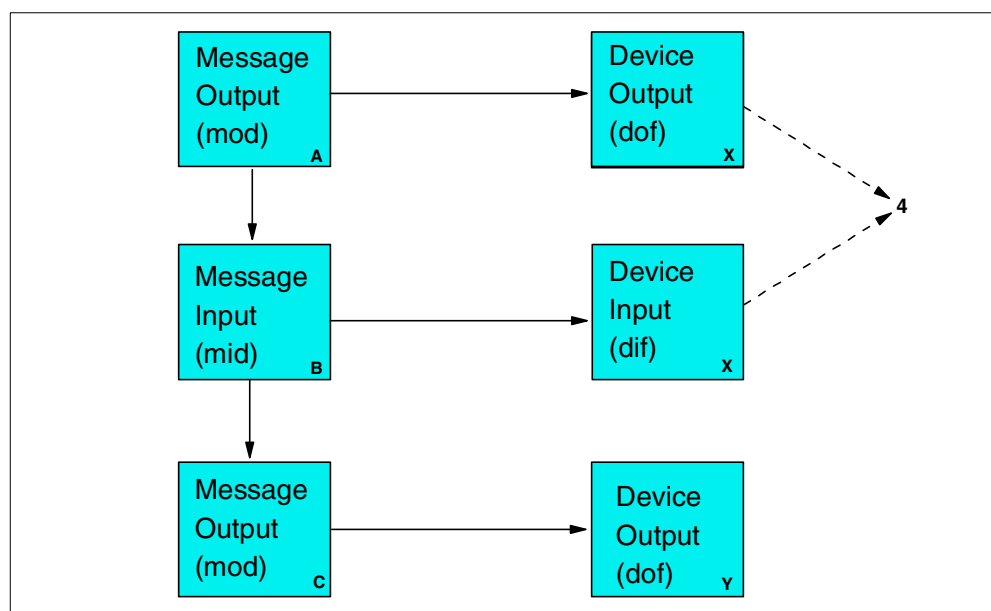


Figure 52. Chained control block linkage

Legend:

1. This linkage must exist.
2. If the linkage does not exist, device input data from 3270 devices is not processed by MPS. It is always used in our subset.
3. This linkage is provided for application program convenience. It provides a MOD name to be used by IMS if the application program does not provide a name via the format name option of the insert call. The default MOD, DFISM02, will be used if none is specified at all, or if the input is a message switch to an MFS-supported terminal.

4. The user-provided names for the DOF and DIF used in one output/input sequence are normally the same. The MFS language utility alters the internal name for the DIF to allow the MFS pool manager to distinguish between the DOF and DIF.

The direction of the linkage allows many message descriptions to use the same device format if desired. One common device format can be used for several application programs whose output and input message formats, as seen at the application program interface, are quite different.

18.3.2 Linkage between DFLD and MFLD

Figure 53 shows the second level of linkage, that between message fields and device fields. The arrows show the direction of reference in the MFS language utility control statements, not the direction of data flow.

References to device fields by message fields need not be in any particular sequence. An MFLD need not refer to any DFLD, in which case it simply defines space in the application program segment to be ignored if the MFLD is for output, and padded if the MFLD is for input. Device fields need not be referenced by message fields, in which case they are established on the device, but no output data from the output message is transmitted to them. Device input data is ignored if the DFLD is not referenced by the input MFLD.

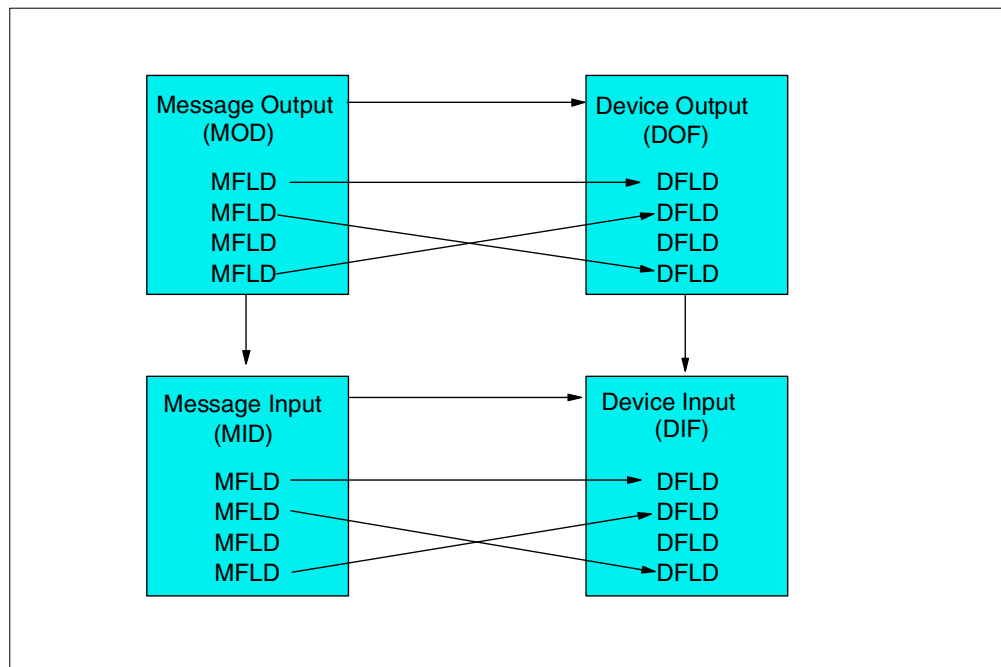


Figure 53. Linkage between message fields and device fields

18.3.3 Linkage between LPAGE and DPAGE

Figure 54 shows a third level of linkage, one which exists between the LPAGE and the DPAGE.

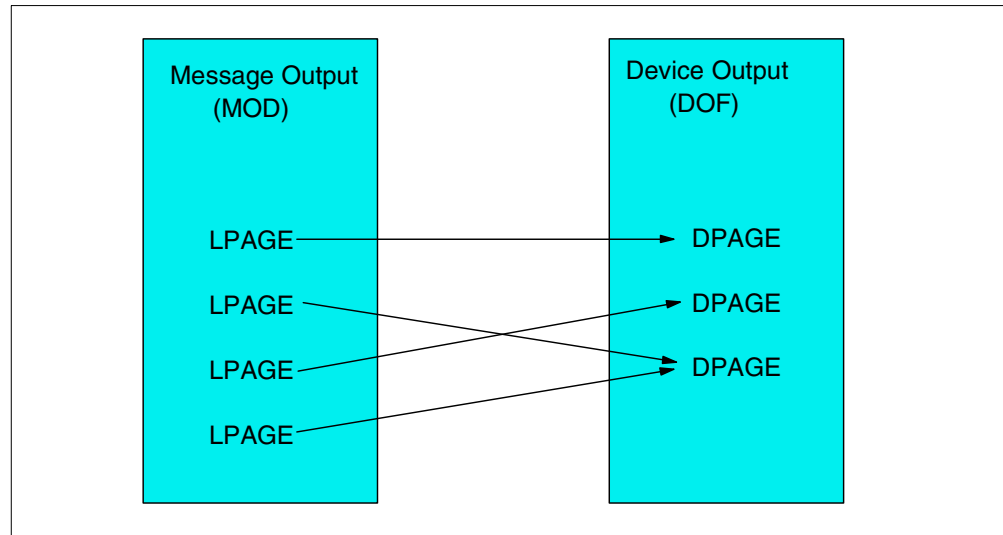


Figure 54. LPAGE -- DPAGE linkage

The LPAGE in the MOD must refer to a DPAGE in the DOF. However, all DPAGES need not be referred to from a given MOD.

Because we will always have single segment input in our subset, the defined MFLDs in the MID may refer to DFLDs in any DPAGE. But input data for any given input message from the device is limited to fields defined in a single DPAGE.

18.3.4 Optional message description linkage

Figure 55 shows a fourth level of linkage. It is optionally available to allow selection of the MID based on which MOD LPAGE is displayed when input data is received from the device.

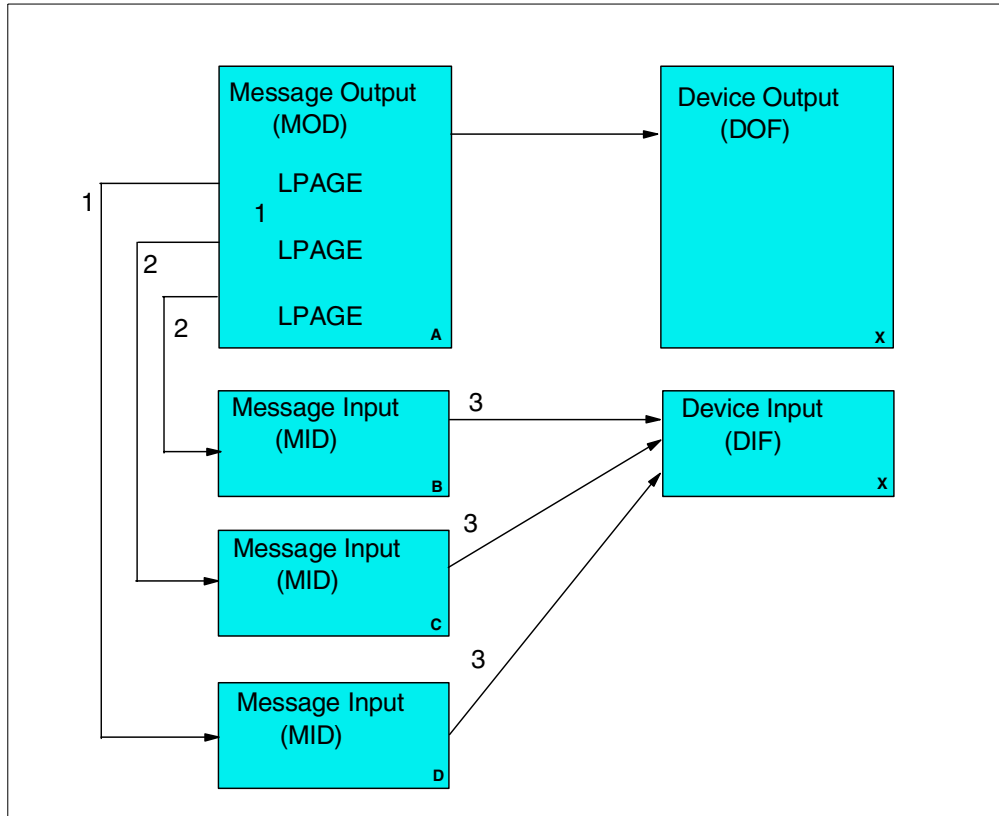


Figure 55. Optional message description linkage

Legend:

1. The next MID name provided with the MSG statement is used if no name is provided with the current LPAGE.
2. If the next MID name is provided with the current LPAGE, input will be processed using this page.
3. For 3270 devices, all MIDs must refer to the same DIF. This is the same user-provided name used to refer to the DOF when the MOD was defined.

18.3.5 3270 Device considerations relative to control blocking linkage

Since output to 3270 display devices establishes fields on the device using hardware capabilities, and field locations cannot be changed by the operator, special linkage restrictions exist. Because formatted input can only occur from a screen formatted by output, the LPAGE and physical page description used for formatting input is always the same as that used to format the previous output. The MPS language utility enforces this restriction by ensuring that the format name used for input editing is the same as the format name used for the previous output editing. Furthermore, if the DIF corresponding to the previous DOF cannot be fetched during online processing, an error message is sent to the 3270 display.

18.4 MFS Functions

The following sections contain a description of the basic MFS functions.

18.4.1 Input message formatting

All device input data received by IMS is edited before being passed to an application program. The editing is performed by either IMS basic edit or MFS. It tells how the use of MFS is determined and how, when MFS is used, the desired message format is established based on the contents of two MFS control blocks — the device input format (DIF) and the message input descriptor (MID).

All 3270 devices included in an IMS system use MFS. The 3270s always operate in formatted mode except when first powered on, after the CLEAR key has been pressed, or when the MOD used to process an output message does not name a MID to be used for the next input data. While in unformatted mode, you can still enter commands and transactions, but they will not be formatted by MFS.

18.4.1.1 Input data formatting using MFS

Input data from terminals in formatted mode is formatted based on the contents of two MFS control blocks, the MID and the DIF. The MID defines how the data should be formatted for presentation to the application program and points to the DIF associated with the input device. See Figure 56.

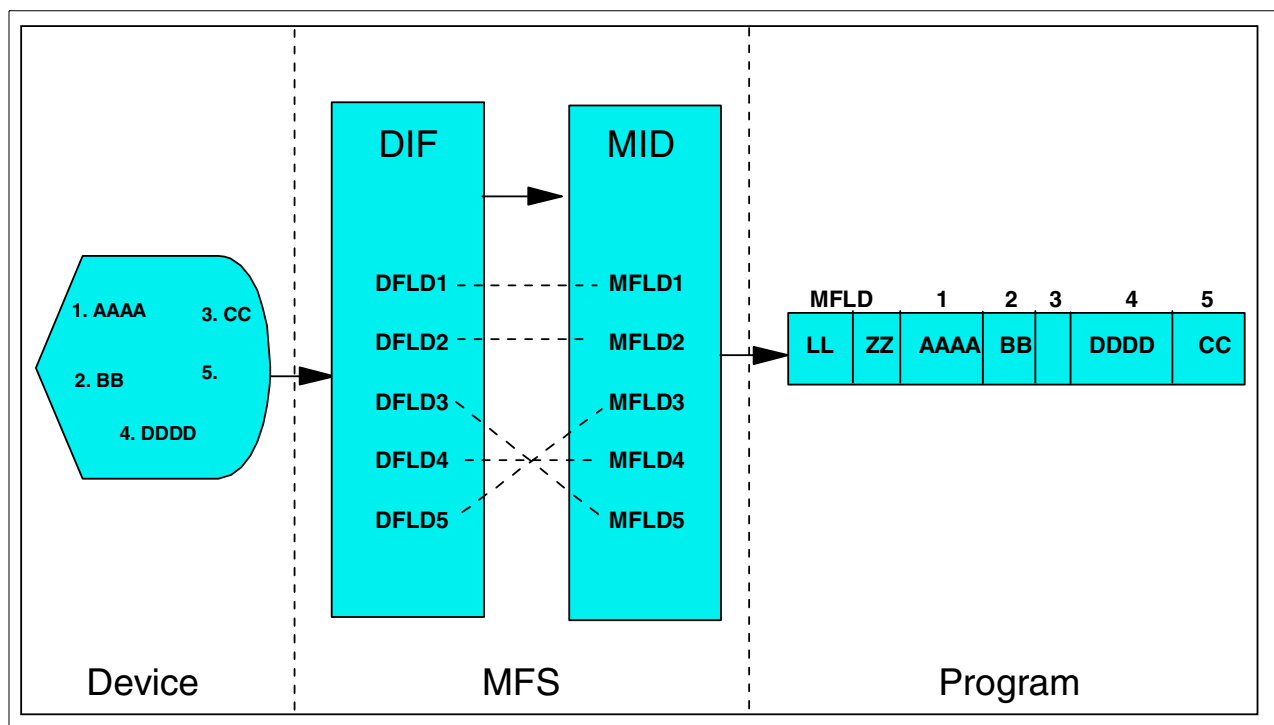


Figure 56. MFS Input Formatting

The MID contains a list of message descriptor fields (MFLDs) which define the layout of the input segment as is to be seen by an application program. The DIF contains a list of device descriptor fields (DFLDs) which define what data is to be expected from which part of the device (that is, the location on the screen). MFS maps the data of the DFLDs into the corresponding MFLDs. The application program is largely device independent because different physical inputs can be mapped into the same input segment.

MFLD statements are to define:

- The device fields (DFLDs) defined in the DIF which contents will be included in the message presented to the application program.
- Constants, defined as literals to be included in the message: a common use of literals is to specify the transaction code.

In addition, the MFLD statement defines:

- The length of the field expected by the application program.
- Left or right justification and the fill character to be used for padding the data received from the device.
- A 'nodata' literal for the MFLD if the corresponding DFLD does not contain any input data.

It should be noted that all message fields as defined by MFLD statements will be presented to the application program in our subset. Furthermore, there will always be only one input message segment, except for conversational transaction, in which case the first segment presented to the program is the SPA. The SPA is never processed by MFS, however.

18.4.1.2 Input message field attribute data

Sometimes input messages are simply updated by an application program and returned to the device. In such a case, it may simplify message definition layouts in the MPP if the attribute data bytes are defined in the message input descriptor as well as the message output descriptor.

Non-literal input message fields can be defined to allow for 2 bytes of attribute data. When a field is so defined, MFS will reserve the first 2 bytes of the field for attribute data to be filled in by the application program when preparing an output message. In this way, the same program area can be conveniently used for both input and output messages. When attribute space is specified, the specified field length must include the 2 attribute bytes.

18.4.1.3 IMS passwords

If the input data is for a password protected transaction, a device field should be designated for the password. The device field in which the operator keys in the password will not be displayed on the screen.

18.4.2 Output message formatting

All output messages for 3270 devices are processed by MFS in a way similar to input.

18.4.2.1 Output data formatting using MFS

All MFS output formatting is based on the contents of two MFS control blocks -- the message output descriptor (MOD) and the device output format (DOF). See Figure 57, the MOD defines output message content and optionally, literal data to be considered part of the output message. Message fields ((MFLDs) refer to device field locations via device field (DFLD) definitions in the DOF. The DOF specifies the use of hardware features, device field locations and attributes, and constant data considered part of the format.

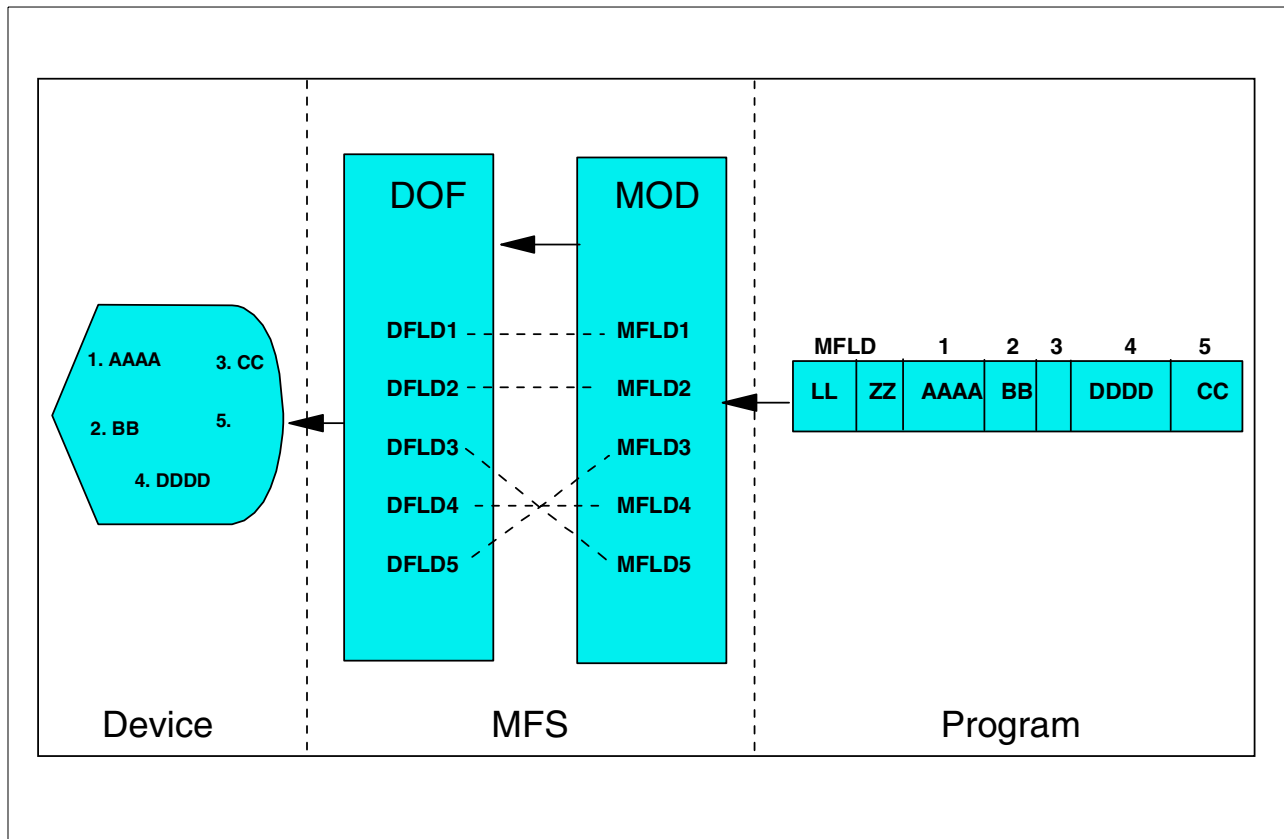


Figure 57. MFS output formatting

The layout of the output message segment to be received by MFS from the program is defined by a list of MFLDs in the MOD. The DOF in turn contains a list of DFLDs which define where the data is to be displayed/printed on the output device. MFS maps the data of the MFLDs into the corresponding DFLDs.

All fields in an output message segment must be defined by MFLD statements. Fields can be truncated or omitted by two methods. The first method is to insert a short segment. The second method is to place a NULL character (X'3f') in the field. Fields are scanned left (including the attribute bytes, if any) to right for NULL character. The first NULL character encountered terminates the field. If the first character of a field is a NULL character, no data is sent to the screen for this field. This means that if the field is protected and the same device format is used, the old data remains on the screen. To erase the old data of a protected field, the application program must send X'403F' to that field.

Positioning of all fields in the segment remains the same regardless of NULL characters. Truncated fields are padded with a program tab character in our subset. Furthermore, we always specify erase-unprotected-all in the display device format. This erases all old data in unprotected fields on the screen.

Notes:

1. Device control characters are invalid in output message fields under MFS. The control characters HT, CR, LF, NL, and BS will be changed to null characters (X'CC'). All other nongraphic characters are X'40' through X'FE'.
2. With MFS, the same output message can be mapped on different device types with one set of formats. This will not be covered in our subset. The formatting discussed will cover one device type per device format, not a mixture. However, the mixture can be implemented later by changing the formats.

In addition to MFLD data, constants can be mapped into DFLDs. These constants are defined as literals in DFLD or MFLD statements.

18.4.2.2 Multiple segment output messages

MPS allows mapping of one or more output segments of the same message onto a single or multiple output screens. In our subset, we will limit ourselves to a one-to-one relationship between output message segments and logical output pages. Also one logical output page is one physical output page (one screen).

18.4.2.3 Logical paging of output messages

Logical paging is the way output message segments are grouped for formatting. When logical paging is used, an output message descriptor is defined with one or more LPAGE statements. Each LPAGE statement relates a segment produced by an application program to corresponding device page.

Using logical paging, the simplest message definition consists of one LPAGE and one segment description. As shown in Figure 58, each segment produced by the application program is formatted in the same manner using the corresponding device page.

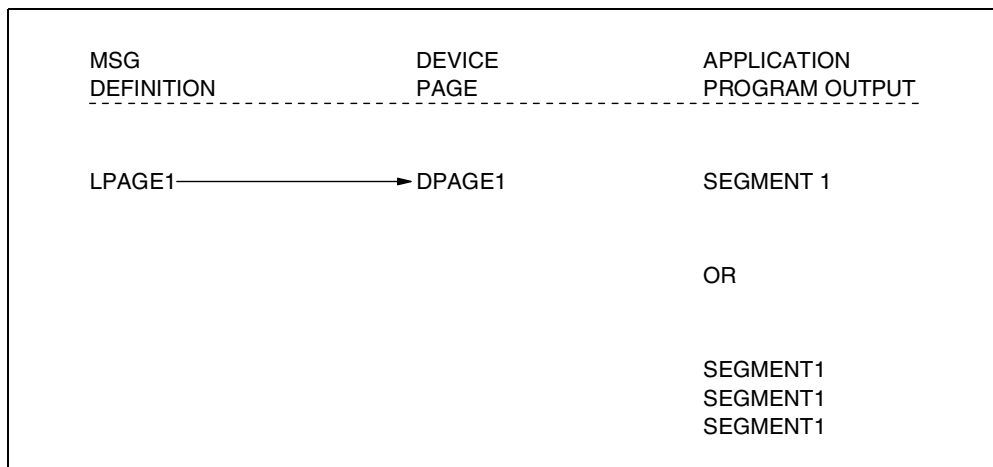


Figure 58. An output message definition with one LPAGE

With the definition shown in Figure 58, each output segment inserted by the MPP will be displayed with the same and only defined MOD/DOF combination.

If different formats are required for different output segments, one LPAGE and SEG statement combination is required for each different format. Each LPAGE can link to a different DPAGE if desired. (This would not be required if only defined constants and MFLDs differ in the MOD.)

The selection of the DPAGE to be used for formatting is based on the value of a special MFLD in the output segment. This value is set by the MPP. If the LPAGE to be used cannot be determined from the segment, the last defined LPAGE is used. See also the description of the COND parameter of the LPAGE statement. Each LPAGE can refer to a corresponding DPAGE with unique DFLDs for its own device layout. See Figure 59.

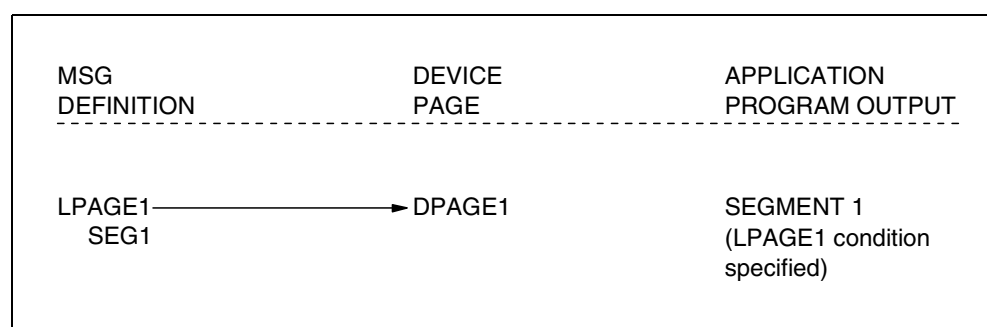


Figure 59. An output message definition with multiple pages

18.4.2.4 Operator paging of output messages

If an output message contains multiple pages, the operator requests the next one with the program access key 1 (PA1). If PA1 is pressed after the last page is received, IMS will send a warning message in our subset. If PA1 is then pressed again, IMS will send the first page of the current output message again.

The operator can always request the next output message by pressing the PA2 key. Also, in our subset, when the operator enters data, the current output message is dequeued.

18.4.2.5 Output message literal fields

Output message fields can be defined to contain literal data specified by the user during definition of the MOD. MFS will include the specified literal data in the output message before sending the message to the device.

MFS users may define their own literal field and/or select a literal from a number of literal provided by MFS. The MFS - provided literals are referred to as system literals and include various date formats, a time stamp, the output message sequence number, the logical terminal name, and the number of the logical page.

18.4.2.6 Output device field attributes

Device field attributes are defined in DFLD statements. For 3270 display devices, specific attributes may be defined in the ATTR= keyword of the DFLD statement. If not, default attributes will be assumed. The message field definition (MFLD) corresponding to the device field (DFLD) may specify that the application program can dynamically modify the device field attributes.

When a field is so defined, the first 2 data bytes of the field are reserved for attribute data. Any error in the 2-byte specification causes the entire specification to be ignored, and the attributes defined or defaulted for the device field are used.

Note: The two attribute bytes should not be included in the length specification of the device field (DFLD) in the DOF.

The default attributes for non-literal 3270 display device fields are alphabetic, not-protected, normal display intensity, and not-modified. Literal device fields have forced attributes of protected and not-modified and default attributes of numeric and normal display intensity. Numeric protected fields provide an automatic skip function on display terminals.

18.4.2.7 Cursor positioning

The positioning of the cursor on the 3270 display device is done in either of two ways:

1. The DPAGE statement defines the default cursor position.
2. The program can dynamically set the cursor to the beginning of a field via its attribute byte.

18.4.2.8 System message field (3270 display devices)

Output formats for 3270 display devices may be defined to include a system message field. If so defined, all IMS messages except DFS057 REQUESTED FORMAT BLOCK NOT AVAILABLE are not sent to the system message field whenever the device is in formatted mode. Providing a system message field avoids the display of an IMS message elsewhere on the screen, thereby overlaying the screen data.

When MPS sends a message to the system message field, it activates the device alarm (if any) but does not reset modified data tags (MDTs) or move the cursor. Since an IMS error message is an immediate response to input, MDTs remain as they were at entry and the operator merely has to correct the portion of the input in error.

In our subset we will always reserve the bottom line of the screen for the system message field. This field can also be used to enter commands, for example, /FORMAT

18.4.2.9 Printed page format control

The 3270 printer devices are also supported via MFS. Three basic options can be specified in the DEV statement (PAGE=operand):

- A defined fixed number of lines should always be printed for each page (SPACE). This is the recommended option because it preserves forms positioning.
- Only lines containing data should be printed. Blank lines are deleted (FLCAT).
- All lines defined by DFLDs should be printed, whether or not the DFLDs contain data (DEFN).

18.4.3 MFS formats supplied by IMS

Several formats are included in the IMSVS.FORMAT library during IMS system definition. They are used mainly for the master terminal, and for system commands and messages. All these formats start with the characters DFS. One of the most interesting in our subset is the default output message format. This format is used for broadcast messages from the master terminal and application program output messages with no MOD name specified. It permits two segments of input, each being a line on the screen. DFSDF2 is the format name, DFSM02 the MOD and DFSMI2 the MID name.

When the master terminal format is used, any message whose MOD name begins with DFSMO (except DFSM03) is displayed in the message area. Any message whose MOD name is DFSDP01 is displayed in the display area.

Messages with other MOD names cause the warning message USER MESSAGE WAITING to be displayed at the lower portion of the display screen.

18.5 MFS control statements

This section describes the control statements used by the MFS language utility. There are two major categories of control statements:

- Definition statements are used to define message formats and device formats.
- Compiler statements are used to control the compilation and listings of the definition statements.

The definition of message formats and device formats is accomplished with separate hierarchical sets of definition statements. The statement set used to define message formats consists of the following statements:

MSG	Identifies the beginning of a message definition.
LPAGE	Identifies a related group of segment/field definitions.
PASSWORD	Identifies a field to be used as an IMS password.
SEG	Identifies a message segment.
MFLD	Defines a message field. Interactive processing of MFLD statements can be invoked by specifying DC and statements. To accomplish interactive processing the DO statement is placed before the MFLD statement (s) and the ENDDO after the MFLD statements (s). See the following discussion on compilation statements.
MSGEND	Identifies the end of a message definition.

The statement set used to define device formats consists of the following statements:

FMT	Identifies the beginning of a format definition.
DEV	Identifies the device type and operational options.
DIV	Identifies the format as input, output, of both.
DPAGE	Identifies a group of device fields corresponding to an LPAGE group of message fields.

DFLD Defines a device field. Interactive processing of DFLD statements can be invoked by specifying DC and ENDDO statements. To accomplish interactive processing, the DO statement is placed before the DFLD statement (s) and the ENDD after the DFLD statement(s). See the following discussion on compilation statements.

FMTEND Identifies the end of the format definition.

Compilation statements have variable functions. The most common ones are:

DO Requests interactive processing of MFLD or DFLD definition statements.

EJECT Ejects SYSPRINT listing to the next page.

END Defines the end of data for SYSIN processing.

ENDDO Terminates interactive processing of MFLD or DFLD

PRINT Controls SYSPRINT options.

SPACE Skips lines on the SYSPRINT listing.

TITLE Provides a title for the SYSPRINT listing.

Compilation statements are to be inserted at logical points in the sequence of control statements. For example, TITLE could be placed first, and EJECT could be placed before each MSG or FMT statement.

18.5.1 Relations between source statements and control blocks

In general, the following relations exists between the MFS source statements and control blocks.

- One MSG statement and its associated LPAGE, SEG, and MFLD statements generate one MID or MOD.
- One FMI statement and its associated DEV, DIV, DPAG and DFLD statements generate one DIF and/or DOF. For displays, both the DIF and DOF are generated, because the output screen is used for input too.

In addition the MFS utilities will establish the linkages between the MID, MOD, DIF, and DOF. These are the result of the symbolic name linkages defined in the source statements.

18.5.2 MFS control block generation

MFS control blocks are generated by execution of the MFS language utility program. This is a two-stage process. See Figure 60.

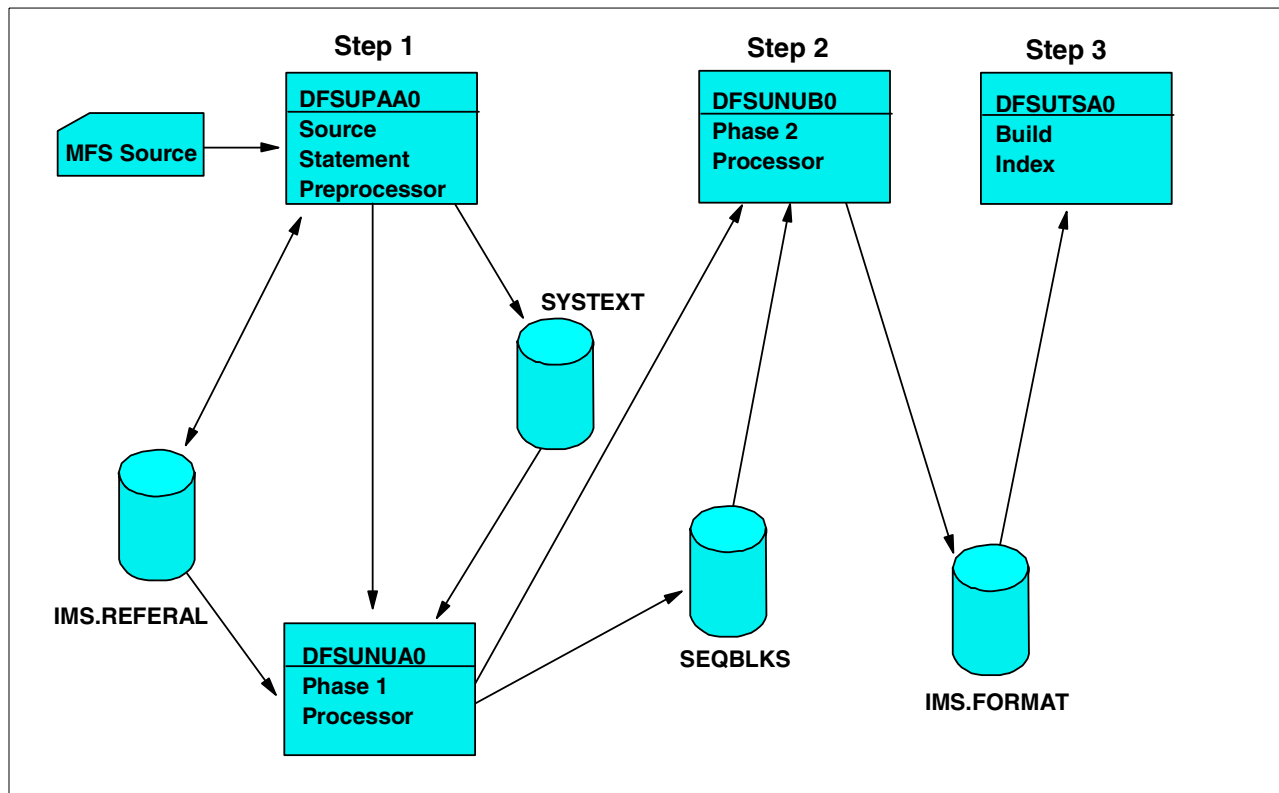


Figure 60. Creation of MFS control blocks

The MFS control block generated can be executed by an IMS supplied cataloged procedure: MFSUTL. Multiple formats can be generated with one execution. In general you would process a complete format set, i.e, the related message and format descriptions, in one execution of MFSUTL. Three executions of MFSUTL are involved to process the three sample format sets.

18.5.2.1 Step 1

Preprocessor

The MFS language utility preprocessor generates intermediate text blocks (ITBs), based on the MFS language source statement. Definitions of the MFS language utility source input are contained in this chapter under the topic “MFS Control Statements.” The primary function of the preprocessor is to perform syntax and relational validity checks on user specifications and generate ITBs. The ITBs are then processed by phase 1 of the utility to generate message (MSG) and format (FMT) descriptors. An ITB generated for each MSG or FMT description can be re-used by the same or another format set, once it has been successfully added to the IMSVS.REFERAL data set. Each such description must start with a MSG or FMT statement and end with a MSGND or FMTEND statement.

Phase 1

The preprocessor invokes phase 1 if the highest return code generated by the preprocessor is less than 16. Phase 1 places the newly constructed descriptors on the SEQBLKS data set. Each member processed has a control record placed on the SEQBLKS data set identifying the member, its size, and the date and time of creation. This control record is followed by the image of the descriptor as constructed by phase 1. Alternatively, if an error is detected during descriptor building, an error control record is placed on the SEQBLKS data set for the description in error, identifying the member in error, and the date and time the error control record was created. In addition, phase 1 returns a completion code of 12 to MVS. If execution of step 2 is forced, phase 2 will delete descriptors with build errors.

18.5.2.2 Step 2

Phase 2

Phase 2 receives control as a job step following phase 1. After final processing, it will place the new descriptors into the IMSVS.FORMAT library. Phase 2 passes a completion code to MVS for step 2 based on all the descriptor maintenance to IMSVS.FORMAT for a given execution of the MFS language utility.

18.5.2.3 Step 3

In our subset, we will always execute the MFS service utility after MFS control block generation. This utility will build a new index directory block which will eliminate the need for directory search operations during the IMS online operation.

18.5.3 MFS library maintenance

The IMSVS.FORMAT and IMSVS.REFERAL libraries are standard MVS partitioned data sets. Backup and restore operations can be done with the proper MVS utility (IEBCOPY). However, care must be taken that both the IMSVS.FORMAT and the IMSVS.REFERAL data sets are dumped and restored at the same time.

Chapter 19. Application coding for IMS Database Manager

This chapter, which covers database processing, is divided into four sections:

- The first section offers a general introduction to DL/I database processing. It defines the basic structure of a DL/I application program.
- The second section introduces basic DL/I calls against a single hierarchical database structure.
- The third section covers the processing of logical databases which are implemented with the DL/I logical relationships function.
- The fourth section deals with the use of secondary indices.

19.1 Introduction to database processing

In general, database processing is transaction oriented. Generally, an application program accesses one or more database records for each transaction it processes. There are two basic types of DL/I application programs:

- The direct access program
- The sequential access program

A direct access program accesses, for every input transaction, some segments in one or more database records. These accesses are based on database record and segment identification. This identification is essentially derived from the transaction input. Normally it is the root-key value and additional (key) field values of dependent segments. For more complex transactions, segments could be accessed in several DL/I databases concurrently.

A sequential application program accesses sequentially selected segments of all of a consecutive subset of a particular database. The sequence is usually determined by the key of the root-segment. A sequential program can also access other databases, but those accesses are direct, unless the root-keys of both databases are the same.

A DL/I application program normally processes only particular segments of the DL/I databases. The portion that a given program processes is called an application data structure. This application data structure is defined in the program specification block (PSB). There is one PSB defined for each application program. An application data structure always consists of one or more hierarchical data structures, each of which is derived from a DL/I physical or logical database.

19.1.1 Interface to IMS

During initialization, both the application program and its associated PSB are loaded from their respective libraries by the IMS batch system. The DL/I modules, which reside together with the application program in one partition/region, interpret and execute database CALL requests issued by the program.

19.1.1.1 Calls to DL/I

A call request is composed of a CALL statement with an argument list. The argument list specifies the processing function to be performed, the hierarchic path to the segment to be accessed, and the segment occurrence of that segment. One segment may be operated upon with a single DL/I call. However, a single call never will return more than one occurrence of one segment type.

The arguments contained within any DL/I call request have been defined in “Calls to IMS” on page 135. The following is a sample for a basic CALL statement for COBAL:

```
CALL "CBLTDLI" USING function,PCB-name,I/O Area, SSA1,...SSAn.
```

Table 10 describes the components of the CALL statement. Here you will find the basic DL/I call functions to request DL/I database services.

Table 10. *DLI function descriptions*

RSF	DL/I Call Function
GET UNIQUE	'GUbb'
GET NEXT	'GNbb'
GET HOLD UNIQUE	'GHUb'
GET HOLD NEXT	'GHNb'
INSERT	'ISRT'
DELETE	'DLET'
REPLACE	'REPL'

Note: b stands for blank: each CALL function is always 4 characters.

Table 11 constitutes the various categories of segment access types.

Table 11. *Segment access*

Segment Access	DL/I Call Function
Retrieve a segment	GU, GN, GHU, GHN
Replace (Update) a segment	REPL
Delete a segment	DLET
Insert (add) a segment	ISRT
Delete	'DLET'
Replace	'REPL'

In addition to the above database calls, there are the system service calls. These are used for requesting systems services such as checkpoint/restart. All of the above calls and some basic system service calls will be discussed in detail in the following sections.

19.1.1.2 Segment search arguments

For each segment accessed in a hierarchical path, one SSA can be provided. The purpose of the SSA is to identify by segment name and, optionally, by field value, the segment to be accessed.

The basic function of the SSA permits the application program to apply three different kinds of logic to a call:

- Narrow the field of search to a particular segment type, or to a particular segment-occurrence.
- Request that either one segment or a path of segments be processed.
- Alter DL/I's position in the database for subsequent call.

Segment Search Argument (SSA) names represent the fourth (fifth for PL/I) through last arguments (SSA1 through SSAn) in the call statement. There can be 0 or 1 SSA per level, and, since DL/I permits a maximum of 15 levels per database, a call may contain from 0 to 15 SSA names. In our subset, an SSA consists of one, two or three elements: The segment name, command code(s) and a qualification statement, as shown in Table 12. Table 13 shows the values of the relational operators described in Table 12.

Table 12. Segment name, command code and qualifications

Operator	Description
Segment Name	The segment name must be eight bytes long, left-justified with trailing blanks required. This is the name of the segment as defined in a physical and/or logical DBD referenced in the PCB for this application program.
Command Codes	The command code are optional. They provide functional variations to be applied to the call for that segment type. An asterisk (*) following the segment name indicates the presence of one or more command codes. A blank or a left parenthesis is the ending delimiter for command codes. Blank is use when no qualification statement exists.
Qualification statement	The presence of a qualification statement is indicated by a left parenthesis following the segment name or, if present, command codes. The qualification statement consists of a field name, a relational-operator, and a comparative-value.
Begin Qualification Character	The Left parenthesis, "(", indicates the beginning of a qualification statement. If the SSA is unqualified, the eight-byte segment name or if used, the command codes, should be followed by a blank.
Field name	Is the name of a field which appears in the description of the specified segment type in the DBD. The name is up to eight characters long, left-justified with trailing blanks as required. The named field may be either the key field or any data field within a segment. The field name issued for searching the database, and must have been defined in the physical DBD.
Relational Operator	Is a set of two characters which express the manner in which the contents of the field, referred to by the field name, is to be tested against the comparative-value. See Table 13 for a list of the values.

Operator	Description
Comparative-value	Is the value against which the contents of the field, referred to by the field name, is to be tested. The length of this field must be equal to the length of the named field in the segment of the database. That is, it includes leading or trailing blanks (for alphameric) or zeros (usually needed for numeric fields) as required. A collating sequence, not an arithmetic, compare is performed.
End Qualification Character	The right parenthesis, “)”, indicates the end of the qualification statement.

Table 13. Relational operator values

Operator	Meaning
b= or 'EQ'	Must be equal to
>= or 'GE'	Must be greater than or equal to
<= or 'LE'	Must be less than or equal to
'b>' or 'GT'	Must be greater than
'b<' or 'LT'	Must be less than
'b<>' or 'NE'	Must not be equal to

Note: As used above, the lowercase b represents a blank character.

19.1.1.3 Qualification

Just as calls are “qualified” by the presence of an SSA, SSAs are categorized as either “qualified” or “unqualified”, depending on the presence or absence of a qualification statement. Command codes may be included in or omitted from either qualified or unqualified SSAs.

In its simplest form, the SSA is unqualified and consists only of the name of a specific segment type as defined in the DBD. In this form, the SSA provides DL/I with enough information to define the segment type desired by the call.

Example: `SEGNAMEbb` last character blank to unqualified.

Qualified SSAs (optional) contain a qualification statement composed of three parts: A field name defined in the DBD, a relational operator, and a comparative value. DL/I uses the information in the qualification statement to test the value of the segment’s key or data fields within the database, and thus to determine whether the segment meets the user’s specifications. Using this approach. DL/I performs the database segment searching. The program need process only those segments which precisely meet some logical criteria.

Example: `SEGNAMEb (fieldxxx>=value)`

The qualification statement test is terminated either when the test is satisfied by an occurrence of the segment type, or when it is determined that the request cannot be satisfied.

19.1.1.4 Command codes

Both unqualified and qualified SSAs may contain one or more optional command codes which specify functional variations applicable to the call function or the segment qualification. The command codes are discussed in detail later in this chapter.

General Characteristics of segment search arguments:

- An SSA may consist of the segment name only (unqualified). It may optionally also include one or more command codes and a qualification statement.
- SSAs following the first SSA must proceed down the hierarchical path. Not all SSAs in the hierarchical path need be specified. That is, there may be missing levels in the path. DL/I will provide, internally, SSAs for missing levels according to the rules given later in this chapter. However, it is strongly recommended to always include SSAs for every segment level.

Examples of SSAs will be given with the sample calls at each DL/I call discussion in the following section.

19.1.2 Status code handling

After each DL/I call, a two-byte status code is returned in the PCB which is used for that call. We distinguish between three categories of status code:

- The blank status code, indicating a successful call,
- Exceptional conditions and warning status codes from an application point of view.
- Error status codes, specifying an error condition in the application program and/or DL/I.

The grouping of status codes in the above categories is somewhat installation dependent. We will, however, give a basic recommendation after each specific call function discussion. It is also recommended that you use a standard procedure for status code checking and the handling of error status code. The first two categories should be handled by the application program after each single call. Figure 61 gives an example using COBOL.

```
CALL 'CBLTDLI' USING ....  
IF PCB-STATUS EQ 'GE' PERFORM PRINT-NOT-FOUND.  
IF PCB STATUS NE 'bb' PERFORM STATUS-ERROR.  
everything okay, proceed....
```

Figure 61. Testing status codes

Notice that it is more convenient to directly test the regular exceptions in-line instead of branching to a status code check routine. In this way, you clearly see the processing of conditions that you wish to handle from an application point of view, leaving the real error situations to central status code error routine. A detailed discussion of the error status codes and their handling will be presented later in this chapter.

19.1.3 Sample presentation of a call

DL/I calls will be introduced in the following sections. For each call we will give samples. These samples will be in a standard format, as shown in Figure 62.

```
77  GU-FUNC          PICTURE XXXX VALUE 'Gubb'

01  SSA001-GU-SE1PART.
    02  SSA001-BEGIN  PICTURE ...
    02  ...
    02  ...

01  IOAREA          PICTURE X(256) .
-----
CALL 'CBLTDLI' USING GU-FUNC,PCB-NAME,IOAREA,SSA001-GU-SE1PART.
-----
STATUS CODES:
-----
      bb:  succesfull call
      --:  exceptional but correct condition
      other: error condition
```

Figure 62. Sample call presentation

All the calls in the samples are presented in COBOL format. The coding of a call in PI/I or Assembler will be presented later. Each call example contains three sections.

1. The first section presents the essential elements of working storage as needed for the call.
2. The second part, the processing section, contains the call itself. Note that the PCB-NAME parameter should refer to the selected PCB defined in the Linkage Section. Sometimes we will add some processing function description before and/or after the call, in order to show the call in its right context.
3. The third section contains the status codes and their interpretation, which can be expected after the call.

The last category of status code, labeled “other: error situation,” would normally be handled by a status code error routine. We will discuss those error status codes with the presentation of such a routine later in this chapter.

19.2 Processing against a single database structure

This is the section which deals with handling a single database record. A database record is a root segment and all of its physically dependent child segments.

19.2.1 DL/I positioning concept

To satisfy a call, DL/I relies on two sources of segment identification:

- The established position in the database as set by the previous call against the PCB.
- The segment search arguments as provided with the call.

The database position is the knowledge of DL/I of the location of the last segment retrieved and all segments above it in the hierarchy. This position is maintained by DL/I as an extension of, and reflected in, the PCB. When an application program has multiple PCBs for a single database, these positions are maintained independently. For each PCB, the position is represented by the concatenated key of the hierarchical path from the root segment down to the lowest level segment accessed. It also includes the positions of non-keyed segments.

If no current position exists in the database, then the assumed current position is the start of the database. This is the first physical database record in the database. With HDAM this is not necessarily the root-segment with the lowest key value.

19.2.2 Retrieving segments

There are two basic ways to retrieve a segment.

- Retrieve a specific segment - GU
- Retrieve the next segment in hierarchy - GN

If you know the specific key value of the segment you want to retrieve, then the GU call will allow to retrieve only the required segment. If you don't know the key value or don't care then the GN call will retrieve the next available segment which meets your requirements.

19.2.2.1 The get unique call — GU

The basic get unique (GU) call, function code "GUb" normally retrieves one segment in a hierarchical path. The segment retrieved is identified by an SSA for each level in the hierarchical path down to and including the requested segment. Each should contain at least the segment name. The SSA for the root-segment should provide the root-key value. To retrieve more than one segment in the path see 19.2.4.1, "D Command code" on page 185. Figure 63 shows an example of the get unique call.

```

77  GU-FUNC          PICTURE XXXX VALUE 'GUb'

01  SSA001-GU-SE1PART.
    02  SSA001-BEGIN  PICTURE x(19) VALUE 'SE1PARTb(FE1PGPNRb=' .
    02  SSA001-FE1PGPNR PICTURE X(8) .
    02  SS1001-END    PICTURE X      VALUE ')' .

01  IOAREA          PICTURE X(256) .
-----
MOVE PART-NUMBER TO SSA001-FE1PGPNR.
CALL 'CBLTDLI' USING GU-FUNC,PCB-NAME,IOAREA,SSA001-GU-SE1PART.
-----
STATUS CODES:
-----
      bb:  succesfull call
      GE:  exceptional but correct condition
      other: error condition

```

Figure 63. Basic GU call

The main use of the GU call is to position yourself to a database record and obtain (a path of) segment (s). Typically, the GU call is used only once for each database record you wish to access. Additional segments within the database record would then be retrieved by means of get next calls (See the following section.) The GU call can also be used for retrieving a dependent segment, by adding additional SSAs to the call. For example, if you add a second SSA which specifies the stock location, you would retrieve a STOCK segment below the identified part. If the SSA did not provide a stock location number, this would be the first STOCK segment for this part.

19.2.2.2 The get next call — GN

The get next (GN) call, function code 'GNbb', retrieves the next segment in the hierarchy as defined in the PCB. To determine this next segment, DL/I relies on the previously established position.

19.2.2.3 Unqualified get next call

Figure 64 shows a get next call with no SSAs at all that will, if repeated, return the segments in the database in hierarchical sequence. Only those segments are returned to which the program is defined sensitive in its PCB. If this call was issued after the get unique call in Figure 63, then it would retrieve the first STOCK segment for this part (if one existed). Subsequent calls would retrieve all other STOCK, PURCHASE ORDER, and DESCRIPTION segments for this part. After this, the next part would be retrieved and its dependent segments, etc., until the end of the database is reached. Special status codes will be returned whenever a different segment type at the same level or a higher level is returned. No special status code is returned when a different segment at a lower level is returned. You can check for reaching a lower level segment type in the segment level indicator in the PCB. Remember, only those segments to which the program is sensitive via its PCB are available to you.

77	GN-FUNC	PICTURE XXXX VALUE 'GNbb'
01	IOAREA	PICTURE X(256) .

CALL 'CBLTDLI' USING GN-FUNC,PCB-NAME,IOAREA.		

STATUS CODES:		

bb:	if previous call retrieved a PART, then a STOCK segment will be retrieved	
GK:	a segment is returned in IOAREA, but it is a different type at the SAME level, for instance, a PURCHASE ORDER segment after the last STOCK segment.	
GA:	segment returned is IOAREA, but it is of a higher level than the last one, that is, a new PART segment	
GB:	end of database reached, no segment returned	
other:	error condition	

Figure 64. Unqualified GN call

Although the above unqualified GN call may be efficient, especially for report programs, you should use a qualified GN call whenever possible.

19.2.2.4 The qualified get next call

This qualified GN call should at least identify the segment you want to retrieve. In doing so, you will achieve a greater independence toward possible database structure changes in the future. Figure 65 shows an example of a qualified GN call. If you supply only the segment name in the SSA, then you will retrieve all segments of that type from all database records with subsequent get next calls.

```
77  GN-FUNC          PICTURE XXXX VALUE 'GNbb'

01  SSA002-GN-SE1PPUR PICTURE X(9) VALUE 'SE1PPURbb'

01  IOAREA           PICTURE X(256) .
-----
MOVE PART-NUMBER TO SSA001-FE1PGPNR.
CALL 'CBLTDLI' USING GN-FUNC,PCB-NAME,IOAREA,SSA002-GN-SE1PPUR.

-----
STATUS CODES:
-----
      bb:  next PURCHASE ORDER has been move to the IOAREA
      GB:  end of database reached, no segment returned
      other: error condition
```

Figure 65. Qualified GN call

Repetition of the above GN call will retrieve all subsequent PURCHASE ORDER segments of the database, until the end of the database is reached. To limit this to a specific part, you could add a fully qualified SSA for the PART segment. This would be the same SSA as used in Figure 63 on page 179.

An example of a qualified get next call with a qualified SSA is shown in Figure 66.

```
77  GN-FUNC          PICTURE XXXX VALUE 'GNbb'

01  SSA001-GU-SE1PART.
    02  SSA001-BEGIN   PICTURE x(19) VALUE 'SE1PARTb(FE1PGPNRb=' .
    02  SSA001-FE1PGPNR PICTURE X(8) .
    02  SS1001-END     PICTURE X      VALUE ')' .

01  SSA002-GN-SE1PPUR PICTURE X(9) VALUE 'SE1PPURb' .
01  IOAREA           PICTURE X(256) .
-----
CALL 'CBLTDLI' USING GN-FUNC,PCB-NAME,IOAREA,SSA001-GU-SE1PART
                  SSA002-GN-SE1PPUR.

-----
STATUS CODES:
-----
      bb:  next PURCHASE ORDER segment is in IOAREA
      GE:  segment not found; no more purchase orders for this part,
           or part number in SSA001 does not exist
      other: error condition
```

Figure 66. GN call with qualified SSA

This fully qualified get next call should be primarily used. It always clearly identifies the hierarchical path and the segment you want to retrieve.

19.2.2.5 Get hold calls

To change the contents of a segment in a database through a replace or delete call, the program must first obtain the segment. It then changes the segment's contents and requests DL/I to replace the segment in the database or to delete it from the database.

This is done by using the get hold calls. These function codes are like the standard get function, except the letter 'H' immediately follows the letter 'G' in the code (that is, GHU, GHN). The get hold calls function exactly as the corresponding get calls for the user. For DL/I they indicate a possible subsequent replace or delete call.

After DL/I has provided the requested segment to the user, one or more fields, but not the sequence field, in the segment may be changed

After the user has changed the segment contents, he can call DL/I to return the segment to, or delete it from the database. If, after issuing a get hold call, the program determines that it is not necessary to change or delete the retrieved segment, the program may proceed with other processing, and the "hold" will be released by the next DL/I call against the same PCB.

19.2.3 Updating segments

Segments can be updated by application programs and returned to DL/I for restoring in the database, with the replace call, function code 'REPL' Two conditions must be met:

The segment must first be retrieved with a get hold call, (GHU or GHN), no intervening calls are allowed referencing the same PCB.

The sequence field of the segment cannot be changed. This can only be done with combinations of delete and insert calls for the segment and all its dependents.

Figure 67 shows an example of a combination of GHU and REPL call. Notice that the replace call must not specify a SSA for the segment to be replaced. If, after retrieving a segment with a get hold call, the program decides not to update the segment, it need not issue a replace call. Instead the program can proceed as if it were a normal get call without the hold.

Because there is only a very small performance difference between the get and the get hold call, you should use the get hold call whenever there is a reasonable chance (about 5% or more) that you will change the segment.

```

77 GHU-FUNC          PICTURE XXXX VALUE 'GHUb' .
77 REPL-FUNC         PICTURE XXXX VALUE 'REPL' .

01 SSA001-GU-SE1PART.
  02 SSA001-BEGIN    PICTURE x(19) VALUE 'SE1PARTb(FE1PGPNRb=' .
  02 SSA001-FE1PGPNR PICTURE X(8) .
  02 SS1001-END      PICTURE X      VALUE ')' .
01 SSA002-GN-SE1PPUR PICTURE X(9)  VALUE 'SE1PPURbb' .
01 IOAREA            PICTURE X(256) .

-----
MOVE PART-NUMBER TO SSA001-FE1PGPNR.
CALL 'CBLTDLI' USING GU-FUNC,PCB-NAME,IOAREA,SSA001-GU-SE1PART
                      SSA002-GN-SE1PPUR.

    the retrieved PURCHASE ORDER segment can now be changed by the program
    in the IOAREA.
CALL 'CBLTDLI' USING REPL-FUNC,PCB-NAME,IOAREA.

-----
STATUS CODES:
-----
    bb:    segment is replaced with contents in the IOAREA
    other: error condition

```

Figure 67. Basic REPL call

19.2.3.1 Deleting segments

To delete the occurrence of a segment from a database, the segment must first be obtained by issuing a get hold (GHU, GHN) call. Once the segment has been acquired, the DLET call may be issued.

No DL/I calls which use the same PCB can intervene between the get hold call and the DLET call, or the DLET call is rejected. Quite often a program may want to process a segment prior to deleting it. This is permitted as long as the processing does not involve a DL/I call which refers to the same database PCB used for the get hold/delete calls. However, other PCBs may be referred to between the get hold and DLET calls.

DL/I is advised that a segment is to be deleted when the user issues a call that has the function DLET. The deletion of a parent, in effect, deletes all the segment occurrences beneath that parent, whether or not the application program is sensitive to those segments. If the segment being deleted is a root segment, that whole database record is deleted. The segment to be deleted must still be in the IOAREA of the delete call (with which no SSA is used), and its sequence field must not have been changed. Figure 68 gives an example of a DLET call.

```

77  GHU-FUNC          PICTURE XXXX VALUE 'GHUbb'.
77  DLET-FUNC         PICTURE XXXX VALUE 'DLET'.

01  SSA001-GU-SE1PART.
    02  SSA001-BEGIN   PICTURE x(19) VALUE 'SE1PARTb(FE1PGPNRb=' .
    02  SSA001-FE1PGPNR PICTURE X(8) .
    02  SS1001-END     PICTURE X      VALUE ')' .
01  SSA002-GN-SE1PPUR  PICTURE X(9)  VALUE 'SE1PPURbb' .
01  IOAREA             PICTURE X(256) .

-----
MOVE PART-NUMBER TO SSA001-FE1PGPNR.
CALL 'CBLTDLI' USING GU-FUNC,PCB-NAME,IOAREA,SSA001-GU-SE1PART
      SSA002-GN-SE1PPUR.

      the retrieved PURCHASE ORDER segment can now be processed by the
      program in the IOAREA.

CALL 'CBLTDLI' USING DLET-FUNC,PCB-NAME,IOAREA.
-----
STATUS CODES:
-----
      bb:   requested purchase order segment is deleted from the database;
           all its dependents, if any, are deleted also.
      other: error condition

```

Figure 68. Basic DLET call

19.2.3.2 Inserting segments

Adding new segment occurrences to a database is done with the insert call, function code 'ISRT'

The DL/I insert call is used for two distinct purposes: It is used initially to load the segments during creation of a database. It is also used to add new occurrences of an existing segment type into an established database. The processing options field in the PCB indicates whether the database is being added to or loaded. The format of the insert call is identical for either use.

When loading or inserting, the last SSA must specify only the name of the segment being inserted. It should specify only the segment name, not the sequence field. Thus an unqualified SSA is always required.

Up to a level to be inserted, the SSA evaluation and positioning for an insert call is exactly the same as for a GU call. For the level to be inserted, the value of the sequence field in the segment in the user I/O area is used to establish the insert position. If no sequence field was defined, then the segment is inserted at the end of the physical twin chain. If multiple non-unique keys are allowed, then the segment is inserted after existing segments with the same key value.

Figure 69 shows an example of an ISRT call. The status codes in this example are applicable only to non-initial load inserts. The status codes at initial load time will be discussed under the topic 19.7.1, "Loading a database" on page 196.

```

77  ISRT-FUNC          PICTURE XXXX VALUE 'ISRT' .

01  SSA001-GU-SE1PART.
    02  SSA001-BEGIN    PICTURE x(19) VALUE 'SE1PARTb(FE1PGPNRb=' .
    02  SSA001-FE1PGPNR PICTURE X(8) .
    02  SS1001-END      PICTURE X      VALUE ')' .
01  SSA002-GN-SE1PPUR  PICTURE X(9)   VALUE 'SE1PPURbb' .
01  IOAREA             PICTURE X(256) .

-----
MOVE PART-NUMBER TO SSA001-FE1PGPNR.
MOVE PURCHASE-ORDER TO IOAREA.
CALL 'CBLTDLI' USING ISRT-FUNC,PCB-NAME,IOAREA,SSA001-GU-SE1PART
                      SSA002-GN-SE1PPUR.

-----
STATUS CODES:
-----
bb:   new PURCHASE ORDER segment is inserted in database
II:   segment to insert already exists in database
GE:   segment not found; the requested part number (that is, a
      parent of the segment to be inserted) is not in the database
other: error condition

```

Figure 69. Basic ISRT call

Note: There is no need to check the existence of a segment in the database with a preceding retrieve call. DL/I will do that at insert time, and will notify you with an II or GE status code. Checking previous existence is only relevant if the segment has no sequence field.

19.2.4 Calls with command codes

Both unqualified and qualified SSAs may contain one or more optional command codes which specify functional variations applicable to either the call function or the segment qualification. Command codes in an SSA are always prefixed by an asterisk (*), which immediately follows the 8 byte segment name. Figure 70 illustrates an SSA with command codes D and P.

```

01  SSA001-GU-SE1PART.
    02  SSA001-BEGIN    PICTURE x(19) VALUE 'SE1PARTb*DP(FE1PGPNRb=' .
    02  SSA001-FE1PGPNR PICTURE X(8) .
    02  SS1001-END      PICTURE X      VALUE ')' .

```

Figure 70. SSA with D and P command codes

19.2.4.1 D Command code

The 'D' command code is the one most widely used. It requests DL/I to issue path calls. A "path call" enables a hierarchical path of segments to be inserted or retrieved with one call. (A "path" was defined earlier as the hierarchical sequence of segments, one per level, leading from a segment at one level to a particular segment at a lower level.) The meaning of the 'D' command code is as follows:

For retrieval calls, multiple segments in a hierarchical path will be moved to the I/C area with a single call. The first through the last segment retrieved are concatenated in the user's I/C area. Intermediate SSAs may be present with or without the 'D' command code. If without, these segments are not moved to the user's I/O area. The segment named in the PCB "segment name feedback area" is the lowest-level segment retrieved, or the last level satisfied in the call in case of a non-found condition. Higher-level segments associated with SSAs having the 'D' command code will have been placed in the user's I/O area even in the not-found case. The 'D' is not necessary for the last SSA in the call, since the segment which satisfies the last level is always moved to the user's I/O area. A processing option of 'P' must be specified in the PSBGEN for any segment type for which a command code 'D' will be used.

For insert calls, the 'D' command code designates the first segment type in the path to be inserted. The SSAs for lower-level segments in the path need not have the D command code set, that is, the D command code is propagated to all specified lower level segments.

Figure 71 shows an example of a path call.

```

77  GU-FUNC          PICTURE XXXX VALUE 'Gubb' .

01  SSA004-GU-SE1PART.
    02  SSA004-BEGIN  PICTURE x(21) VALUE 'SE1PARTb*D(FE1PGPNRb=' .
    02  SSA004-FE1PGPNR PICTURE X(8) .
    02  SS1004-END    PICTURE X      VALUE ')' .
01  SSA005-GN-SE1PGDSC PICTURE X(9)  VALUE 'SE1PGDSCb' .

01  IOAREA          PICTURE X(256) .
-----

CALL 'CBLTDLI' USING GU-FUNC,PCB-NAME,IOAREA,SSA004-GU-SE1PART
                      SSA004-GN-SE1PGDSC.

-----
STATUS CODES:
-----
bb:   both segments (PART and DESCRIPTION) have been placed in IOAREA
GE:   segment not found; PART segment may be retrieved in IOAREA;
      check segment name and level indicator in PCB.
other: error condition

```

Figure 71. Sample path retrieve call

The above example shows a common usage of the path call. Although we don't know if the requested part has a separate DESCRIPTION segment (SE1PGDSC), we retrieve it at almost no additional cost if there is one.

The correct usage of path calls can have a significant performance advantage. You should use it whenever possible, even if the chance of the existence or the need for the dependent segment (s) is relatively small. For instance, if you would need, in 10% or more of the occurrences, the first dependent segment after you inspect the parent, then it is generally advantageous to use a path call to retrieve them both initially.

19.2.4.2 N command code

When a replace call follows a path retrieve call, it is assumed that all segments previously retrieved with the path call are being replaced. If any of the segments have not been changed, and therefore, need not be replaced, the 'N' command code may be set at those levels, telling DL/I not to replace the segment at this level of the path. The status codes returned are the same as for a replace call.

19.2.4.3 F command code

This command code allows you to back up to the first occurrence of a segment under its parent. It has meaning only for a get next call. A get unique call always starts with the first occurrence. Command code F is disregarded for the root segment.

19.2.4.4 L command code

This command code allows you to retrieve the last occurrence of a segment under its parent. This command code should be used whenever applicable.

19.2.4.5 - command code

The hyphen is a null command code. Its purpose is to simplify the maintenance of SSAs using command codes.

19.2.5 Database positioning after DL/I call

As stated before, the database position is used by DL/I to satisfy the next call against the PCB. The segment level, segment name and the key feedback areas of the PCB are used to present the database position to the application program.

The following basic rules apply:

1. If a get call is completely satisfied, current position in the database is reflected in the PCB key feedback area.
2. A replace call does not change current position in the database.
3. Database position after a successful insert call is immediately after the inserted segment.
4. Database position after return of an II status code is immediately prior to the duplicate segment. This positioning allows the duplicate segment to be retrieved with a GN call.
5. Database position after a successful delete call is immediately after all dependents of the deleted segment. If no dependents existed, database position is immediately after the deleted segment.
6. Database position is unchanged by an unsuccessful delete call.
7. After an (partial) unsuccessful retrieve call, the PCB reflects the lowest level segment which satisfied the call. The segment name or the key feedback length should be used to determine the length of the relevant data in the key feedback area. Contents of the key feedback area beyond the length value must not be used, as the feedback area is never cleared out after previous calls. If the level-one (root) SSA cannot be satisfied, the segment name is cleared to blank, and the level and key feedback length are set to 0.

In considering 'current position in the database', it must be remembered that DL/I must first establish a starting position to be used in satisfying the call. This starting position is the current position in the database for get next calls, and is a unique position normally established by the root SSA for get unique calls.

The following are clarifications of 'current position in the database' for special situations:

- If no current position exists in the database, then the assumed current position is the start of the database.
- If the end of the database is encountered, then the assumed current position to be used by the next call is the start of the database.
- If a get unique call is unsatisfied at the root level, then the current position is such that the next segment retrieved would be the first root segment with a key value higher than the one of the unsuccessful call, except when end of the database was reached (see above) or for HDAM, where it would be the next segment in physical sequence.

You can always reestablish your database positioning with a GU call specifying all the segment key values in the hierarchical path. It is recommended that you use a get unique call after each not found condition.

19.2.6 Using multiple PCBs for one database

Whenever there is a need to maintain two or more independent positions in one database, you should use different PCBs. This avoids the reissue of get unique calls to switch forward and backward from one database record or hierarchical path to another. There are no restrictions as to the call functions available in these multiple PCBs. However, to avoid "position confusion" in the application program, you should not apply changes via two PCBs to the same hierarchical path. For simplicity reasons you should limit the updates to one PCB unless this would cause additional calls.

19.2.6.1 System service calls

Besides call functions for manipulating database segments, DL/I provides special system service calls. The most common ones are:

- STATISTICS (STAT) -- This call is used to obtain various statistics from DL/I.
- CHECKPOINT (CHKP) -- CHPK informs DL/I that the user has "checkpointed" his program and that thus may be restarted at this point. The current position is maintained in GSAM databases. For all other databases, you must reposition yourself after each checkpoint call with a get unique call.
- RESTART (XRST) -- XRST requests DL/I to restore checkpointed user areas and reposition GSAM database for sequential processing if a checkpoint ID for restarting has been supplied by the call or in the JCL.

The XRST and CHKP calls will be discussed under the topic 19.8, "Batch checkpoint/restart" on page 200.

19.2.7 Processing GSAM databases

All accessing to GSAM databases is done via DL/I calls. A check is made by DL/I to determine whether a user request is for a GSAM database. If so, control is passed to GSAM, which will be resident in the user region. If not, control is passed to DL/I, and standard hierarchical processing will result.

Calls to be used for GSAM accessing are:

```
CALL 'CBLTDLI' USING call-func,pcb-name,ioarea.
```

where:

call-func

is the name of the field which contains the call function:

- OPEN Open GSAM database
- CLSE Close GSAM database
- GN Retrieve next sequential record
- ISRT Insert a new logical record (at end of database only)

The open and close call are optional calls to be used to explicitly initiate or terminate database operations. The database will automatically be opened by the issuance of the first processing call used and automatically closed at "end-of-data" or at program termination.

Records may not be randomly added to GSAM data sets. The data set may be extended by opening in the load mode, with DISP=MOD, and using the ISRT function code.

pcb-name

is the name of the GSAM PCB

ioarea

is the name of the I/O area for GN/ISRT calls. See Table 14 for status codes.

Table 14. GSAM status codes

Status Codes	Meaning
bb	Successful call, Proceed
GL	end of input data (Get Next only)
other	error situation

19.2.7.1 Record formats

Records may be fixed or variable length, blocked or unblocked. Records must not have a sequence key. The record in the IOAREA includes a halfword record length for variable length records.

The use of GSAM data sets in a checkpoint/restart environment is further discussed later in this chapter.

19.3 COBOL programming considerations

There are a few considerations that apply when you are coding DL/I programs in COBOL. Refer to Figure 72 for this discussion as the numbers between parenthesis in the text below refer to the corresponding code lines. Specific parameter values and formats are explained elsewhere throughout this chapter.

```
ID
DIVISION.                                000001

ENVIRONMENT DIVISION.                    000002
                                         000003
                                         000004
DATA DIVISION.                           000005
WORKING-STORAGE SECTION.                 000006
77      GU-FUNC          PIC XXXX  VALUE  'GU  '.          000007
77      GN-FUNC          PIC XXXX  VALUE  'GN  '.          000008
77      ERROPT           PIC XXXX  VALUE  '1    '.          000009
77      DERRID           PIC X(8)   VALUE  'DERROR01'.      000010
01      IOAREA           PIC X(256) VALUE  SPACES.          000011
01      SSA001-GU-SE1PART.                 000012
      02 SSA001-BEGIN     PIC X(19) VALUE  'SE1PART (FE1PGPNR =' . 000013
      02 SSA001-FE1PGPNR  PIC X(8) .          000014
      02 SSA001-END       PIC X     VALUE  ') ' .          000015
                                         000016
LINKAGE SECTION.                         000017
01      D1PC.                             000018
      02 D1PCDBN  PIC X(8) .                000019
      02 D1PCLEVL PIC 99 .                  000020
      02 D1PCSTAT PIC XX .                  000021
      02 D1PCPROC PIC XXXX .                000022
      02 D1PCRESV PIC S9(5) COMP.           000023
      02 D1PCSEGN PIC X(8) .                000024
      02 D1PCKFBL PIC S9(5) COMP.           000025
      02 D1PCNSSG PIC S9(5) COMP.           000026
      02 D1PCKFBA PIC X(20) .               000027
                                         000028
PROCEDURE DIVISION.                      000029
ENTRY 'DLITCBL' USING D1PC.              000030
:                                         000031
:                                         000032
CALL 'CBLTDLI' USING GU-FUNC, D1PC, IOAREA, 000033
      SSA001-GU-SE1PART.                 000034
:                                         000035
CALL 'CBLTDLI' USING GN-FUNC, D1PC, IOAREA. 000036
IF D1PCSTAT NOT = ' ' ,                  000037
      CALL 'ERRRTN' USING D1PC, DERRID, IOAREA, ERROPT. 000038
      MOVE +4 TO RETURN-CODE.             000039
:                                         000040
CALL DFSOAST USING D1PC.                  000041
:                                         000043
:                                         000044
GOBACK.                                   000045
```

Figure 72. COBOL batch program

- The DL/I function codes (7)(, IOAREA (11), and Segment Search Arguments (12) should be defined in the Working-Storage Section of the Data Division. Typically, either the IOAREA would be REDEFINED to provide addressability to the fields of each segment, or separate IOAREAs would be defined for each segment.
- The program Communication Blocks (PCBS) Should be defined in the Linkage Section of the Data Division (18). When there are multiple database structures (thus multiple PCBs) in a program, there must be one PCB defined in the Linkage Section for each PCB in the PSB. However, these PCBs need not be in any specific order.

- An ENTRY statement (30) should be coded at the entry to your program. A parameter of the USING clause should exist for each database structure (PCB) that is used in your program. The order of PCBs in this clause must be the same as specified in the Program Specification Block (PSB) for your program.
- Each DL/I CALL statement should be coded as in statement (33). The parameters of the DL/I call are explained elsewhere in this chapter, and differ in number for different functions.
- The status code in the PCB should be checked after each call (37). The status-code error routine is discussed below (38),
- At the end of processing, control must be returned to DL/I via a GOBACK statement (44). If you wish you may set the COBOL 'RETURN-CODE' (39). If DL/I detects no errors, and thus does not set the return code, the COBOL 'RETURN-CODE' value will be passed on to the next job step.

19.4 PL/I programming considerations

This section refers to Figure 73 as the numbers between parenthesis in the text refer to the corresponding code line.

When DL/I invokes your PL/I program it will pass the addresses, in the form of pointers, to each PCB required for execution. These will be passed in the same sequence as specified in PSB. To use the PCBs, you must code parameters in your PROCEDURE statement, and declare them to have the attribute POINTER.

In the example, DC_PTR and DB_PTR are specified in the PROCEDURE statement (6) and declared POINTER variables (15 and 16). These pointer variables should be used in declaring the PCBs as BASED structures (18 and 21), and in calling DL/I(55).

The format of the PL/I CALL statement to invoke DL/I (55) IS:

```
CALL PLITDLI (parmcount, function, pcb-ptr, io-area,ssal,...,ssan):
```

where:

parmcount

is the number of arguments in this call following this argument. It must have the attributes FIXED BINARY (31). See (38).

function

is the DL/I function code. It must be a fixed length character string of length 4. pcb-ptr is a pointer variable containing the address of the PCB. This is normally the name of one of the parameters passed to your program at invocation.

io-area

is the storage in your program into/from which DL/I is to store/fetch data. It can be a major structure, a connected array, a fixed-length character string (CHAR (n)), a pointer to any of these or a pointer to a minor structure. It cannot be the name of a minor structure of a character string with the attribute VARYING.

ssal,...

is one or more optional segment search arguments. Each SSA argument must be one of the same PL/I forms allowed for io-areas, described above. See (47) in the example.

Upon completion of your program, you should return either via a RETURN statement or by executing the main procedure END statement.

```
/*-----SAMPLE PL/I PROGRAM-----*/ /0000001
/*                                     */ /0000002
/*-----*/ /0000003
PE2PROD:                                0000005
PROCEDURE (DC_PTR,DB_PTR) OPTIONS (MAIN); 0000006
/* DECLARE POINTERS AND PCBS.           */ /0000008
DECLARE                                0000010
  PLITDLI ENTRY,                       /* DL/I WILL BE CALLD*/ 0000012
  DFSOAST ENTRY OPTIONS (ASSEMBLER INTER), /* STATISTICS PRINT */ 0000013
  DFSOER ENTRY OPTIONS (ASSEMBLER INTER), /* STATUS COOE PRINT */ 0000014
  DC_PTR POINTER,                      /* CHPAT IN PSB */ 0000015
  DB_PTR POINTER,                      /* ORDER DB PCB */ 0000016
  01 CLPC BASED (DC_PTR),              /* NOT USED IN */ 0000018
    02 DUMMY CHAR (32),                /* BATCH DL/I */ 0000019
  01 DLPC BASED (DB_PTR),              /* PHASE 2 ORDER DB */ 0000021
    02 DLPCBDN CHAR (8),               /* DBD NAME */ 0000022
    02 DLPCLEVL CHAR (2),              /* SEGMENT LEVEL */ 0000023
    02 DLPCSTAT CHAR (2),              /* STATUS CODE */ 0000024
    02 DLPCPROC CHAR (4),              /* PROCESSING OPTN */ 0000025
    02 OLPCRESV FIXED BINARY(31),      /* RESERVED */ 0000026
    02 DLPCSEGN CHAR (8),              /* SEGMENT NAME */ 0000027
    02 DLPCFBL FIXED BINARY(31),       /* KEY FEEBACK LNG */ 0000028
    02 DLPCNSSG FIXED BINARY(31),      /* NO. OF SENSEGS */ 0000029
    02 DLPCFBA CHAR (14);              /* KEY FEEDBACK */ 0000030
/* DECLARE FUNCTION COOES, I/O AREA, CALL ARG LIST LENGTHS */ /0000032
DECLARE                                0000034
  IO_AREA CHAR (256)                   /* I/O AREA */ 0000036
  GU_FUNC STATIC CHAR (4) INIT t'GU'I, /* CALL FUNCTION */ 0000037
  FOUR STATIC FIXED BINARY (31) INIT I4 ), /* ARG LIST LENGTH */ 0000038
  ERROPT1 CHAR (4) INIT ('0') STATIC, /* OPTN FOR DFSOER */ 0000039
  ERROPT2 CHAR (4) INIT ('2') STATIC, /* FINAL OPTN:DFSOER*/ 0000040
  DERRID CHAR (8) INIT ('DERFOR01') STATIC; /* ID FOR DFSOER */ 0000041
/* DECLARE SEGMENT SEARCH AFGUMENT (SSA) - ORDER SEGMENT. */ /0000043
DECLARE                                0000045
  01 SSA007_GU_SE2OPDER,                0000047
    02 SSA007_BEGIN CHAR (19) INIT ('SE2ORDER(FE2OGPEF =)'), 0000048
    02 SSA007_FE2OG2EF CHAR (6),        0000049
    02 SSA007_END CHAR (1) INIT ('1');  0000050
/* PROCESSING PORTION OF THE PROGRAM */ /0000052
SSACO7_FE2OGREF = 'XXXXXXX';           /* SET SSA VALUE */ 0000054
CALL PLITDLI (FOUR,GU_FUNC,.DB_PTR,IO_AREA, /* THIS CALL WILL */ 0000055
              SSA007_GU_FE2ORDER);          /* RETURN 'GE' STAT */ 0000056
IF DLPCSTAT -- ' ' THEN                 /* CALL EROOR PRINT */ 0000057
  CALL DFSOER (DLPC,DERRID,IO_AREA,ERROPT1); 0000058
  CALL DFSOER (DLPC,DERRID,IO_AREA,ERROPT2); /* FINAL CALL TO ERR*/ 0000059
/* RETURN TO CALLER.                    */ /0000065
END PE2PORD;                            0000067
```

Figure 73. PL/I batch program structure

19.5 Processing with logical relationships

Generally, there is no difference between the processing of physical databases and logical databases: all call functions are available for both. Some considerations do apply, however, when accessing a logical child of a concatenated segment.

19.5.1 Accessing a logical child in a physical database

When accessing a logical child in a physical DBD, you should remember the layout of the logical child. It always consists of the logical parent concatenated key (that is, all the consecutive keys from the root segment down to and including the logical parent) plus the logical child itself: the intersection data (see Figure 66 on page 181). This is especially important when inserting a logical child. You will also get an IX status code when you try to insert a logical child and its logical parent does not exist (except at initial load time). This will typically happen when you forget the LPCK in front of the LCHILD.

Note: In general, physical databases should not be used when processing logical relationships.

19.5.2 Accessing segments in a logical database

The following considerations apply for each call function when accessing segments in logical DBDs

19.5.2.1 Retrieve calls

These calls function as before with the same status codes. Remember, however, that the concatenated segment always consists of the logical child segment plus, optionally (dependent on the logical DBD), the destination parent segment (see Figure 69 on page 185).

19.5.2.2 Replace calls

In general, these calls function the same as before. When replacing a concatenated segment you may replace both the logical child segment and the destination parent. Remember, however, that you never can change a sequence field. The following sequence fields can occur in a concatenated segment (see also:

- Destination parent concatenated key.
- Real logical child sequence field, (that is, the sequence of the physical twin chain as defined for the real logical child). This field can (partially) overlap the logical parent concatenated key.
- Virtual logical child sequence field, (that is, the sequence of the logical twin chain as defined for the virtual logical child). This field can (partially) overlap the physical parent concatenated key.
- The key of the destination parent itself.

If any of the above fields is changed during a replace operation, a DA status code will be returned, and no data will be changed in the database.

19.5.2.3 Delete calls

In general, these calls function the same as before. If, however, you delete a concatenated segment (either of the two versions), only the logical child and its physical dependents (that is, the dependents of the real logical child) will be deleted. the destination parent can be deleted only via its physical path. In other words: "The delete is not propagated upwards across a logical relation." You can delete only those dependents of concatenated segments which are real dependents of the logical child. Examples:

- If the logical DBD of Figure 23 on page 75, a PART segment was deleted, the associated STOCK and DETAIL segments are deleted, too. However, the associated CUSTOMER ORDER and SHIPMENT segments remain.
- If the logical DBD of Figure 23 on page 75, a CUSTOMER ORDER segment was deleted, the associated DETAIL and SHIPMENT segments are deleted too. However, the associated PART and, STOCK segments remain.

Notice the logical child (and its physical dependents) is always deleted whenever one of its parents is deleted.

19.5.2.4 Insert calls

Whenever you insert a concatenated segment, the destination parent must already exist in the database. You can provide the destination parent together with the logical child in the IOAREA, but it is not used. Besides the normal status codes, an IX status code is returned when the destination parent does not exist.

19.6 Processing with secondary indices

Access segments via a secondary index allows a program to process segments in a order which is not the physical sequence of the database. One good example of this is the ORDER segment. To process an order when only the Customer order number is known, the ORDER segment can be access via the customer order number. This is the simplest from of secondary index.

Another basic use for a secondary index is to provide a method of processing a subset of the segments in a database without having to read the entire database. An example of this would be to provide a secondary index on a Balance owning field in the customer database. The secondary index database could be defined to only contain those database records for which a non-zero balance is owning.

19.6.1 Accessing segments via a secondary index

The format of the CALL parameters for accessing segments via a secondary index are identical to those access through the primary path. The difference is in the PCB coded in the PSB. The second PCB in the PSB in Figure 74 shows how to define a process using the secondary index.

19.6.1.1 Retrieving segments

The same calls are used as before. However, the index search field, defined by an XDFLD statement in the DBD will be used in the SSA for the get unique of the root segment. It defines the secondary processing sequence.


```

*
* PSB with Secondary index PCB
*
      PCB      TYPE=DB, PROCOPT=G,
              DBDNAME=BE2CUST, , KEYLEN=6

      PCB      TYPE=DB, PROCOPT=G,
              DBDNAME=BE2CUST, , PROCSEQ=FE2CNAM, , KEYLEN=20
*
              SENSEQ NAME=SE2PCUST

      PSBGENG, LANG=COBOL, PSBNAME=SE2PCUST, CMPAT=YES
      END

```

Figure 74. PSB with secondary index defined

After the successful completion of this get unique call, the PCB and ICAREA look the same as after the basic GU of Figure 63 on page 179, except that the key feedback area now starts with the customer name field.

When using the secondary processing sequence, consecutive get next calls for the CUSTOMER ORDER segment will present the CUSTOMER ORDER segments in customer name sequence.

If both the primary and the secondary processing sequence are needed in one program, you should use two PCBs as shown in Figure 75.

```

77  GU-FUNC                                PICTURE XXXX   VALUE 'Gubb'

01  SSA002-GU-SE2PCUST.
    02  SSA002-BEGIN                        PICTURE x(19) VALUE 'SE2PCUSTb(FE2PCNAMb=' .
    02  SSA002-FE2PCNAM PICTURE X(20) .
    02  SS1002-END                          PICTURE X      VALUE ') ' .

01  IOAREA                                PICTURE X(256) .

-----
MOVE CUSTOMER-NAME TO SSA002-FE2PCNAM.
CALL 'CBLTDLI' USING GU-FUNC, PCB-NAME, IOAREA, SSA002-GU-SE2PCUST.
-----

STATUS CODES:
-----
      bb:  succesfull call
      GE:  exceptional but correct condition
      other: error condition

```

Figure 75. GU call using a secondary index

19.6.1.2 Replacing segments

To replace segments in the indexed database a combination of get hold and replace calls can be used as before. Again, no sequence fields may be changed. The index search fields, however, can be changed. If an index search field is changed, DL/I will automatically update the index database via a delete old and insert new pointer segment.

Note: When using a secondary processing sequence, this could result in the later re accessing of a database record

19.6.1.3 Deleting segments

When using a secondary processing sequence, you cannot delete the index target segment (that is, the root segment). If you have a need to do so, you should use a separate PCB with a primary processing sequence.

19.6.1.4 Inserting segments

Again, when using a secondary processing sequence, you cannot insert the index target segment. In all other cases, the ISRT call will function as before.

19.6.2 Secondary index creation

A secondary index can be created during initial load of the indexed database or later. The secondary index database is created with the DL/I reorganization utilities. No application program requirements.

19.7 Loading databases

Loading databases with information has some considerations for the application program and the PSB used.

19.7.1 Loading a database

Basically the load program inserts segments into the database from some kind of input. It builds the segments and inserts them in the database in hierarchical order. Quite often the data to be stored in the database already exists in one or more files, but merge and sort operations may be required to present the data in the correct sequence.

The process of loading database is different than updating a database with segments already in the it. A database must be initialized before it can be used by most application programs. A database can be initialize in several ways:

- Data reloaded by the database recovery utility
- Data loaded by a database reload utility
- Data loaded by a program with the PROCOPT of L

Once the database is initialize it will remains so until it has been deleted and redefined. Therefore is it possible to have an empty initialize database. A database which is not empty can not be used by a PSB with a PROCOPT of L nor can it be recovered or loaded with the reload utility.

If the database has no secondary indices or logical relationship, then the load process is very straight forward. Any program with a PROCOPT of L can load it. Once that program has completed and close the database, the database can then be used by any program for read or update.

The loading of database with logical relationships and secondary indices are discussed next.

19.7.1.1 Loading a HIDAM database

When loading a HIDAM database initially, you must specify PROCPT=LS in the PCB. Also, the database records must be inserted in ascending root sequence, and the segment must be inserted in their hierarchical sequence.

19.7.1.2 Loading an HDAM database

When initially loading an HDAM database, you should specify PROCOPT=L in the PCB. There is no need for DL/I to insert the database records in root key order, but you must still insert the segments in their hierarchical order. For performance reasons it is advantageous to sort the database records into sequence. The physical sequence should be the ascending sequence of the block and root anchor point values as generated by the randomizing algorithms. This can be achieved by using a tool from the IMS/ESA System Utilities/Database Tools. This tool provides a sort exit routine, which gives each root key to the randomizing module for address conversion, and then directs SORT to sort on the generated address + root key value. Status Codes for Database Loading

The status codes, as shown in Table 15, can be expected when loading basic databases after the ISRT call:

Table 15. Database load status codes

Status Code Return	Explanation
bb or CK	Segment is inserted in database
LB	the segment already exists in database
IC	key field of segment is out of sequence
LD	no parent has been inserted for this segment in the database
other	error situation

Status code error routine

There are essentially two categories of error status codes: those caused by application program errors and those caused by system errors. Sometimes, however, a clear split cannot be made immediately.

This listing is not complete, but does contain all the status codes you should expect using our subset of DL/I. You should refer to Appendix B of the "IMS Application Programming Reference Manual," if you should need a complete listing of all possible status codes.

19.7.2 Loading databases with logical relationships

To establish the logical relationships during initial load of databases with logical relationships, DL/I provides a set of utility programs. These are necessary because the sequence in which the logical parent is loaded is normally not the same as the sequence in which the logical child is loaded. To cope with this, DL/I will automatically create a workflow whenever you load a database which contains the necessary information to update the pointers in the prefixes of the logically related segments.

Before doing so, the work file is sorted in physical database sequence with the prefix resolution utility (DFSURG10). This utility also checks for missing logical parents. Next, the segment prefixes are updated with the prefix update utility (DFSURGP0). After this, the database (s) are ready to use. The above database load, prefix resolution and update should be preceded by the prereorganization utility (DFSURPRO). This utility generates a control data set to be used by database load, DFSURG10 and DFSURGP0. Figure 76 illustrates the process.

If both any of the databases involved in the logical relationship also has secondary indices, then the process for loading a database with secondary indices must be used as well. See Figure 78 for an illustration of the complete process.

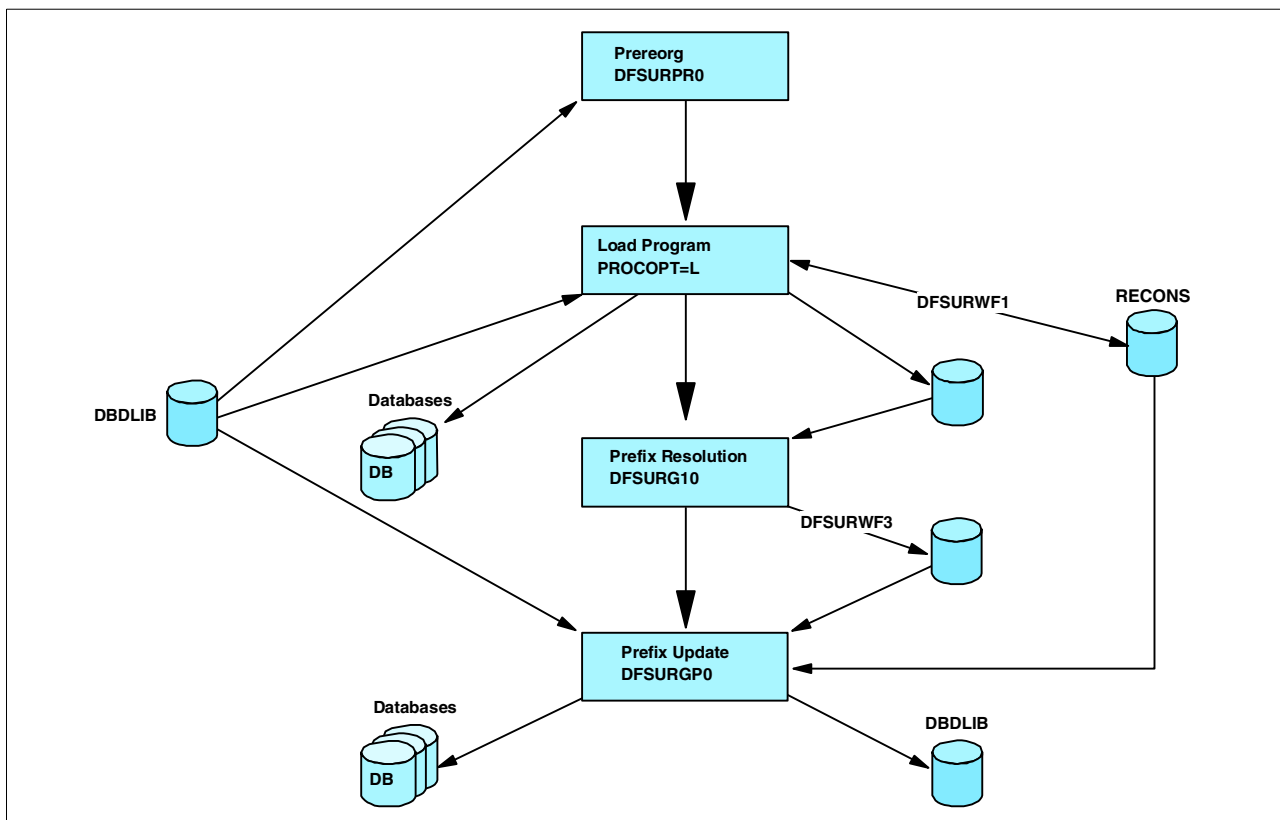


Figure 76. Database load with logical relationships

Notes:

1. You cannot use a logical DBD when initially loading a database (PROCOPT=L (S) in the PCB).
2. You must load all database involved in the logical relationship and pass the work files to the prefix resolution utility.

19.7.3 Loading a database with secondary indices

To load a database which has secondary indices the primary database must be uninitialized as shown in Figure 77. IMS will extract the required information into the work file to build the secondary index database(s).

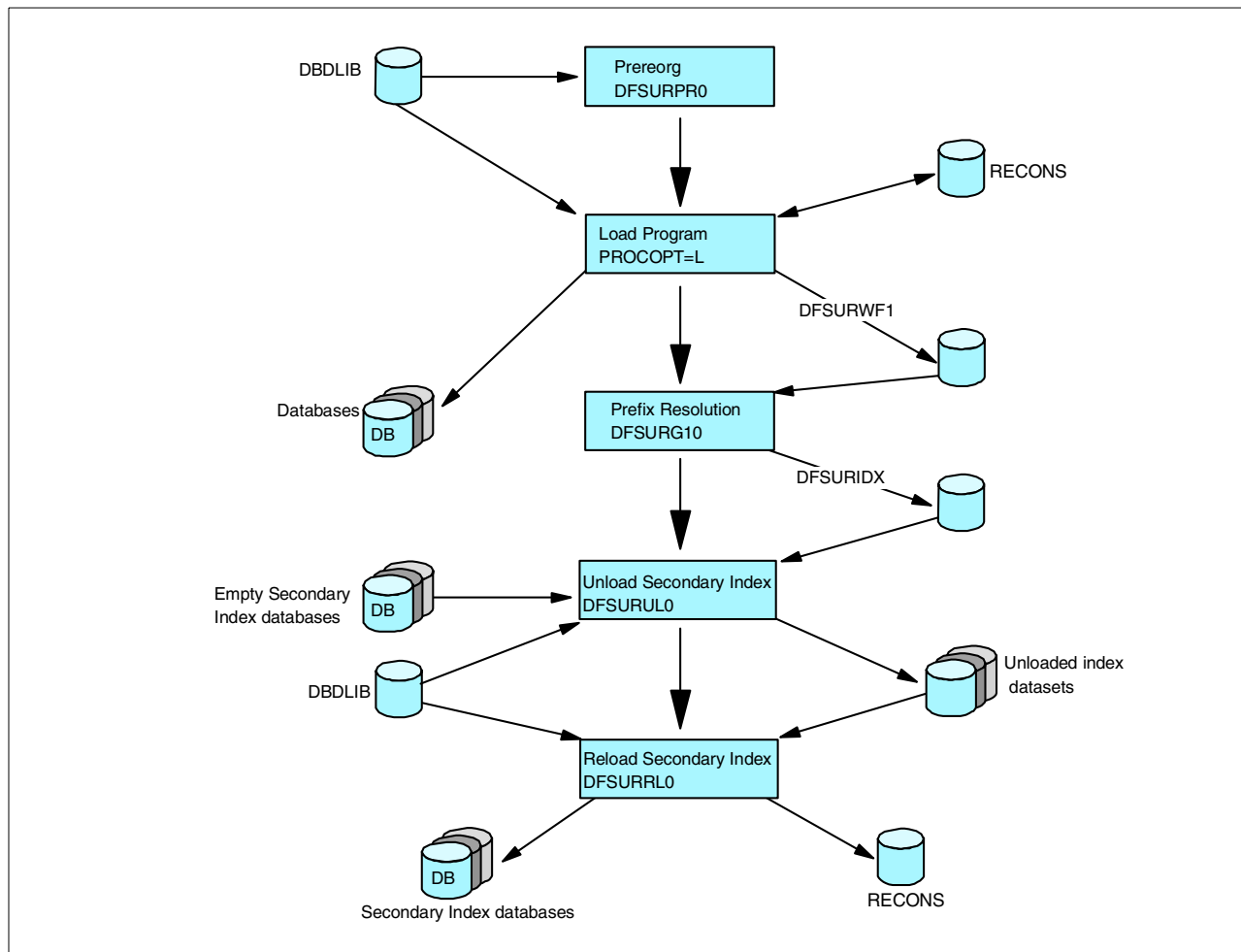


Figure 77. Database load with secondary indices

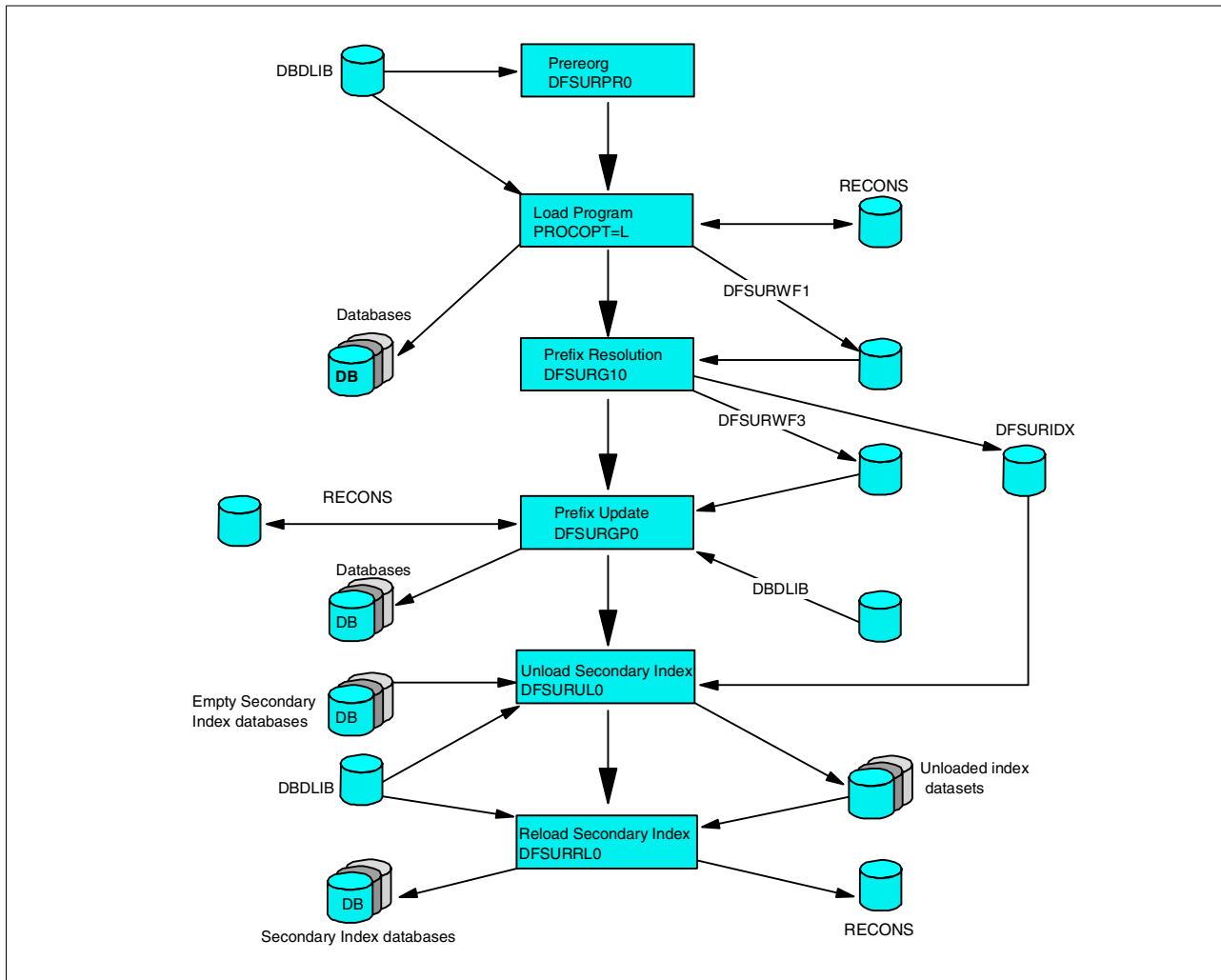


Figure 78. Database load with logical relationships and secondary indices

19.8 Batch checkpoint/restart

The batch checkpoint/restart facility of DL/I allows long running programs to be restarted at an intermediate point in case of failure. At regular intervals (CHKP calls) during application program execution, DL/I saves on its log data set, designated working storage areas in the user's program, the position of GSAM databases, and the key feedback areas of non-GSAM databases.

For each checkpoint, a checkpoint ID (message DFS681I) will be written to the MVS system console and to the job system output.

At restart, the restart checkpoint ID is supplied in the PARM field of the EXEC statement of the job. DL/I will then reposition the GSAM databases and restore the designated program areas. This is accomplished with a special restart call (XRST) which must be the very first DL/I call in the program. At initial program execution, the XRST call identifies the potential program areas to be checkpointed by later CHKP calls.

19.8.1 Using the XRST and CHKP calls

To utilize the checkpoint/restart function of DL/I for batch programs, you should consider the following guidelines:

1. All the data sets that the program uses must be DL/I databaseS. GSAM should be used for sequential input and output files, including SYSIN and SYSOUT. Any other file cannot be repositioned by DL/I and can result in duplicate or lost output.
2. The GSAM output data sets should use DISP=(NEW,KEEP,KEEP) for the initial run and DISP=(OLD,KEEP,KEEP) at restart (s).
3. SYSOUT should not be used directly. The output should be written to a GSAM file (as in 2) and be printed with the additional jobstep. IEBGENER can be used for this purpose.
4. The first call issued to DL/I must be XRST call. Its format will be discussed later.
5. The frequency of the checkpoint call is your choice. A basic recommendation is on checkpoint for every 50 to 500 update transactions. It is good practice to program for an easy adjustment of this frequency factor.
6. After each checkpoint call, you must reposition yourself in the non-GSAM databases by issuing a get unique call for each of those databases. Repositioning of GSAM databases is done by DL/I, and you should proceed with a get next (input) or an insert (output) call.

19.8.1.1 The restart call

Upon receiving the restart call (XRST), DL/I checks whether a checkpoint ID has been supplied in the PARM field of the EXEC card or in the work area pointed to by the XRST call. If no ID has been supplied, a flag is set to trigger storing of repositioning data and user areas on subsequent CHKP calls (that is, DL/I assumes that this is the initial program execution, not a restart).

If the checkpoint at which restart is to occur has been supplied, the IMS batch restart routine reads backwards on the log defined in the //IMSLOGR DD card to locate the checkpoint records. User program areas are restored.

The GSAM databases active at the checkpoint are repositioned for sequential processing. Key feedback information is provided in the PCB for each database active at the checkpoint. The user program must reposition itself on all non-GSAM databases, just as it must do after taking a checkpoint.

The format of the XRST call is:

COBOL:

```
CALL 'CBITDLI' using call-func,IOPCB-name, I/O-area-len,work-area  
[,1st-area-len, 1st rea,...,nth-area-len,nth-area].
```

PL/I:

```
CALL PLITDLI (parmcount,call-func,IOPCB-name. I/O-area-len,work-ar  
[,1st-area-len,1st-area,...,nth-area-len,nth-area]) :
```

ASSEMBLER:

CALL

ASMTDLI, (call-func, IOPCB-name, I/O-area-len, work-area[, 1st-area-len, 1st-area
, ..., nth-area-len, nth-rea]),

where:

parmcount

is the name of a binary fullword field containing the number of arguments following. PL/I only.

call-func

is the name of a field which contains the call function 'XRST'.

IOPCB-name

is the name of the I/O PCB or the "dummy" I/O PCB supplied by the CMPAT option in PSEGEN (C1PCB in the sample programs).

I/O-area-len

is the name of the length field of the largest I/O area used by the user program: must be a fullword. work-area is the name of a 12 byte work area. This area should be set to blanks (X'40') before the call and tested on return. If the program is being started normally, the area will be unchanged. If the program is being restarted from checkpoint, the ID supplied by the user in that CHKP call and restart JCL will be placed in the first 8 bytes. If the user wishes to restart from a checkpoint using the method other than IMS Program Restart, he may use the XRST call to reposition GSAM databases by placing the checkpoint ID in this area before issuing the call. This ID is the 8-byte left-aligned, user supplied ID.

1st-area-len

is the name of a field which contains the length of the first area to be restored: must be a fullword.

1st area

is the name of the first area to be restored

nth-area-len

is the name of a field which contains the length of the nth area to be restored (max n=7): must be a fullword. nth-area is the name of the nth area to be restored (max n=7).

Notes:

1. The number of areas specified on the XRST call must be equal to the maximum specified on any CHKP call.
2. The lengths of the areas specified on the XRST call must equal to or larger than the lengths of the corresponding (in sequential order) areas of any CHKP call.
3. The XRST call is issued only once and it must be the first request made to DL/I.
4. The only correct status code is bb: any other implies an error condition.

5. All “area-len” fields in PL/I must be defined as substructures. The name of the major structure should, however, be specified in the call.

Example:

```
DCL 1 i/c-area-len,  
    2 L NTH FIXED BIN (31) INT (length):
```

19.8.1.2 The checkpoint call

When DL/I receives a CHKP call from a program which initially issued a XRST call, the following actions are taken:

- All database buffers modified by the program are written to DASD.
- A log record is written, specifying this ID to the OS/VS system console and job sysout.
- The user-specified areas (for example, application variables and control tables) are recorded on the DL/I log data set. They should be specified in the initial XRST call.
- The fully-qualified key of the last segment processed by the program on each DL/I database is recorded on the DL/I log dataset.

the format of the CHKP call is:

COBOL:

```
CALL 'OBLTDLI' using call-func,IOPCB-name, I/O-area-len,I/O=area  
[,1st-area-len,1st-area,...,nth-area-len,nth-area]).
```

PL/I:

```
CALL PLITDLI [parmcount, call-func,IOPCB-name,I/O-area-len, I/O=area  
[,1st-area-len,1st-area,...,nth-area-len,nth-area]):
```

ASSEMBLER:

```
CALL ASMTDLI, (call-func,IOPCB-name,I/O-area-len,I/O=area  
[,1st-area-len,1st-area,...,nth-area-len,nth-area]):
```

parmcount

is the name of a binary fullword field containing the number of arguments following: PL/I only.

call-func is the name of a field with the call function “CHKP”.

IOPCB-name is the name of the I/O PCB or dummy PCB in batch.

I/O-area-len

is the name of the length field of the largest I/O area used by the application program: must be a fullword.

I/O-area

is the name of the I/O area. The I/O area must contain the 8 byte checkpoint ID. This is used for operator or programmer communication and should consist of EBCDIC characters. In PL/I, this parameter should be specified as a pointer to a major structure, an array, or a character string.

Recommended format:

MMMMnnnn

MMMM= 4 character program identification

nnnn = 4 checkpoint sequence number, incremented at each CHKP call.

1st-area-len (optional)

is the name of a field that contains the length of the first area to checkpoint:
must be a fullword.

1st-area (optional)

is the name of the first area to checkpoint

nth-area-len (optional)

is the name of the field that contains the length of the nth area to checkpoint
(max n=7): must be a fullword.

nth-area (optional)

is the name of the nth area to checkpoint (maxn n=7)

Notes:

1. The only correct status code in batch is bb: any other specifies an error situation.
2. Before restarting a program after failure, you always must first correct the failure and recover your databases. You must reestablish your position in all IMS database (except GSAM) after return from the checkpoint (that is, issue a get unique).
3. All "area-len" fields in PL/I must be defined as substructures see the example under note 5 of the XRST call.
4. Because the log tape is read forward during restart, the checkpoint ID must be unique for each checkpoint.

Part 5. IMS system administration

This part contains five chapters:

- A discussion of database recovery control (DBRC). Refer to Chapter 20, “Database recovery control (DBRC)” on page 207.
- A discussion of RECON record types. Refer to Chapter 21, “RECON record types” on page 221.
- A discussion of IMS logging. Refer to Chapter 22, “IMS logging” on page 239.
- A discussion of the IMS system generation process. Refer to Chapter 23, “IMS system generation process” on page 245.
- A discussion of IMS security. Refer to Chapter 24, “IMS security overview” on page 253.

Chapter 20. Database recovery control (DBRC)

DBRC includes the IMS functions which provide IMS system and database integrity and restart capability.

DBRC records information in a set of VSAM data sets called RECONS. Two of these RECONS are a pair of VSAM clusters which work as a set to record information. A third RECON can be made available as a spare. IMS normally works with two active RECONS. If one becomes unavailable, the spare will be activated if it is available.

IMS records the following information in the RECON:

- Log data set information
- Database data set information
- Event information
 - Allocation of a database
 - Update of a database
 - Image copy of a database
 - Abend of a subsystem
 - Recovery of a database
 - Reorganization of a database
 - Archive of a OLDS data set

20.1 DBRC usage

There are three aspects to DBRC usage, as discussed below.

20.1.1 DBRC options

The first option is whether the DBRC function is active in address spaces executing IMS (controlled by parameters on the IMSCTRL macro, and overrides at execution), whether databases must be registered in the RECON (controlled by FORCER PARM on RECON header) and the level of DBRC functions offers (controlled by SHARECTL/RECOVCTL on RECON header).

1. DBRC is always active in an IMS control region (DBCTL/DCCTL/DBDC). It is required for log archive management, at least. Two sub parameters on the DBRC= parameter of the IMSCTL macro in the IMSGEN control DBRC usage in other environments -
 - FORCE - this forces DBRC usage in all other address spaces. It cannot be overridden in the JCL. Any job attempting to run with DBRC=N abends. There are also YES,NO options, but these are only valid for a Batch IMSGEN, not a DBCTL IMSGEN, for DBCTL you must have DBRC support generated (even if its not forced for batch).
 - YES/NO - this sets the default for DBRC usage for batch execution, it can be overridden at execution time on the DBRC EXEC parm (unless, of course, you defined DBRC=FORCED).

A BMP does not have a DBRC usage parameter, its always active in the DBCTL it connects to.

The above parameters only control whether DBRC is active in an address space, the level of functions available is controlled by parameters in the RECON header.

2. The FORCER/NOFORCER option on the RECON header controls whether database:hp2.Must:ehp2. be registered in the RECON.
 - If NOFORCER is specified, databases may, or may not be registered in the RECON. If a database is not registered in the RECON, and DBRC is active, you get a warning message each time the database is opened.
 - If FORCER is specified, then, if DBRC is active in the address space, all databases must be registered in the RECON, if not DBRC rejects authorization and the job abends (or PSB fails to schedule in the DBCTL, the DBCTL stays up). There is not much point in using this if you plan to run regularly with DBRC=N, as you end up with an incomplete record of updates, which is unless for recovery purposes.

If a database is registered in the RECON, and you run a job with DBRC=N, the next time you run a job with DBRC=Y a warning message is issued flagging the fact the database has been accessed outside of DBRC control (normal suggestion is to take an Image copy at that point).

3. The SHARECTL option on the RECON header controls the level of information stored in the RECON (and the checks performed when you run a job).
 - If RECOVCTL is set, then all online IMS logs, and all batch logs for jobs that have executed DBRC=Y are recorded in the RECON. Additionally, if a database is registered in the RECON, all allocations (online and by batch jobs executing DBRC=Y), links to the corresponding logs, change accumulations, image copies, reorganizations and recoveries are recorded in the RECON. This gives you (providing all batch jobs execute DBRC=Y) a complete history of DB access. You can then use DBRC to generate recovery jobs. IMS version 6 does not support RECOVCTL. RECOVCTL is available only in version 5.
 - If SHARECTL is specified, you get all the above, but in addition DBRC will also ensure only one address space access a database for update at any one time (providing the database is registered, and all jobs run DBRC=Y). It allows multiple accesses by jobs with read intent (PROCOPT=G), or one updater, plus multiple access by read without integrity (PROCOPT=GO). DBRC will also prevent other address spaces accessing a database that has outstanding backout action required (after address space failure).

20.1.2 DBRC configurations

How much DBRC support you use depends, to some degree, on the expectations placed upon the DBA for database recovery.

1. If a DBA is always expected to recover databases, then a fairly tight configuration should be used.

IMS should be generated with DBRC usage forced in batch, and all databases should be registered in the RECONS.

A separate IMSGEN should be run, with DBRC not forced, and kept in a library only accessible to Systems Programming/DBA for use as last resort to correct things.

2. If the application support personnel can be trusted to take correct recovery actions, and are prepared to lose databases to last backup if they become corrupt, then a slightly looser set up can be used.

The IMSGEN does NOT have DBRC usage forced, but the default is DBRC Y. Databases are defined in the RECON.

3. If databases are regularly copied around between environments, and DBRC is causing problems with this, then the second option can be used, but the copy databases not registered in the RECONs.

If they become corrupt, they are restored by copying again.

20.1.3 Database authorization

A DBRC sharing environment introduces a new concept of database authorization. This process determines if a subsystem (IMS online or IMS batch) can have access to the requested databases. DBRC authorizes or refuses to authorize the databases depending on the current authorizations and the access intent of the subsystem.

20.1.4 Access intent

Access intent is determined by DBRC when a subsystem tries to allocate a database:

- For a batch job, DBRC uses the processing option (PROCOPT) of the PSB for each database to determine the access intent. If the PSB has multiple PCBs for the same database, the highest intent for that database is used.
- For an IMS DC online system, the ACCESS parameter of the DATABASE macro sets the access intent.

There are four processing intent attributes. They are listed below in reverse order, from the highest access intent (the most restrictive), to the lowest (the least restrictive):

1. Exclusive (EX)

The subsystem requires exclusive access of the database and no sharing is allowed regardless of the share options registered in DBRC.

- PROCOPT of L or xE (batch) (where x = A,D,G,I,D)
- ACCESS of Ex (online)

2. Update (UP)

The subsystem may update the database. Even if no updates actually take place the database is held in update mode. Any logs created with actual changes during this process are required for recovery or change accumulation.

- PROCOPT of A,I,R,D (batch)
- ACCESS of UP online

3. Read with integrity (RD)

The subsystem only reads the database but it also checks any enqueue or lock held by other subsystems. It waits the lock to be released before processing.

- PROCOPT of G (batch)
- ACCESS of RD (online)

4. Read without integrity (RO)

The subsystem only reads the database and it does not check any lock or enqueue held by other subsystems.

- PROCOPT of GO (batch)
- ACCESS of GO (online)

20.2 RECON data sets

The RECON data set is the most important data set for the operation of DBRC and data sharing. The RECON data set holds all resource information and event tracking information used

The RECON data set can consist of one, two, or three data sets:

1. The original data set
2. The copy of the original data set
3. The spare data set

Important: The best solution, from an availability point of view, is to use all three data sets. This is strongly recommended. Using three data sets for the RECON causes DBRC to use them in the following way:

- The first data set is known as copy1. It contains the current information. DBRC always reads from this data set, and when some change has to be applied, the change is written database first to this data set.
- The second data set is known as copy2. It contains the same information as the copy1 data set. All changes to the RECON data sets are applied to this copy2 only after the copy1 has been updated.
- The third data set (the spare) is used in the following cases:
 - A physical I/O error occurs on either copy1 or copy2.
 - DBRC finds, when logically opening the copy1 RECON data set, that a spare RECON has become available, and that no copy2 RECON data set is currently in use.
- The following command is executed:

```
CHANGE.RECON REPLACE(RECONn)
```

When the third RECON data set is used, the remaining valid data set is copied to the spare. When the copy is finished the spare becomes whichever of the data sets was lost, missing or in error.

Note: From the RECON point of view, the copy1 and the copy2 are normally identified by a 1 or a 2 in a field of the RECON header information.

20.2.1 Database related information

A database and its associated data sets should only be defined in one RECON data set.

The fundamental principle behind the RECON data set is to store all recovery related information for a database in one place. It is not possible to use multiple RECON data sets in the recovery processing for the same database.

20.2.2 Subsystem

An IMS subsystem can only be connected to one set of RECON data sets.

All databases that are accessed by IMS DC subsystems under the control of DBRC must be registered in the RECON referenced by the online subsystem only if the RECON has the FORCER option set on.

All batch subsystems that access any database accessed by the online subsystem should reference the same RECONS referenced by the online subsystem.

20.2.3 Database name

The database names (DBD names) defined in one RECON data set must all be unique.

The database records, stored in the RECON data set, are registered with a key based on the DBD name. Therefore, DBRC cannot be used to control both test and production databases, using the same RECON data sets, unless some naming convention is adopted.

The rule can be simplified as follows. More than one set of RECON data set is necessary if all the following conditions are true:

1. Multiple versions of the same database exist (for example, test and production).
2. The same DBD name is used for the different versions of the database.
3. More than one version of the databases can be used by only one can be registered to DBRC in the RECON data set. The other are treated as not registered (unless FORCER has been set in the RECON).

The application of the previous rules usually results in the need for at least two different sets of RECON data sets, one shared between the production subsystems and one for the test subsystems.

Note: On the INIT.DBDS command, which is used to create the database data set record in the RECON, the user must supply the database data set name (DSN). When IMS opens the database, DBRC checks the DSN against the name registered in the RECON. If this name does not match, DBRC treats this database as if it was not registered. In this case, the test database—with a DSN different than the production database, even if with the same DBD name and data set name—can coexist with the production environment, but not under the control of the DBRC.

20.2.4 RECON definition and creation

The RECON data sets are VSAM KSDSs. They must be created by using the VSAM AMS utilities.

The same record size and CI size must be used for all the RECON data sets.

The RECON data sets should be given different FREESPACE values so that CA and CI splits do not occur at the same time for both active RECON data sets.

For availability, all three data sets should have different space allocation specifications. The spare data set should be at least as large as the largest RECON data set. Figure 79 shows an example of the RECON data set definition that was used to define the RECON for the hands-on.

```
DELETE STIMS220.RECONB

SET LASTCC=0

DEFINE CLUSTER (NAME (STIMS220.RECONB) -
              VOLUMES (SBV010) -
              INDEXED -
              KEYS (24 0) -
              CYLINDERS C5 2) -
              RECORDSIZE (128 32600) -
              SPANNED -
              FREESPACE (30 80) -
              CISZ(4096) -
              NOREUSE -
              NERAS SPEED REPL IMBD -
              UNORDERED -
              SHAREOPTIONS (3 3)) -
INDEX (NAME (STIMS220.RECONB.INDEX)) -
DATA (NAME (STIMS220.RECONB.DATA))
```

Figure 79. Example of RECON data set definition

20.3 Initializing RECON data sets

After the RECON data sets are created, they must be initialized by using the **INIT.RECON** command of the DBRC recovery control utility. This causes the RECON header records to be written in both current RECON data sets.

The RECON header records must be the first records written to the RECON data sets because they identify the RECON data sets to DBRC.

When the **INIT.RECON** command is used to initialize the RECON, specify either the RECOVCTL parameter or the SHARECTL parameter to select either:

- Recovery control environment
- Share control environment

Once one of these environments has been selected, it applies to all databases. IMS Version 6 only support the share control environment.

20.4 Allocation of RECON data sets to subsystems

To allocate the RECON data set to an IMS subsystem, the user must choose one of the following two ways:

- Point to the RECON data sets by inserting the DD statements in the start-up JCL for the various subsystems.
- Use dynamic allocation.

If a DD statement is specified for RECON, DBRC does not use dynamic allocation. Otherwise, DBRC uses dynamic allocation.

With multiple subsystems sharing the same databases and RECON data sets, dynamic allocation for both the RECON data sets and the associated databases should be used. This ensures that:

- The correct and current RECON data sets are used.
- The correct RECON data sets are associated with the correct set of databases.

It also allows recovery of a failed RECON data set, since DBRC dynamically de-allocates a RECON data set if a problem is encountered with it.

To establish dynamic allocation, a special member naming the RECON data sets must be added to IMS RESLIB or to an authorized library that is concatenated to IMS RESLIB. This is done using the IMS DFSMDA macro. Figure 80 shows an example of the required macros for dynamic allocation of the RECON data sets.

```
//DYNALL JOB. .  
//STEP      EXEC IMSDALOC  
//SYSIN     DD  *  
DFSMDA TYPE=INITIAL  
DFSMDA TYPE=RECON,DSNAME=PROD.RECON01,  
          DDNAME=RECON1  
DFSMDA TYPE=RECON,DSNAME=PROD.RECON02,  
          DDNAME=RECON2  
DFSMDA TYPE=RECON,DSNAME=PROD.RECON03,  
          DDNAME=RECON3
```

Figure 80. Dynamic allocation of RECON data sets

RECON data sets are always dynamically allocated with DISP=SHR specified.

When using multiple RECON data sets (for example, test and production), be sure that each subsystem uses the correct RECON data set group. This can be done by altering the SYSLMOD DD in the procedure IMSDALOC to place the dynamic allocation parameter lists for the various RECON data set groups in different IMS RESLIBs. The appropriate RESLIB or concatenated RESLIBs must be included for each subsystem start-up JCL.

Important: When multiple processors are accessing the same RECON data set, the dynamic allocation parameter lists must be kept synchronized in the IMS RESLIBs being used by the different processors. This does not happen automatically.

Important: The usage of dynamic allocation in some subsystems and JCL allocation in others is not recommended.

20.5 Placement of RECON data sets

The placement of the RECON data sets in the DASD configuration is very important. The primary rule is to configure for availability. This means, for example, to place all three RECON data sets on:

- Different volumes
- Different control units
- Different channels
- Different channel directors

20.6 RECON data set maintenance

There are several procedures and commands that can be used to maintain the RECON data set.

20.6.1 RECON backup

Operational procedures should be set up to ensure that regular backups of the RECON data set are taken.

These backups should be performed using the BACKUP.RECON DBRC utility command. The command includes a reserve mechanism to ensure that no updating of the RECON takes place during the backup. If possible, the backup should be taken when there are no subsystems active.

The backup copy is created from the copy1 RECON data set. The command to create the backup copy invokes the AMS REPRO command, with its normal defaults and restrictions. For instance, the data set that is receiving the backup copy must be empty.

20.6.2 DELETE.LOG INACTIVE command

The only recovery related records in the RECON data set that are not automatically deleted are the log records (PRILOG and LOGALL). These records can be deleted using the DELETE.LOG INACTIVE command. This command can be added to the job that takes a backup of the RECON data set.

A log is considered inactive when the following conditions are all true:

- The log volume does not contain any DBDS change records more recent than the oldest image copy data set known to DBRC. This check is performed on a database data set (DBDS) basis.
- The log volume was not opened in the last 24 hours.
- The log has either been terminated (nonzero stop time) or has the ERROR flag in the PRILOG and SECLOG record set on.

20.6.3 LIST.RECON STATUS command

Regular use should be made of the LIST.RECON STATUS command to monitor the status of the individual RECON data sets.

Using the LIST.RECON command produces a formatted display of the contents of RECON. The copy1 RECON data set is used as a source. DBRC ensures that the second RECON data set contains the same information as the first RECON data set.

The optional parameter **STATUS** can be used to request the RECON header record information and the status of all RECON data sets. The use of this parameter suppresses the listing of the other records.

This command should be executed two or three times a day during the execution of an online system, to ensure that no problems have been encountered with these data sets.

20.7 RECON reorganization

In addition to the regular backups, procedures to monitor utilization of the RECON data sets space should be put in place.

Since all current levels of VSAM support CI reclaim (and DBRC does not turn it off), the requirement to reorganize RECONs to reclaim space has diminished. For instance, when all the records in a CI have been erased, the CI is returned to the free CI pool in the CA. Some users have decided to perform a monthly reorganization.

A plan for reorganizing the RECON data sets to reclaim this space on a regular basis must be considered. The RECON data sets can be reorganized while the IMS online systems are active

20.8 Reorganizing RECON data sets

The RECON data sets can be reorganized easily and quickly with the use of a few DBRC and AMS commands. The AMS REPRO command copies one RECON data set to another, reorganizing it during the process. This command, combined with a DELETE and a DEFINE of the RECON data sets, is enough to complete a reorganization.

Additional information to consider when designing the RECON reorganization procedures, related to the IMS DC status, are as follows:

- If the online system is *active*:

A reorganization of the RECON data sets should be scheduled:

- During a period of low RECON activity.
 - When no BMPs are running.
 - A LIST.RECON STATUS command must be issued from each online system which uses the RECON data sets, after the CHANGE.RECON REPLACE command is issued, in order to de-allocate the RECON before deleting and defining it again.
- If the online system is *not active*:

A reorganization of the RECON data sets should be scheduled:

- After a BACKUP.RECON has been taken.
- When no subsystems are allocating the RECON data sets.

20.9 Recreating RECON data sets

The RECON data sets may need to be recreated, for instance:

- In a disaster recovery site
- After the loss of all the RECON data sets when no current backup is available

Recreating the RECON can be a long and slow process. When designing procedures to handle this process, there are two basic alternatives:

- Restore the RECON from the last backup (if available) and update it to the current status required.
- Recreate and re initialize the RECON data sets.

Both of these procedures have advantages and disadvantages. Which alternative is best suited for an installation depends on:

- The time frame in which the system must be recovered and available
- The point-in-time to which it is acceptable to recover
- The type of processing environment (24 hours online availability or batch)

Further details for restoring RECON data sets:

Before deciding to recreate the RECON data sets from scratch, the following details must be well understood:

- GENJCL functions are normally used to create procedures.

Without the RECON information, recovery procedures cannot be generated until the RECON information is correct. Likewise, image copy procedures cannot be generated until the database and image copy data set information has been recreated.

- Recreation of DB, DBDS, DBDSGRP and CAGRP information must be available.

If the original INIT commands were retained, then the registration can be done easily. Changes made with CHANGE commands must somehow be recorded and reapplied.

The DBDSGRP and CAGRP information is critical because any recovery image copy or change accumulations JCL generated can cause serious problems if incorrectly specified.

- Volume serial information is available.

Unless cataloged data sets are used, the volume serials of all image copy and log data sets must be corrected.

- Image copy time must be adequate.

If the databases are restored with non-IMS utilities (pack restores), then the time required to take an image copy or to notify DBRC of the image copy data sets, also restored with pack restores, must be considered.

In summary:

- For those installations using only Log control, it is probably easier to re initialize the RECON data sets and reapply the information than to update the RECON with the changed information.
- For those installations using Recovery or Share control where the physical restoration of the databases is done outside of the DBRC control, it might be easier to re initialize the RECON data sets.
- For those installations which require the online subsystem to be warm restarted, the only alternative is to use the latest backup of the RECON and to bring all information current to the required point-in-time.

20.10 PRILOG record size

One PRILOG record is created for each subsystem execution. This record must contain all the information about the log data sets created during the life of this subsystem.

The record size can be large if spanned records are used; however, the following limitations should be considered before using spanned records:

- The maximum size of a record to be used by the VSAM REPRO command is 32,760 bytes if the output is a non-VSAM data set.
- RECON backup and transfer to off-site storage is normally performed with a sequential data set.
- PRILOG records are only deleted when every RLDS and SLDS data set within that record is no longer required.

This is a problem only for those installations which have a high volume of log data sets and the requirement for a continuous operation environment.

To calculate the size of the maximum required PRILOG record, the formula in Figure 81 can be used.

$$S = 52 + (120 D) + (32 V)$$

where:

S = the size for the PRILOG/PRI SLDS record in bytes

52 = the required prefix of the PRILOG record

120 = the required number of bytes for each SLDS/RLDS entry

D = the number of SLDS/RLDS data sets created from archive for this execution of the subsystem

32 = the required number of bytes for each volume that contains SLDS/RLDS data sets

V = the number of volumes that can contain SLDS/RLDS data sets

Figure 81. PRILOG record size calculation formula

For example, assume that an installation has the following characteristics:

- An online subsystem is running for 23 hours a day.
- The subsystem fills up an OLDS every 30 minutes.
- Each OLDS is archived to one RLDS and one SLDS.
- There are 2 volumes that can contain RLDS or SLDS data sets.
- There are 46 RLDS and 46 SLDS data sets each day.

Using the formula in Figure 82, the size of the PRILOG record for this example is:

$$\begin{aligned} S &= 52 + (80 D) + (14 V) \\ &= 52 + (80 \ 92) + (14 \ 2) \\ &= 52 + (7360) + (28) \\ S &= 7440 \end{aligned}$$

Figure 82. PRILOG record size calculation formula example 1

This is well under the maximum size, so there is no problem with this subsystem.

Assume, however, that the environment changes to allow the IMS to run 24 hours a day for 6 days before being brought down. There are 48 RLDS and 48 SLDS data sets each day, and a total of 576 for the 6 days. The calculation now becomes like Figure 83.

$$\begin{aligned} S &= 52 + (80 D) + (14 V) \\ &= 52 + (80 \ 576) + (14 \ 2) \\ &= 52 + (46,080) + (28) \\ S &= 46,160 \end{aligned}$$

Figure 83. PRILOG record size calculation formula example 2

This is now over the suggested maximum record size. One solution is to switch to archiving after two OLDS are full. This reduces the number of RLDS and SLDS data sets by half. This brings the PRILOG record size well below the maximum size.

20.11 Summary of recommendations for RECON data sets

- Use three RECON data sets — two current and one spare.
- Define the three RECON data sets with different space allocations.
- Put the RECON data sets on different devices, channels, and so on.
- Use dynamic allocation.
- Do not mix dynamic allocation and JCL allocation.
- Define the RECON data sets for AVAILABILITY, but keep performance implications in mind.

Chapter 21. RECON record types

This chapter lists the record types that can be found in the RECON data sets, and for each record, explains its purpose and its relationship with other record types.

The relationship is never imbedded in the records like a direct pointer, but can be built by DBRC using the information registered in each record type. This allows constant access of the related records through their physical keys.

21.1 RECON records

There are six general classes of RECON record types:

1. Control records
2. Log records
3. Change accumulation records
4. Database data set (DBDS) group records
5. Subsystem records
6. Database records

21.1.1 Control records

Control records are used for controlling the RECON data set and the default values used by DBRC. This class of records includes:

- RECON Header record
- RECON header extension record

21.1.2 Log records

Log records are used for tracking the log data sets used by all subsystems. This class of records includes:

- PRILOG and SECLOG records (including interim log records)
- LOGALL record
- PRIOLD and SECOLD records (including interim OLDS records)
- PRISLDS and SECSLDS records (including interim SLDS records)

21.1.3 Change accumulation records

Change accumulation records are used for tracking information about change accumulation groups. This class of records includes:

- Change accumulation group records
- Change accumulation execution records
- Available change accumulation data set records

21.1.4 DBDS group records

Database Data Set Group (DBDSGRP) records are used to define the members of a DBDS group. The only record type in this class is a DBDS group record.

21.1.5 Database records

Database records are used to track the state of:

- Databases
- DBDSs
- Resources required for recovery of DBDSs

This class of records includes:

- Database record
- AREA authorization record
- DBDS record
- Allocation record
- Image copy record
- Reorganization record
- Recovery record

21.2 RECON header record

The header is the first record registered in the RECON data set by the INIT.RECON command as shown in Figure 84.

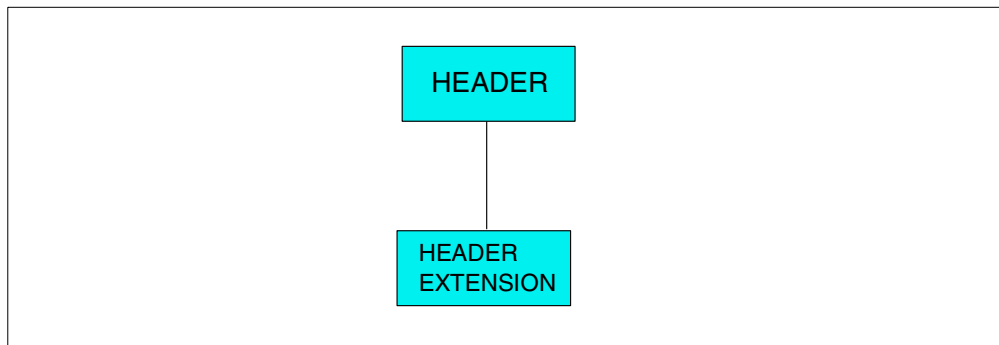


Figure 84. HEADER record

The header identifies the data set as a RECON data set and keeps information related to the whole DBRC system. It also controls the concurrent update of the RECON data set by several subsystems. The information kept in this record is read when the RECON is opened and the values are placed in various control blocks. Hence, the default values are accessible to other DBRC routines without additional I/O operations to the RECON data set.

The RECON header record is related to the RECON header extension record.

21.3 RECON header extension record

The RECON header extension record identifies the individual RECON data sets. It is also used in the synchronization process of the two primary RECON data sets. It is created by the INIT.RECON command, together with the RECON header record.

The RECON header extension record is related to the RECON header record, as shown in Figure 85.

```

RECON
RECOVERY CONTROL DATA SET, IMS/ESA V6R1          COEXISTENCE ENABLED
DMB#=27                      INIT TOKEN=99181F1848059F
FORCER    LOG DSN CHECK=CHECK17    STARTNEW=NO
TAPE UNIT=3490    DASD UNIT=3390    TRACEOFF    SSID=IMSY
LIST DLOG=NO                      CA/IC/LOG DATA SETS CATALOGED=YES
LOG RETENTION PERIOD=00.001 00:00:00.0

TIME STAMP INFORMATION:

TIMEZIN = %SYS

OUTPUT FORMAT:  DEFAULT = LOCORG NONE    PUNC YY
                CURRENT = LOCORG NONE    PUNC YY

-DDNAME-      -STATUS-      -DATA SET NAME-
RECON1        COPY1         IMS.SJIMSY.RECON1
RECON2        COPY2         IMS.SJIMSY.RECON2
RECON3        SPARE         IMS.SJIMSY.RECON3

```

Figure 85. HEADER RECON information

21.4 DB record

The Database (DB) record describes a database. See Figure 86.

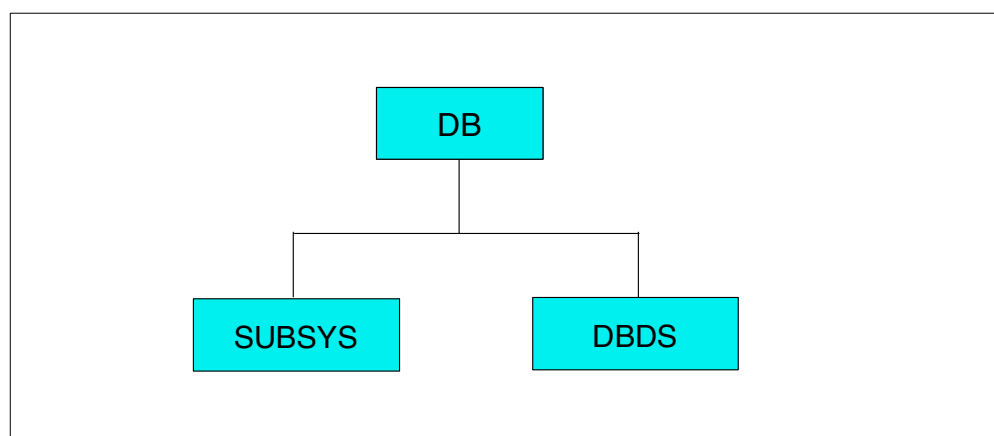


Figure 86. DB record

There is one DB record in the RECON data set for each database that has been registered to DBRC through the use of the INIT.DB command.

A DB record is deleted when the DELETE.DB command is used. After use of DELETE.DB, all DBDS records related to the particular DB record are also deleted.

A DB record includes:

- Name of the DBDS for the database
- Share level specified for the database
- Database status flags

- Current authorization usage

A DB record is symbolically related to:

- The DBDS record for each database data set
- The SUBSYS record for each subsystem currently authorized to use the database.

See Figure 87 below.

```

DB
DB  DBD=BE3PARTS                      DMB#=24      TYPE=IMS
  SHARE LEVEL=1                      GSGNAME=**NULL**  USID=0000000001
  AUTHORIZED USID=0000000000  RECEIVE USID=0000000000  HARD USID=0000000000
  RECEIVE NEEDED USID=0000000000
  FLAGS:                                COUNTERS:
  BACKOUT NEEDED      =OFF              RECOVERY NEEDED COUNT =0
  READ ONLY           =OFF              IMAGE COPY NEEDED COUNT=1
  PROHIBIT AUTHORIZATION=OFF            AUTHORIZED SUBSYSTEMS =0
  RECOVERABLE         =YES              HELD AUTHORIZATION STATE =0
                                           EEQE COUNT           =0
  TRACKING SUSPENDED  =NO              RECEIVE REQUIRED COUNT =0
  OFR REQUIRED         =NO
-

```

Figure 87. DB information

21.5 DBDS record

The Database Data Set (DBDS) record describes a database data set in Figure 88. There is a DBDS record in the RECON data set for each database data set that has been defined to the DBRC using the INIT.DBDS command.

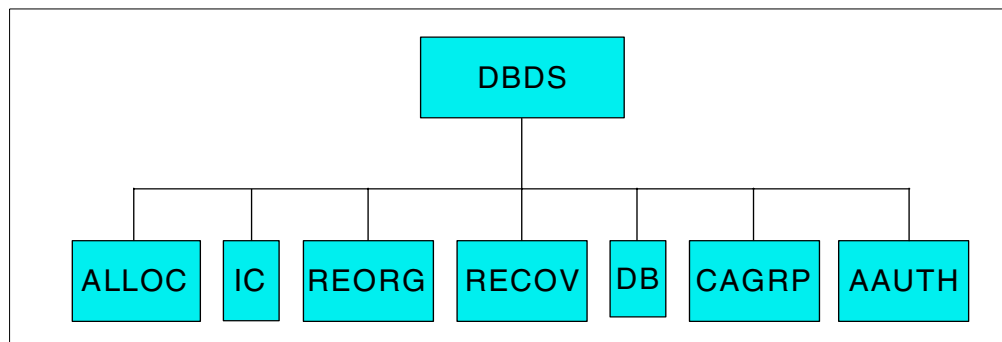


Figure 88. DBDS record

A DBDS record is deleted from RECON with the DELETE.DBDS or the DELETE.DB command.

The DBDS record includes:

- Data set name
- DD name for the data set
- DBD name of the database
- Data set, database organization
- Status flags for the data set
- Information related to image copy or change accumulation

- Name of the JCL member to be used for GENJCL.IC or GENJCL.RECOV.

A DBDS record has the following relationship to other records:

- DB record for the database to which the data set belongs
- CAGRP record for the change accumulation group to which the database data set belongs (when a change accumulation group has been defined)
- ALLOC, IC, REORG, RECOV, AAUTH records.

See Figure 89 below.

DBDS				
DSN=IMS.SJIMSY.BE3PARTS	TYPE=IMS			
DBD=BE3PARTS DDN=BE3PARTS DSID=001 DBORG=HDAM DSORG=VSAM				
CAGRP=**NULL** GENMAX=10 IC AVAIL=0 IC USED=0 DSSN=00000000				
NOREUSE RECOVPD=0				
DEFLTJCL=PARTDFLT ICJCL=ICJCL OICJCL=OICJCL RECOVJCL=PARTRECV				
RECVJCL=ICRCVJCL				
FLAGS:	COUNTERS:			
IC NEEDED =ON				
RECOV NEEDED =OFF				
RECEIVE NEEDED =OFF	EEQE COUNT =0			

Figure 89. DBDS information

21.6 SUBSYS record

The Subsystem (SUBSYS) record informs DBRC that a subsystem is currently active, as shown in Figure 90.

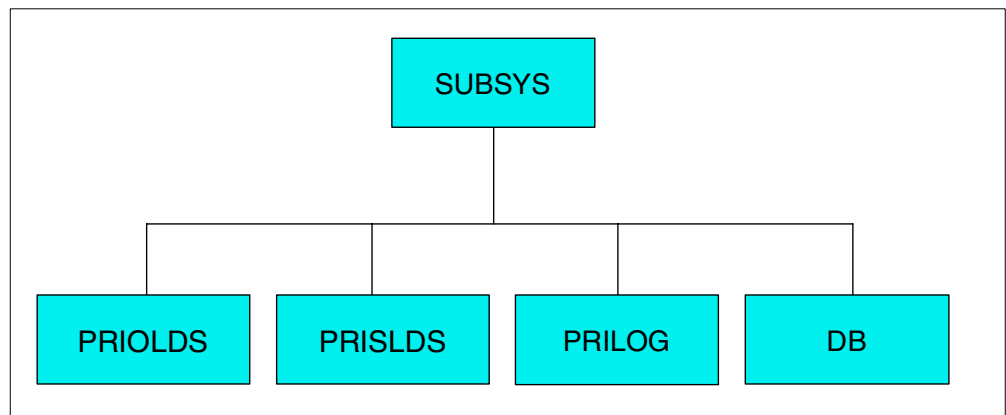


Figure 90. SUBSYS record

A SUBSYS record is created any time a subsystem signs on to DBRCA.

A SUBSYS record is deleted when:

- The subsystem terminates normally
- The subsystem terminates abnormally, but without any database updates
- DBRC is notified of the successful completion of the subsystem recovery process (IMS emergency restart or batch backout).

The SUBSYS record includes:

- ID of the subsystem
- Start time of the log
- Subsystem status flags
- DBDS name for each database currently authorized to the subsystem.

A symbolic relationship exists with the following record types:

- PRILOG record for the log that the subsystem is creating (or PRIOLD or PRISLDS depending on the type of subsystem)
- DB record for each database currently authorized to the subsystem.

See Figure 91.

SSYS				
SSID=IMSY	LOG START=99.207 14:05:48.7			
SSTYPE=ONLINE	ABNORMAL TERM=OFF	RECOVERY STARTED=NO	BACKUP=NO	
TRACKED=NO	TRACKER TERM=OFF	SHARING COVERED DBS=NO		
IRLMID=**NULL**	IRLM STATUS=NORMAL	GSGNAME=**NULL**		
AUTHORIZED DATA BASES/AREAS=4				
VERSION=6.1				
ENCODED				
-DBD-	-AREA-	-LEVEL-	-ACCESS INTENT-	-STATE-
BE3PARTS	**NULL**	1	UPDATE	6
BE3PSID1	**NULL**	1	UPDATE	6
BE3ORDER	**NULL**	1	UPDATE	6
BE3ORDRX	**NULL**	1	UPDATE	6

Figure 91. SUBSYS information

21.7 DBDSGRP record

The Database Data Set Group (DBDSGRP) record describes a DBDS group, as shown in Figure 92.

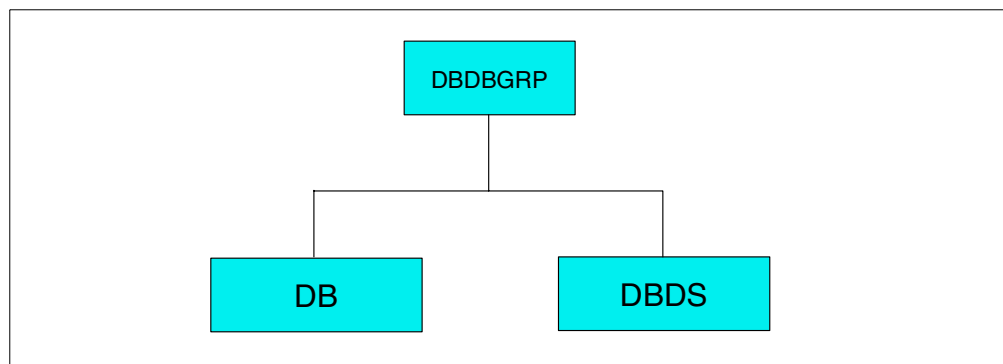


Figure 92. DBDSGRP record

The DBDSGRP is created with the use of the INIT.DBDSGRP command.

It is deleted with the use of the DELETE.DBDSGRP command.

The DBDSGRP record includes:

- DBDSGRP name
- Name of all the DBDSs for the databases defined in the group

- DD name for all the DBDS of the databases defined in the group.

The DBDSGRP record is symbolically related to each:

- DB record defined in the group
- DBDS record defined in the group.

See Figure 93.

DBDSGRP			
GRPNAM=DBGGRP1	#MEMBERS=4	-DBD-	-DDN/AREA-
		BE3PARTS	BE3PARTS
		BE3PSID1	BE3PSID1
		BE3ORDER	BE3ORDER
		BE3ORDRX	BE3ORDRX

Figure 93. DBDSGRP information

21.8 CAGRP record

The Change Accumulation Group (CAGRP) record describes a change accumulation group as shown in Figure 94. Each CAGRP record lists up to 1024 DBDSs whose change records are accumulated together during an execution of the CAGRP utility.

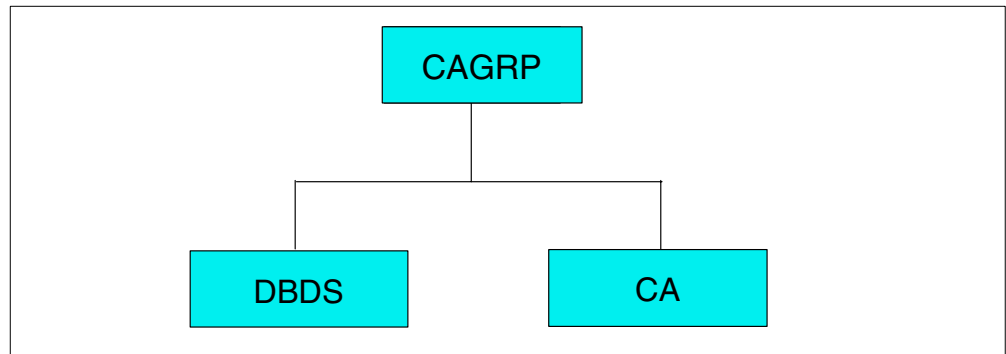


Figure 94. CAGRP record

The CAGRP record is created when the INIT.CAGRP command is used to define the CAGRP to DBRC.

The CAGRP record is deleted when the DELETE.CAGRP command is used. This command also deletes all the CA records related to this particular CAGRP record.

The CAGRP record includes:

- CAGRP name
- DBDS name and the DD name for each DBDS belonging to the CAGRP
- Information related to the CA records
- Skeletal JCL name to be used when the GENJCL.CA command is used

The CAGRP record in Figure 95 below is symbolically related to:

- DBDS record for each member of the CAGRP
- CA records each describing a change accumulation output data set (either used or only pre-allocated).

CAGRP			
GRPNAME=DBGCA	GRPMAX=3	CA AVAIL=0	CA USED=1
NOREUSE	CAJCL=DBGCA	DEFLTJCL=**NULL**	
	#MEMBERS=4	-DBD-	-DDN-
		BE3PARTS	BE3PARTS
		BE3PSID1	BE3PSID1
		BE3ORDER	BE3ORDER
		BE3ORDRX	BE3ORDRX

Figure 95. CAGRP information

21.9 CA record

The Change Accumulation (CA) record in Figure 96, describes a change accumulation data set. There is a CA record for each used change accumulation output data set and one for each change accumulation output data set predefined to DBRC but not yet used.

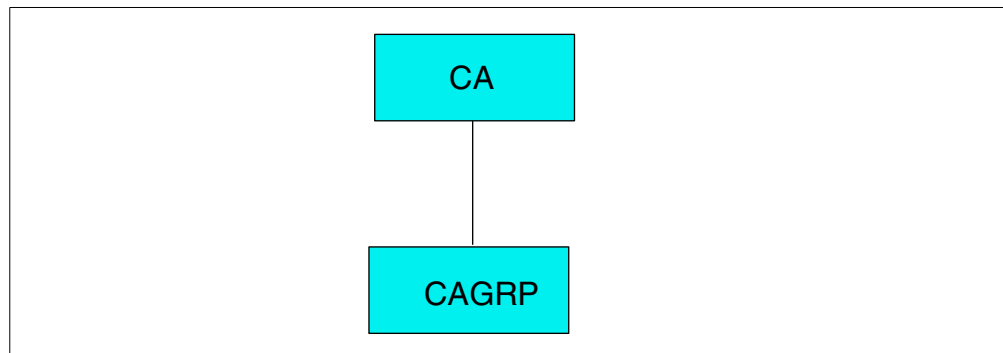


Figure 96. CA record

The INIT.CA command is used to predefine a change accumulation data set to DBRC when the REUSE parameter has been chosen on the INIT.CAGRP command.

The DELETE.CA command (dealing with a single CA record) or the DELETE.CAGRP command (dealing with all the CA records related to a given CAGRP record) can delete the CA records.

The CA record includes:

- Name of the CAGRP to which it belongs
- Data set name of the change accumulation output data set and the volume serial information
- List of purge times for each of the logs input to the change accumulation.

The CA record in Figure 97 below, is symbolically related to the CAGRP record to which it belongs.

CA											
DSN=IMS.SJIMSY.DBG.CALOG.G0001V00						FILE SEQ=1					
CAGRP=DBGCA						UNIT=3400					
STOP		=	99.207	11:22:48.9	VOLS DEF=1		VOLS USED=1				
						VOLSER=STIMS5					
RUN		=	99.207	11:24:17.6							
-DBD-		-DDN-		-PURGETIME-		-CHG-CMP-		-LSN-		-DSSN-	
BE3PARTS		BE3PARTS		99.207	11:11:50.5	YES	YES	000000000000		00000003	
BE3PSID1		BE3PSID1		99.207	11:12:06.6	NO	YES	000000000000		00000000	
BE3ORDER		BE3ORDER		99.207	11:12:26.2	YES	YES	000000000000		00000003	
BE3ORDRX		BE3ORDRX		99.207	11:12:42.9	YES	YES	000000000000		00000003	

Figure 97. CA information

21.10 PRILOG/SECLOG record

The Primary Recovery Log (PRILOG) record or the Secondary Recovery Log (SECLOG) record in Figure 98, describes a log RLDS created by an IMS DC or CICS/OS/VS online system, a batch DL/I job, or the archive utility.

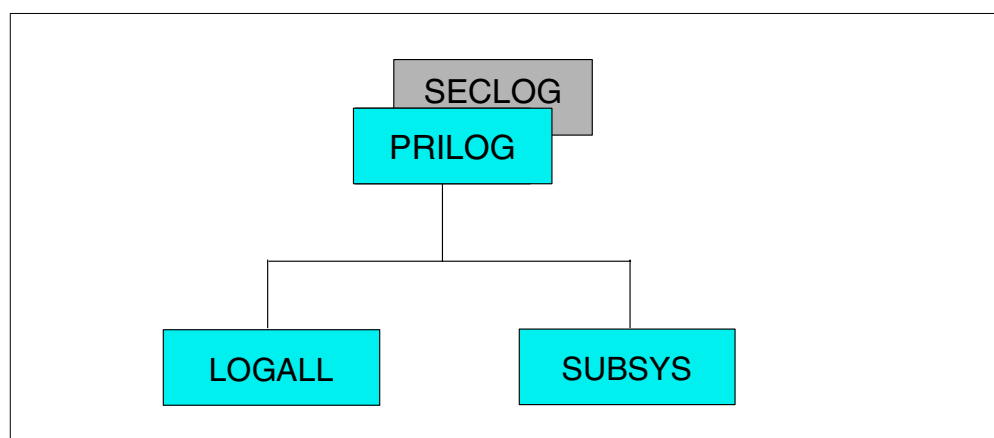


Figure 98. PRILOG/SECLOG record

A PRILOG record is created, together with a LOGALL record, whenever a log is opened. If the subsystem is an IMS batch job and dual log is in use, a SECLOG record is also created.

A PRILOG record is deleted in the following cases:

- The command DELETE.LOG INACTIVE deletes all the log records no longer needed for recovery purposes.
- The command DELETE.LOG TOTIME deletes all the inactive log records older than the specified time.
- The command DELETE.LOG STARTIME deletes a particular log record.

The PRILOG (SECLOG) record in Figure 99 below is symbolically related to:

- The LOGALL record for the same log
- The SUBSYS record for the subsystem creating the log (primary or dual) when the subsystem is active

```

PRILOG
START   = 99.207   11:14:43.9*  STOP    = 99.207   11:19:55.3
SSID=BATCHJ1   #DSN=1           IMS

DSN=IMS.SJIMSY.DBG.B01LOG.G0003V00                                UNIT=3400
START   = 99.207   11:14:43.9   STOP    = 99.207   11:19:55.3
FILE SEQ=0001   #VOLUMES=0001  -VOLSER-  -STOPTIME-
                                TOTIMS1   99.207   11:19:55.3

```

Figure 99. PRILOG information

21.11 PRISLDS/SECSLDS record

The Primary System Log (*PRISLDS*) record or the Secondary System Log (*SECSLDS*) record in Figure 100 describes a system log SLDS created by an *IMS DC online system*.

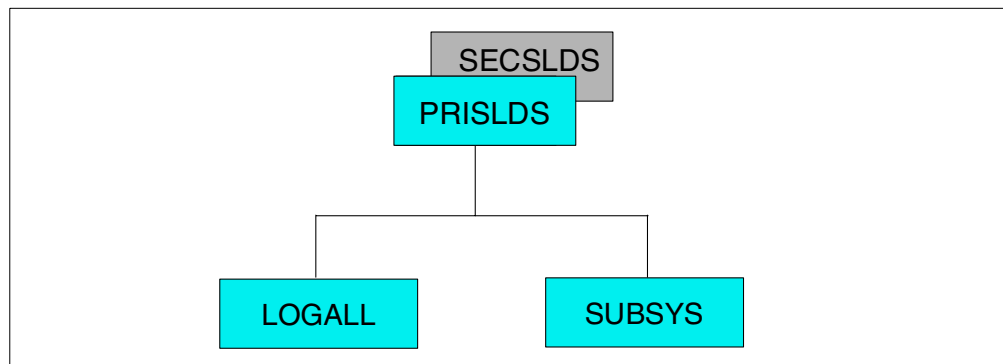


Figure 100. PRISLDS/SECLDS record

PRISLDS record is created, along with a LOGALL record, whenever a system log is opened. A SECSLDS record can be created at archive time.

A PRISLDS record is deleted in the following cases:

- The command DELETE.LOG INACTIVE deletes all the log records no longer needed for recovery purposes.
- The command DELETE.LOG TOTIME deletes all the inactive log records older than the specified time.
- The command DELETE.LOG STARTIME deletes a particular log record.

The PRISLDS (SECSLDS) record in Figure 101 below is symbolically related to:

- The LOGALL record for the same log
- The SUBSYS record for the subsystem creating the log (primary or dual) when the subsystem is active

```
PRISLD
START   = 99.207    12:41:15.3*  STOP    = 99.207    13:03:26.1
SSID=I220          #DSN=1

DSN=IMS.SJMISY.SLDS.G0003V00          UNIT=3400
START   = 99.207    12:41:15.3    STOP    = 99.207    13:03:26.1
FILE SEQ=0001    #VOLUMES=0001 - VOLSER-          -STOPTIME-
                                TOTIMS1    99.207    13:03:26.1
```

Figure 101. PRISLDS information

21.12 PRIOLD/SECOLD record

The Primary OLDS (PRIOLD) record and the Secondary OLDS (SECOLD) record in Figure 102 describe the IMS DC Online Data Sets (OLDS) defined for use. Whenever an OLDS is defined to IMS DC, the PRIOLD record is updated. If IMS dual logging is in use, the SECOLD record is also updated. The PRIOLD (SECOLD) record is deleted by the DELETE.LOG command.

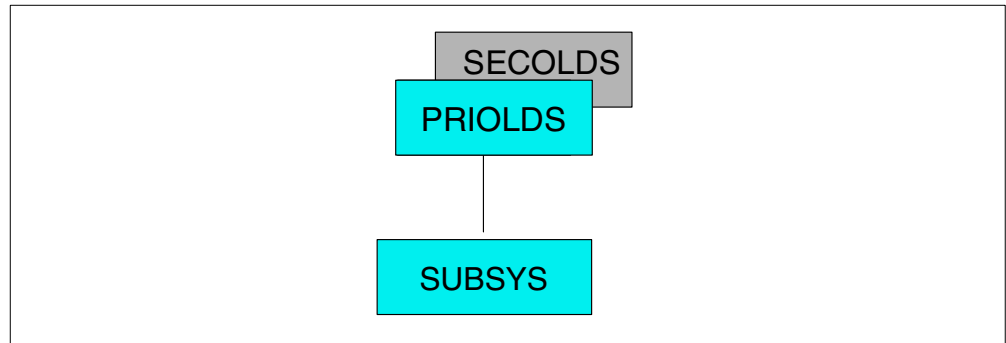


Figure 102. PRIOLDS/SECOLDS record

The PRIOLD (SECOLD) record in Figure 103 is symbolically related to the SUBSYS record for the subsystem using the OLDS (primary or dual).

```

PRIOLD
SSID=I220                # DD ENTRIES=3

DDNAME=DFSOLP02  DSN=STIMS220.OLP02
START   = 88.319   18:46:13.6  STOP    = 88.319   19:01:57.0
STATUS=ARC COMPLT                                FEOV=NO AVAIL
PRILOG TIME=88.319   18:39:58.7      ARCHIVE JOB NAME=ST220ARC

DDNAME=DFSOLP01  DSN=STIMS220.OLP01
START   = 88.321   17:24:41.8  STOP    = 88.321   17:27:33.3
STATUS=ARC COMPLT                                FEOV=NO AVAIL
PRILOG TIME=88.321   17:15:53.9      ARCHIVE JOB NAME=ST220ARC

DDNAME=DFSOLP00  DSN=STIMS220.OLP00
START   = 88.327   12:41:15.3  STOP    = 88.327   13:03:26.1
STATUS=ARC COMPLT                                FEOV=NO AVAIL
PRILOG TIME=88.327   12:41:15.3      ARCHIVE JOB NAME=ST220ARC

SECOLD
SSID=I220                # DD ENTRIES=3
DDNAME=DFSOLS02  DSN=STIMS220.OLS02
START   = 88.319   18:46:13.6  STOP    = 88.319   19:01:57.0
                                           AVAIL

PRILOG TIME=88.319   18:39:58.7
.
DDNAME=DFSOLS01  DSN=STIMS220.OLS01
START   = 88.321   17: 24:41.8    STOP    = 88.321   17:27:33.3
                                           AVAIL

PRILOG TIME=88.321   17:15:53.9

DDNAME=DFSOLS00  DSN=STIMS220.OLS00
START   = 88.327   12:41:15.3  STOP    = 88.327   13:03:26.1
                                           AVAIL

PRILOG TIME=88.327   12:41:15.3

```

Figure 103. PRIOLD/SECOLD information

21.13 LOGALL record

The Log Allocation (LOGALL) record in Figure 104 lists the DBDSs for which database change records have been written to a particular log.

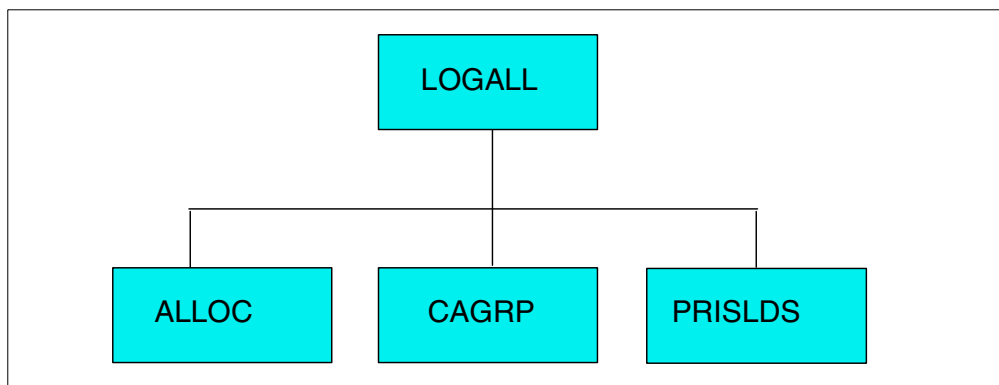


Figure 104. LOGALL record

A LOGALL record is created whenever a PRILOG record is created.

A LOGALL record is deleted from RECON whenever its corresponding PRILOG record is deleted.

The LOGALL record contains a list of the names of DBDSs that have change records on the log. There is a one-to-one correspondence between entries in this list and ALLOC records. Entries are added in this list when ALLOC records are created and deleted when ALLOC records are erased. When there are no more ALLOC records, this list is empty and the log is no longer needed for future recovery.

The LOGALL record in Figure 105 below is symbolically related to:

- The ALLOC record for each of the entry in the LOGALL record
- The PRILOG record for the same recovery log
- The PRISLDS record for the same system log

LOGALL									
START	=	99.207	11:14:43.9*						
DBDS	ALLOC=4			-DBD-	-DDN-	-ALLOC-			
				BE3PARTS	BE3PARTS	1			
				BE3PSID1	BE3PSID1	1			
				BE3ORDER	BE3ORDER	1			
				BE3ORDRX	BE3ORDRX	1			

Figure 105. LOGALL information

21.14 ALLOC record

The Allocation (ALLOC) record in Figure 106 shows that a DBDS has been changed, and that database change records have been written to a particular log.

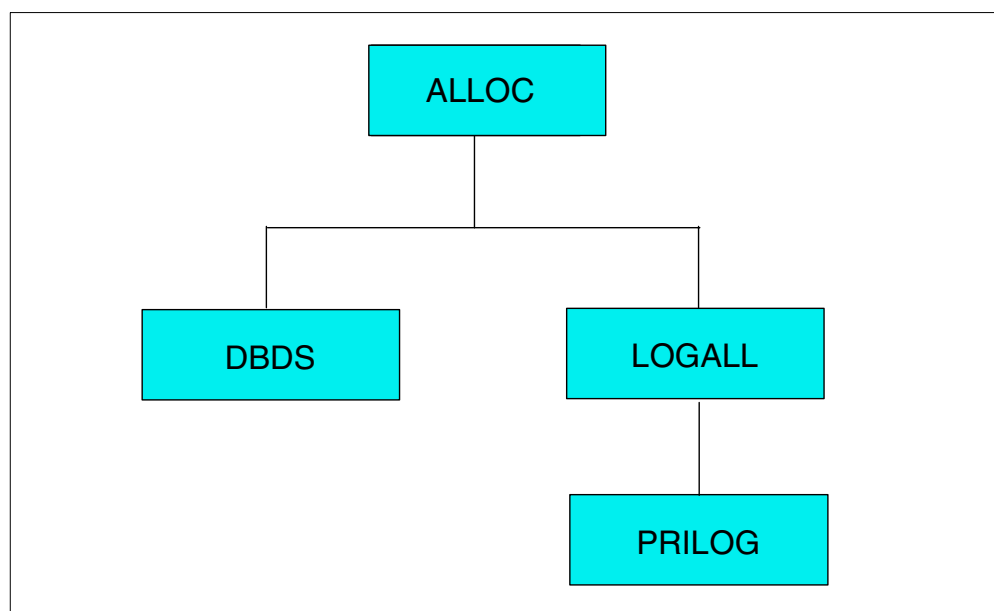


Figure 106. ALLOC record

An ALLOC record is created for a DBDS when a subsystem, signed on to DBRC, updates that DBDS for the first time. The ALLOC record, if still active when the need for recovery arises, shows that the related log must be included in the recovery process.

The ALLOC record is deleted when its DEALLOC time-stamp becomes older than the oldest image copy registered to DBRC for the DBDS.

The ALLOC record in Figure 107 is symbolically related to:

- The DBDS record for the DBDS to which the ALLOC record belongs
- The LOGALL record for the log that the ALLOC record identifies
- The PRILOG record through the LOGALL record

ALLOC				
ALLOC	= 99.207	11:20:48.5	START = 99.207	11:20:40.4
			DSSN=00000003	
ALLOC				
ALLOC	= 99.207	12:56:53.0	START = 99.207	12:41:15.3
DEALLOC	= 99.207	12:59:10.3	DSSN=00000004	

Figure 107. ALLOC information

21.15 IC record

The Image Copy (IC) record in Figure 108 describes an image copy output data set.

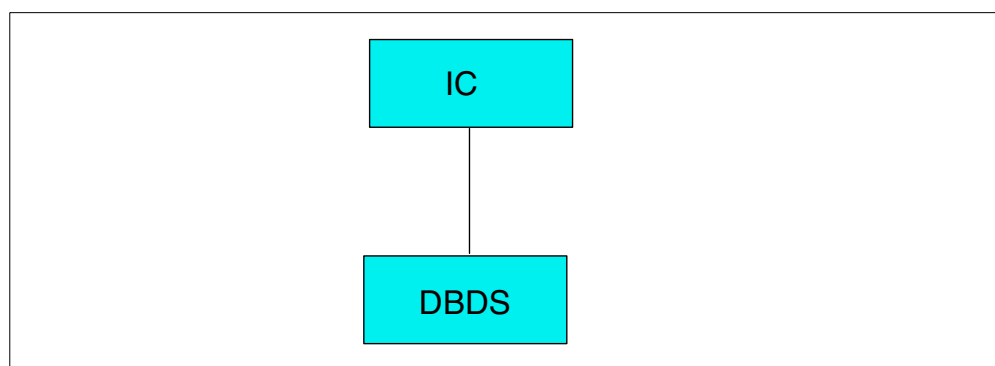


Figure 108. IC record

This record can be created:

- Automatically, when the image copy utility is executed to create a standard image copy
- With the NOTIFY.IC command, when a standard image copy has been created with DBRC = NO
- With the NOTIFY.UIC command, when another nonstandard image copy has been created
- In advance, and reserved for future use with the INIT.IC command, when the related DBDS record has the REUSE option
- By the HISAM reload utility, which creates an IC record pointing to the unload data set if the REUSE option is not being used for the DBDS under reload.

This record is deleted when the maximum image copy generation count is exceeded and its time-stamp is beyond the recovery period.

An option available with the image copy utility allows the user to create two copies of the same IC, referred to as image copy-1 and image copy-2. Both copies are described by the IC record.

The IC record is symbolically related to the DBDS record for the DBDS to which it belongs. See Figure 109 below.

IMAGE					
RUN	=	99.207 13:49:10.5		* RECORD COUNT =81	
STOP	=	00.000 00:00:00.0		BATCH	USID=0000000001
IC1					
DSN=IMS.SJIMSY.BE3PARTS.BKUP.G0001V00				FILE SEQ=0001	
UNIT=3390				VOLS DEF=0001 VOLS USED=0001	
				VOLSER=TSMS12	

Figure 109. IC information

21.16 REORG record

The Reorganization (REORG) record in Figure 110 informs DBRC that a reorganization of a particular DBDS has taken place.

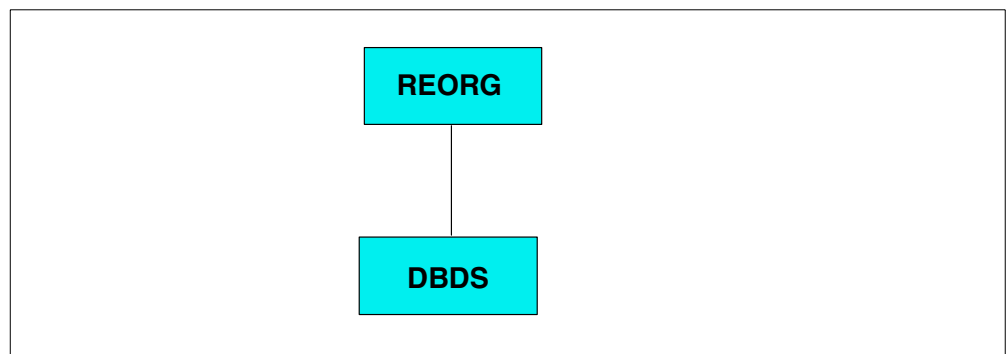


Figure 110. REORG record

With this information, DBRC will not allow recovery operations beyond the time-stamp of this reorganization.

The REORG record is created when:

- A HISAM or HDAM reload utility is successfully executed
- A prefix update utility is executed

The REORG record is deleted when its creation time-stamp is older than the last IC associated with the database data set.

The REORG record in Figure 111 is symbolically related to the DBDS record for the database data set to which it belongs.

REORG	
RUN	= 99.20911:51:35.1*

Figure 111. REORG information

21.17 RECOV record

The Recovery (RECOV) record in Figure 112 informs DBRC that the recovery of a particular DBDS has taken place. With this information, DBRC knows when a time-stamp recovery has been performed.

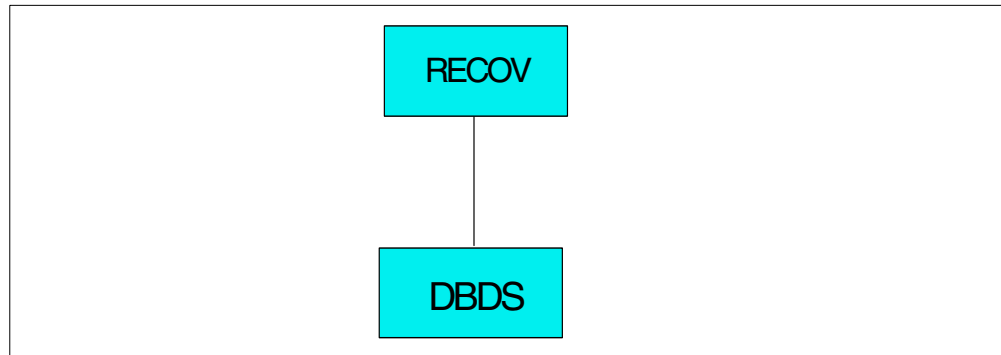


Figure 112. RECOV record

The RECOV record is created when the IMS DB recovery utility is successfully executed.

A RECOV record is erased when its creation time-stamp is found to be older than the oldest IC record associated with the DBDS.

The RECOV record in Figure 113 below is symbolically related to the DBDS record for the database data set to which it belongs.

RECOV			
RUN	= 99.209	12:24:15.3*	

Figure 113. RECOV information

21.18 AAUTH record

The Authorization (AAUTH) record in Figure 114 indicates the sharing status of a Fast Path Database Area.

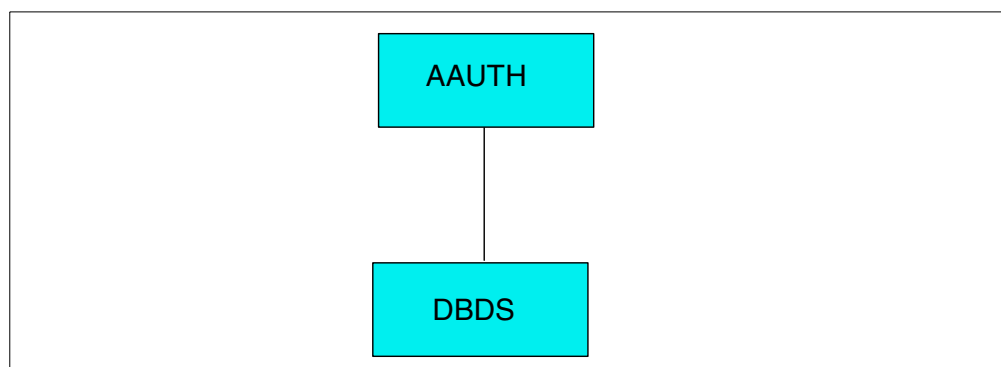


Figure 114. AAUTH record

It is symbolically related to the DBDS record for the DBDS to which it belongs.

21.19 Interim log records

During the DUP phase of the IMS log recovery utility DFSULTRO, interim log records are created for the RECON data set. All these records are temporary records and are deleted when:

- The REP phase of the utility successfully completes
- The DUP phase resolves all errors

These interim log records are as follows:

- IPRILog Interim primary recovery log record
- ISECLOG Interim secondary recovery log record
- IPSLDS Interim primary system log record
- ISSLDS Interim secondary system log record
- IPRIOLD Interim primary OLDS record
- ISECOLD Interim secondary OLDS record

Chapter 22. IMS logging

During IMS execution, all information necessary to restart the system in the event of hardware or software failure is recorded on a system log data set.

The following critical system information is recorded on the logs:

- The receipt of an input message in the input queue
- The start of an MPP/BMP
- The receipt of a message by the MPP for processing
- Before and after images of data base updates by the MPP/BMP
- The insert of a message into the queue by the MPP
- The termination of an MPP/BMP
- The successful receipt of an output message by the terminal

The IMS logs are made up of a number of components, which are described in the following sections:

- Log Buffers, on page 239.
- Online Log Data sets (OLDS), on page 240.
- Write Ahead Data sets (WADS), on page 242.
- System Log Data sets (SLDS), on page 243.
- Recovery Log Data sets (RLDS), on page 244.

22.1 Checkpointing

At regular intervals during IMS execution, checkpoints are written to the log without having to wait to do any physical I/O. A checkpoint is taken after a specified number of log records are written to the log since the previous checkpoint, or after a checkpoint command. Special checkpoint commands are available to stop IMS in an orderly manner.

22.2 IMS log buffers

The log buffers are used for IMS to write any information required to be logged, without having to do any real I/O.

Whenever a log buffer is full, the complete log buffer is scheduled to be written out to the OLDS as a background, asynchronous task. In a busy system, IMS will generally chain these log buffer writes together.

Should any application or system function require a log record to be externalized (that is, IMS believes that for recoverability, this log record must be physically written to DASD before proceeding), then the WADS data set is used. Refer to “Write ahead data sets (WADS)” on page 242.

The OLDS buffers are used in such a manner as to keep available as long as possible the log records that may be needed for dynamic backout. If a needed log record is no longer available in storage, one of the OLDS buffers will be used for reading the appropriate blocks from the OLDS.

The number of log buffers is an IMS start-up parameter, and the maximum is 999. The size of each log buffer is dependent on the actual blocksize of the physical OLDS. The IMS log buffers now reside in extended private storage, however, there is a log buffer prefix that still exists in ECSA.

22.3 Online log data sets (OLDS)

The OLDS are the data sets which contain all the log records required for restart and recovery. These data sets must be pre-allocated (but need not be pre-formatted) on DASD and will hold the log records until they are archived.

The OLDS is written by BSAM. OSAM is used to read the OLDS for dynamic backout.

The OLDS are made up of multiple data sets which are used in a wrap around manner. At least 3 data sets must be allocated for the OLDS to allow IMS to start, while an upper limit of 100 is supported.

Only complete log buffers are written to the OLDS, to enhance performance. Should any incomplete buffers need to be written out, they are written to the WADS. The only exceptions to this are at IMS shutdown, or in degraded logging mode, when the WADS are unavailable, then the WADS writes will be done to the OLDS.

All OLDS should be dynamically allocated, by using the DFSMDA macro, and not hardcoded in the IMS control region JCL.

22.3.1 OLDS dual logging

Dual logging can also be optionally implemented, with a primary and secondary data set for each defined OLDS.

- A primary and secondary data set will be matched and, therefore, the pair should have the same space allocation. Because an OLDS pair will contain the same data, extra space allocated to one will not be used in the other.
- Secondary extent allocation cannot be used
- OLDS can be allocated on different supported DASD
- All OLDS must have the same blocksize, and be a multiple of 2Kb (2048 bytes). the maximum allowable blocksize is 30kb.

22.3.2 Dynamic backout

In addition to the above logging, all previous database record images are written to the OLDS, and can also be used for dynamic back-out processing of a failing MPP/BMP. As soon as the MPP/BMP reaches a synchronization point, the dynamic log information of this program is discarded.

22.3.3 Archiving

The current OLDS (both primary and secondary) is closed and the next OLDS is used whenever one of the following situations occurs:

- OLDS becomes full
- I/O error occurs
- MTO command is entered to force a log switch (such as /SWI OLDS)
- MTO command is issued to close a database (such as /DBR DB) without specifying the NOFEOV parameter.

DBRC is automatically notified that a new OLDS is being used. When this occurs, IMS may automatically submit the archive job.

IMS can define whether the log archive process will occur with every log switch, or every second log switch, and the DBRC skeletal JCL that controls the archiving, can be defined to also create 1 or 2 System Log data sets, and 0, 1 or 2 Recovery Log Data sets. After the last allocated OLDS has been used, the first OLDS will again be used in a wrap around fashion, as long as it has been archived.

The IMS log archive JCL is in DBRC skeletal JCL, and can be tailored to create the required SLDS, and optionally dual SLDS, 1 or 2 RLDS data sets, and any user data sets. Refer to the IMS log archive utility in the IMS Utilities Reference Manual applicable to your IMS release. Figure 115 shows the data sets for the archive utility.

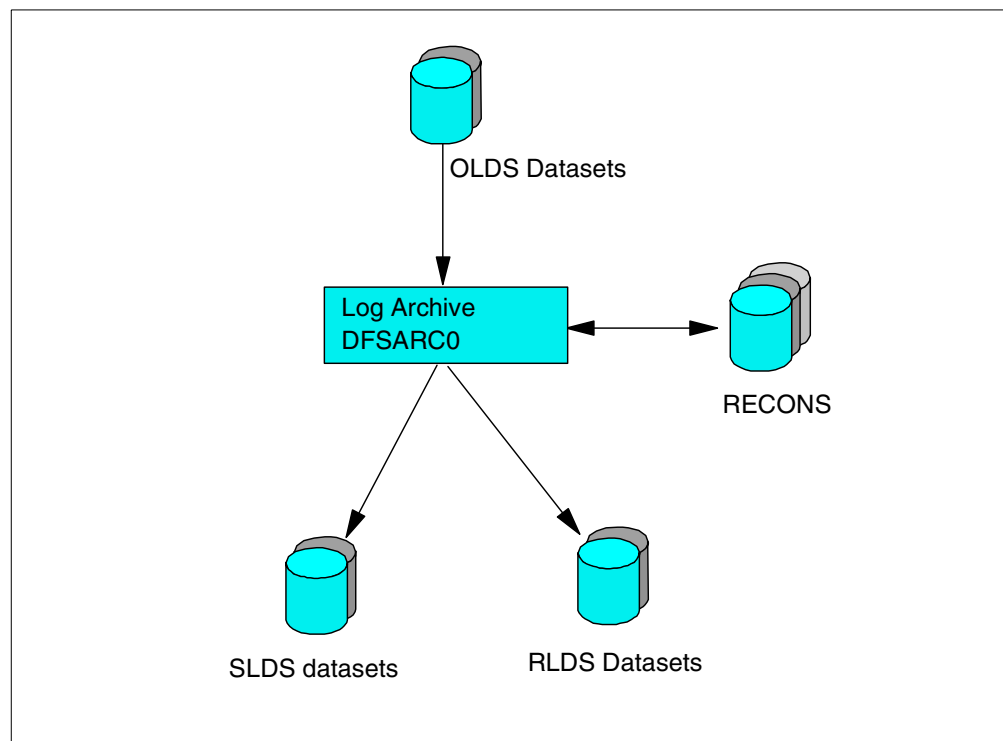


Figure 115. IMS log archive utility

22.3.4 OLDS I/O errors

In the case of a write error, the subject OLDS (or pair of OLDS) will be put into a stopped status and will not be used again. This is equivalent to a user issuing the command /STO OLDS.

If using dual OLDS, then the data set without error will be used for IMS archives.

If data set errors result in only a single OLDS remaining, a /CHE FREEZE command is internally scheduled by IMS. If an error occurs on the very last OLDS, IMS will abend with a U0618.

22.3.5 DBRC

Information is kept in the RECON data set about the OLDS for each IMS system. The data in the RECON indicates whether an OLDS contains active log data which must be archived, or whether it is available for use.

22.3.6 Lack of OLDS

IMS issues messages when it is running out of OLDS.

During the use of the last available OLDS, IMS will indicate that no spare OLDS are available.

When all the OLDS are full, and the archives have not successfully completed, then IMS will stop, and have to wait until at least 1 OLDS has been archived. The only thing IMS will do is repeatedly issue messages to indicate that it is has run out of OLDS, and is waiting.

22.4 Write ahead data sets (WADS)

The WADS is a small direct access data set which contains a copy of committed log records which are in OLDS buffers, but have not yet been written to the OLDS.

When IMS processing requires writing of a partially filled OLDS buffer, a portion of the buffer is written to the WADS. If IMS or the system fails, the log data in the WADS is used to terminate the OLDS, which can be done as part of an Emergency Restart, or as an option on the IMS Log Recovery Utility.

The WADS space is continually reused after the appropriate log data has been written to the OLDS. This data set is required for all IMS systems, and must be pre-allocated and formatted at IMS start-up when first used.

In addition, the WADS provide extremely high performance. This is achieved primarily through the physical design of the WADS. Each WADS track is divided into 2080 byte blocks with a 1 byte key. Each block has the same key (key value = 0). This was done for efficiency on conventional rotational DASD, and is still valid for newer types of DASD.

All WADS should be dynamically allocated by using the DFSMDA macro, and not hardcoded in the control region JCL.

All the WADS must be on the same device type and have the same space allocation.

22.4.1 Dual WADS

Dual WADS is supported to provide backup in the event of a read error while terminating the OLDS from the WADS. The primary and secondary WADS will contain the same data. Single or Dual WADS logging is determined from an IMS start-up parameter.

22.4.2 WADS redundancy

Regardless of whether there are single or dual WADS, there can be up to 10 WADS defined to any IMS. (WADS0, WADS1, ..., WADS9).

WADS0 (and WADS1 if running dual WADS) are active, and the rest remain as spares in case any active WADS has an I/O error. The next spare will then replace the one with the error.

22.5 System log data sets (SLDS)

The SLDS is created by the IMS log archive utility, possibly after every OLDS switch. It is usually placed on TAPE or CARTIDGE, but can reside on DASD. The SLDS can contain the data from one or more OLDS data sets.

The SLDS can also be used as input to all IMS log utilities, and IMS restart.

Information about SLDS is maintained by DBRC in the RECON data set. Calls to DBRC are made by the Archive Utility identifying the OLDS being archived and the SLDS being created. OLDS that have been archived are then available for reuse by IMS.

Dual SLDS

Dual archiving to 2 SLDS data sets (primary and secondary) is supported.

When archiving to TAPE or CARTRIDGE, the user can also force the primary and secondary volumes to contain the same data by specifying the number of log blocks per volume. When this number is reached, a force-end-of-volume (FEOV) will occur on both the primary and secondary SLDS. In this way, both primary and secondary SLDS are identical and interchangeable should a subsequent I/O error occur on one of them.

The user can also specify which records are copied from the OLDS to the SLDS. Generally, the SLDS should contain all the log records from the OLDS, but if the user wants to omit types of log records from the SLDS, these can be specified within the log archive utility.

The SLDS must always contain those log records required for database recovery, batch backout or system recovery.

The blocksize of the SLDS is independent of the OLDS blocksize, and can be specified to maximize space on the SLDS device type.

22.6 Recovery log data sets (RLDS)

When the IMS log archive utility is run, the user can request creation of an output data set that contains all of the log records needed for database recovery. This is the RLDS and is also known to DBRC.

The RLDS is preferred by many installations. All database recoveries and change accumulation jobs will always use the RLDS if one exists, and this can considerably speed up any of these processes because the only contents of these data sets are database recovery log records. All other IMS TM, application scheduling and checkpoint log records are not included on the RLDS's.

The RLDS is optional, and you can also have dual copies of this, in a similar way to the SLDS.

Chapter 23. IMS system generation process

The IMS system generation process is used to build the IMS system, at installation time, as well as at maintenance upgrade, as well as when standard application definition changes are required.

The IMS generation process (usually called “IMS gen”) assumes a working knowledge of SMP/E, required for the installation and is included within parts of the IMS gen process.

23.1 Types of IMS generation

There are 7 different levels of IMS system definitions, as documented in Table 16.

Table 16. Types of IMS system definitions

IMSCTRL option	When Used	Result
ALL	Typical initial generation, usually needed at maintenance time.	Builds most IMS libraries. Includes BATCH and ONLINE options.
ON-LINE	Major update, or initial generation. Often required for maintenance.	Builds most IMS libraries. Includes all but BATCH option.
NUCLEUS	Major maintenance, affecting the contents of the IMS nucleus, or required to build a new nucleus with a new suffix.	Builds IMS nucleus and control blocks. Includes CTLBLKS option.
CTLBLKS	Convenience update. Includes link-edit of existing nucleus with the same suffix, and required for most communications related updates.	Control block generation. Includes MSVERIFY and MODBLKS.
MODBLKS	Convenience update. Used for changes installable using on-line change, such as programs, transactions and database definitions.	Builds control blocks that enable database, program, transaction, and Fast Path routing code changes to be cut over during online operation.
MSVERIFY	Only appropriate for MSC.	Builds control blocks for the MSC Verification utility.
BATCH	Only for the batch environment.	Builds batch environment libraries.

After your initial system definition, usually with an ALL gen, the ON-LINE, CTLBLKS, and NUCLEUS types of generation are used to implement most changes. These generations require a cold start of the IMS online system to take effect.

However, for certain modifications and additions, you can take advantage of the online change method using the MODBLKS generation. The changes are made active during the execution of the online system and do not require a restart operation.

23.2 IMS generation macros

As these change from release to release of IMS, please refer to the *IMS/ESA V6 Installation Volume 2*, (GC26-8737) (or equivalent for your release of IMS) for details of the various macros. They have been summarized here very briefly:

- APPLCTN** The APPLCTN macro allows you to define the program resource requirements for application programs that run under the control of the IMS DB/DC environment, as well as for applications that access databases through DBCTL. An APPLCTN macro combined with one or more TRANSACT macros defines the scheduling and resource requirements for an application program. Using the APPLCTN macro, you only describe programs that operate in message processing regions, Fast Path message-driven program regions, batch message processing regions, or CCTL threads. You do use the APPLCTN macro to describe application programs that operate in batch processing regions. When defining an IMS data communication system, at least one APPLCTN macro is required.
- BUFPOOLS** The BUFPOOLS macro statement is used to specify default storage buffer pool sizes for the DB/DC and DBCTL environments. The sizes specified are used unless otherwise expressly stated for that buffer or pool at control program execution time for an online system.
- COMM** The COMM macro is used to specify general communication requirements that are not associated with any particular terminal type. COMM is always required for terminal types supported by VTAM. It is optional for BTAM, BSAM, GAM, and ARAM terminal types. It can also be required to specify additional system options, such as support for MFS on the master terminal
- CONFIG** The CONFIG macro statement provides the configuration for a switched 3275 terminal. Because the configuration provided by CONFIG is referenced when the named 3275 dials into IMS, differently configured 3275s can use the same communication line. All CONFIG macro statements must be between the LINEGRP macro and the LINE macros. LINE macros can refer to named CONFIG macros defined previously in this line group or in previously defined line groups.
- CTLUNIT** The CTLUNIT macro statement specifies 2848, 2972, and 3271 control unit characteristics. CTLUNIT is valid only for 3270 remote line groups.
- DATABASE** The DATABASE macro statement is used to define the set of physical databases that IMS is to manage. One DATABASE macro instruction must be specified for each HSAM, HISAM, and HDAM database. Two DATABASE macro instructions are required for a HIDAM database: one for the INDEX DBD and one for the HIDAM DBD. One DATABASE macro instruction must be included for each secondary index database that refers to any database defined to the online system. For Fast Path, a DATABASE macro statement must be included for each Main Storage Database (MSDB) and Data Entry Database (DEDB) to be processed.

FPCTRL	The FPCTRL macro statement defines the IMS Fast Path options of the IMS control program, and the DBCTL environment. It is ignored when the IMSCTRL statement specifies that only a BATCH or MSVERIFY system definition is to be performed.
IDLIST	The IDLIST macro statement is used to create a terminal security list for switched 3275s.
IMSCTF	The IMSCTF macro statement defines parameters to IMS, and to the DBCTL environment.
IMSCTRL	The IMSCTRL macro statement describes the basic IMS control program options, the MVS system configuration under which IMS is to execute, and the type of IMS system definition to be performed. <i>The IMSCTRL macro instruction must be the first statement of the system definition control statements.</i>
IMSGEN	IMSGEN specifies the assembler and linkage editor data sets and options, and the system definition output options and features. <i>The IMSGEN must be the last IMS system definition macro, and it must be followed by an assembler END statement.</i>
LINE	The LINE macro statement describes both switched and nonswitched communication lines.
LINEGRP	The LINEGRP macro statement defines the beginning of a set of macro instructions that describe the user's telecommunications system.
MSGQUEUE	You must include the MSGQUEUE macro statement when the type of generation specified in the IMSCTRL macro statement is ALL, ON-LINE, CTLBLKS, or NUCLEUS. This statement defines the characteristics of the three message queue data sets (QBLKS, SHMSG, and LGMSG). The information you specify in this macro is also used in a shared queues environment.
MSLINK	The MSLINK macro statement defines a logical link to another system.
MSNAME	The MSNAME macro statement provides a name for the remote and local system identifications that it represents.
MSPLINK	The MSPLINK macro statement defines a physical MSC link.
NAME	The NAME macro statement defines a logical terminal name (LTERM) associated with a physical terminal. Preparation of the NAME macro can be required for each of the following macros: TERMINAL, SUBPOOL, MSNAME,
POOL	The POOL macro statement describes a pool of logical terminals that are to be associated with a set of switched communication lines.
RTCODE	The RTCODE macro statement is used one or more times with the APPLCTN macro statement that defines an IMS Fast Path application program. It specifies the routing codes that identify the program named in the preceding APPLCTN macro statement. A TRANSACT macro statement that specifies an IMS Fast Path-exclusive transaction generates an internal RTCODE macro statement with a routing code identical to the transaction code.

SECURITY	The SECURITY macro statement lets you specify optional security features to be in effect during IMS execution unless they are overridden during system initialization.
STATION	The STATION macro statement describes the physical and logical characteristics of the System/3 or System/7.
SUBPOOL	The SUBPOOL macro statement, when used: <ul style="list-style-type: none"> • in a VTAM macro set, is a delimiter between groups of NAME macro statements to create LU 6.1 LTERM subpools, or • in a switched communication device macro set, defines a set of logical terminals.
TERMINAL	The TERMINAL macro statement defines physical and logical characteristics of VTAM nodes and non-VTAM communication terminals.
TRANSACT	The TRANSACT macro statement is used one or more times with each APPLCTN macro statement to identify transactions as IMS exclusive, IMS Fast Path potential, or IMS Fast Path exclusive. It specifies the transaction codes that cause the application program named in the preceding APPLCTN macro to be scheduled for execution in an IMS message processing region. It also provides the IMS control program with information that influences the application program scheduling algorithm.
TYPE	The TYPE macro statement defines the beginning of a set of communication terminals and logical terminal description macro statements which include TERMINAL and NAME. The TYPE macro statement begins a description of one set, that contains one or more terminals of the same type. TYPE defines terminals attached to IMS through VTAM. It is equivalent to the LINEGRP/LINE macro set used to define terminals attached to IMS by means other than VTAM.
VTAMPOOL	This macro, required for parallel session support, begins the definition of the LU 6.1 LTERM subpools.

23.3 The IMS generation process

The IMS gen process is made up of any steps, some which only occur depending on the type of IMS gen being undertaken. Some of these steps are:

- Stage1 assembly
- Stage2 assembly, including the optional MFS gen and PROCLIB updates.
- JCLIN
- Re-apply unaccepted maintenance.
- Security Gen

For an overview of the Stage1 and Stage2 components of the gen process, please refer to Figure 116.

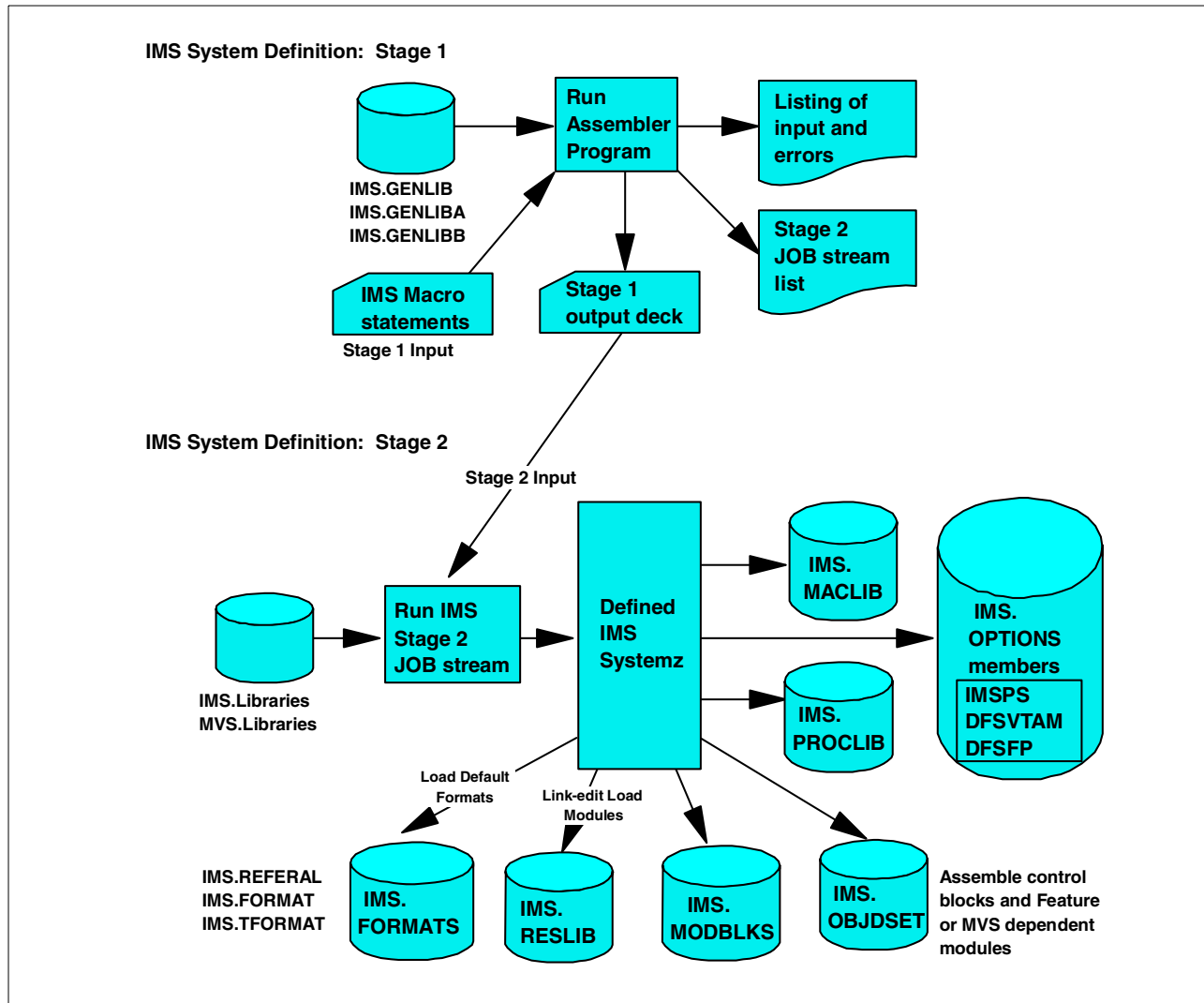


Figure 116. Summary of the two stages of system definition processing

23.3.1 Stage1

The IMS Stage1 job is a simple assembler step, with the SYSIN being the IMS macros, as discussed in 23.2, “IMS generation macros” on page 246. Other references are to the IMS distribution macro libraries (GENLIB, GENLIBA, GENLIBB)

The output of the IMS Stage1 includes:

- Standard assembler listing output with any appropriate error messages.
- IMS Stage2 input JCL, also for use as JCLIN.

23.3.2 Stage2

The output of the Stage1 is then used as the JCL to run the Stage2 gen.

Depending on the Stage1 definitions within the MSGEN macro, the Stage2 can be divided up into a single job with many steps, or many jobs with fewer steps. This is all dependant on how your site prefers to run this process.

The Stage2 will do all the module assembling and linking as required to build all the necessary load modules, depending on what type of gen is being run.

NOTE:

1. In the case of an ALL gen, it is advisable to empty all target libraries first (RESLIB, MACLIB, MODBLKS, MATRIX), to ensure all modules generated are valid, and none are left lying around from previous options no longer in use.
2. An ALL gen will assemble/link almost all modules, whereas a MODBLKS gen will only assemble/link those modules required to define the programs, transactions databases.

As for the Stage1, these steps will all refer to the IMS distribution macro libraries (GENLIB, GENLIBA, GENLIBB) at assembly time, and distribution load library (LOAD) at link time.

The output of the IMS gen includes:

- Executable load modules in datasets RESLIB, MODBLKS.
- IMS Options definitions in dataset OPTIONS.
- Assembled object code for use in later gens in datasets OBJDSET.
- Optionally create the runtime MACLIB dataset. Refer to 23.3.2.1, "MACLIB update" on page 250.
- Optionally create the runtime PROCLIB dataset. Refer to 23.3.2.2, "PROCLIB update" on page 250.
- Optionally create the runtime IMS default MFS screens in datasets FORMAT, TFORMAT, REFERAL. Refer to 23.3.2.3, "MFS gen" on page 251.

23.3.2.1 MACLIB update

A parameter in the IMS Stage1 macro MSGEN (MACLIB=UTILITY/ALL) determines to what level the MACLIB dataset will be populated if the gen type is anything other than CTLBLKS or NUCLEUS. These two options provide:

- UTILITY** Populates MACLIB with only those macros necessary for IMS developers or user generations, such as PSB generation, DBD generation or dynamic allocation generation.
- ALL** Populates MACLIB with all IMS macros, except those necessary for an IMS system generation, and hence, not required by IMS developers or users.

23.3.2.2 PROCLIB update

A parameter in the IMS Stage1 macro MSGEN (PROCLIB=YES/NO) determines whether or not the PROCLIB dataset is to be populated by this gen, or not. The PROCLIB contains all IMS started task and JCL procedures, as well as the IMS PROCLIB members required by IMS and IMS utilities to provide startup options.

23.3.2.3 MFS gen

A parameter in the IMS Stage1 macro MSGEN (MFSDfmt=YES/NO) determines whether or not the default message format screens are built as part of the IMS Stage2. This would then use the REFERAL within the MFS gen, to build the FORMAT dataset.

This option also applies to the test MFS library TFORMAT, if test MFS is required in the system (as specified in the COMM macro).

23.3.3 JCLIN

JCL is an SMP/E process, that tell SMP/E how to assemble and link any module.

As the IMS Stage2 actually assembles and links all the IMS modules based on the definitions for that particular system, and is run outside of SMP/E control, the enter IMS Stage1 output / Stage2 input must be used as input to the JCLIN process, so that SMP/E will know how to manage any maintenance added to the system following this IMS gen.

JCLIN should be run following any IMS gen, to ensure that SMP/E is always kept informed on any parameter changes in the IMS generation.

23.3.4 Re-apply SMP/E maintenance not accepted

All IMS gens are run based on the IMS SMPE distribution libraries.

As a result, any SMP/E maintenance (SYSMODs - PTFs, APARs or USERMODs) that were applied prior to an IMS gen, are likely to be regressed as a result of an IMS gen, depending on the type of IMS gen, and the impact of the SYSMOD.

As a result, it should become routine, that any maintenance that has been APPLIED but not ACCEPTED should be re-APPLIED following an IMS gen, unless further investigations have shown specific cases where this is not necessary.

23.3.5 IMS security maintenance utility generation

For security beyond that provided by default terminal security, you can use the various security options specified with the Security Maintenance utility (SMU). The utility is executed offline after completion of IMS Stage2 processing for system definition. Its output is a set of secured-resource tables placed on the MATRIX dataset. The tables are loaded at system initialization time, and, for certain options, work with exit routines and/or RACF during online execution to provide resource protection.

The IMS Security gen must ALWAYS be run after any MSGEN (full or MODBLK's) as the members in the MATRIX library are generated depending on the position of the resources being protected in the MODBLK's members. If they get out of step, IMS will not start.

For further details, refer to the *IMS/ESA V6 Utilities Ref: System*, SC26-8770, or the redbook *IMS Security Guide*, SG24-5363.

23.4 Automating the IMS system generation process

IMS is shipped to customers with the Installation Verification Procedure (IVP), which will help tailor the initial IMS system, with all provided JCL and sample input, including the IMS system generation jobs. Once these jobs have been generated, the execution of these jobs is manual.

To then tailor the IMS system to suit yourself, the IMS Stage1 macros need to be altered/customized to contain all the necessary definitions required at your site, and the process repeated.

Every time a changed definitions is required, this process will need to be repeated, although the type of IMS gen required may vary from time to time, depending on what has changed, so depending on how often your site requires changes to the IMS Stage1, will determine how often you need to run the IMS gen.

Many customers have put together elaborate or simple means of automating all the steps necessary to run an IMS gen, and maintain their various IMS systems. The more systems maintained within the site, the larger need to have a simple method of maintaining them all.

Although IBM does not make any recommendations on the best method, they could include anything from:

- The various jobs pre-defined in such a way, that each job will automatically submit the next one upon successful completion. This could be done by JCL itself, or the use of a job scheduler.

Notes:

1. Keep in mind that the Stage2 JCL is generated each time from the output of the Stage1, and may vary each time.
2. ISPF driven dialogs to help select the choice of which system to generate, and which type of gen is required, as well as automatically ensuring all required jobs are submitted.

Chapter 24. IMS security overview

This chapter covers some of the issues around IMS security. For further details, please refer to the ITSO redbook *IMS Security Guide*, SG24-5363, as well as the relevant IMS manuals.

24.1 Background to IMS security

When IMS was developed, security products like the Resource Access Control Facility (RACF), had not been developed, or were not in use by most installations. It was common during this period to have each subsystem implement its own security. Therefore, the IMS product offered some basic levels of protection for IMS resources. These internal IMS security facilities (for example, Security Maintenance Utility or SMU) are still available for protecting many IMS resource types and are used by some IMS installations today.

With the development and introduction of security products, like RACF, more and more installations have implemented security for IMS resources using security products. Two advantages of using a security product for securing access to resources are:

- One product may be used to implement the security requirements for multiple subsystems, such as IMS, CICS, and other subsystems.
- All of the security information may be kept and maintained in one place, like the RACF database. One centralized database repository containing all the installations' security specifications eliminated, or significantly minimized, the previous requirements to have:
 - security information distributed among several subsystems, and
 - the security enforcement functions implemented in multiple products

RACF offered a wide range of security choices to the installation. For example, RACF contained new security features, such as user identification (userid) and verification based security which is not available through IMS internally provided SMU security.

IMS provided a new SYSDEF macro that allowed the installation to code all of the security specifications (available during this era) on one macro, the SECURITY macro. This macro was used to specify security options for IMS internally provided SMU security, RACF security, and/or an installation provided security exit routine.

24.2 The security macro

The SECURITY macro was implemented in IMS mainly for the purpose of specifying the installations' security choices to an external security product, like RACF. Again, IMS also provided keywords and parameters on the SECURITY macro that allowed security specifications for SMU provided security as well as for installation-provided security exits. That is, the same security options that could be specified on the COMM and MSGEN macros could now be specified on the SECURITY macro. This maintained the compatibility between security specifications on the SECURITY, COMM, and MSGEN macros.

As in the previous cases, if the SECURITY macro is present in the IMS SYSDEF macros, the specifications and/or defaults of the SECURITY macro will take precedence over security specifications on the COMM and IMSGEN macros. Some of the SECURITY macro keywords and parameters apply to:

- RACF
- IMS SMU
- Installation provided security exit routines
- Combinations of the above (such as SMU and an installation provided security exit routine; or RACF and an installation provided security exit routine)

IMS provides ample flexibility in allowing the installation to secure almost any type of resource. Before deciding which resources to secure, make sure you understand the type of protection that is provided with the specific resource type.

24.3 Protecting IMS terminals

Two type of terminals may be protected in the IMS environment:

1. Physical terminals
2. Logical terminals or LTERMS.

The physical terminals may be static terminals (those which have been defined to IMS using the TYPE and TERMINAL macros); or Extended Terminal Option (ETO) terminals that are dynamically defined to IMS at sign-on.

Terminals can be protected for:

- Signon verification, which will determine whether signon is required
- Resource access security, which will determine whether the terminal has access to issue specific IMS commands, or transactions.

24.4 Protecting IMS commands

IMS commands request the system to perform specific functions, such as displaying information about IMS resources or altering the status of system resources. IMS commands may be entered from several sources:

- A user terminal.
- The IMS master terminal.
- Multiple Console Support/Extended Multiple Console Support (MCS/EMCS) MVS consoles.
- Automated operator programs that issue the DL/I CMD call or the DL/I ICMD call.

The types of protection offered for commands are:

Default terminal security	This type of security is automatically implemented by SMU for a large subset of IMS commands when the SECURITY macro has been coded (or defaulted) to settings that result in an IMS environment where no resource has been protected.
LTERM command security	This is for static terminals. SMU is used to enforce LTERM command security for static terminals. If this type of command protection is used, the installation decides from which LTERMs specific IMS commands may be entered.
Password protected commands	Command password security is implemented using SMU. The statements provided tell IMS the commands that are restricted to use by users that provide the correct password upon entering the command.
Program DL/I CMD call security	Automated operator (AO) programs that issue the DL/I CMD call may be secured by SMU. A parameter on the SECURITY macro (TRANCMD=NO/YES/FORCE) tells IMS whether or not any AO program is permitted to issue IMS commands.
Programs DL/I ICMD call security	AO programs that issue the DL/I ICMD call may also be secured by RACF. The SECURITY macro does not contain a parameter that tells IMS whether or not ICMD calls may be issued from AO application programs. IMS is informed whether or not AO programs may issue ICMD calls based on the specification of the AOIS= parameter in the IMS startup parameters.
Userid based command security	RACF enforces security by checking the CIMS/DIMS RACF classes for command. If a security profile for a command exists in one of the classes (CIMS or DIMS) the command has been secured and protected. RACF checks to see if the userid (or group name) has been authorized to issue the command.
Extended command security	If a greater level of command authorization is required than SMU or RACF can provide (such as verifying authorization to use keywords on specific commands), the Command Authorization Exit (DFSCCMD0) routine may be used to achieve this.

As you may have noted, the security facility used to determine whether or not the command will be processed depends on the origin of the command. For example, the command may have been entered from a static or ETO terminal; or from a program that issued either the DL/I CMD or ICMD call. Furthermore, DFSCCMD0 may also be customized by the installation to provide a more granular level of security. This exit can be used in conjunction with SMU or RACF; or it may be used alone without SMU or RACF to provide command security.

24.5 Protecting IMS transactions

There are six methods that may be used to secure IMS transactions. Five of the security options will be covered here.

Resource access security (LTERM based) — As with IMS LTERM based command security, SMU is used to determine which LTERMs may be used for entering transactions codes. As previously mentioned, password and LTERM based security may be combined. Since SMU is used to provide this type of security the physical terminal (that is associated with the LTERM) must be statically defined to IMS.

Resource access security (password based) — Transaction authorization that is based on securing the transaction with a password is implemented using SMU. Thus the terminals from which the password protected transactions are entered are required to be static terminals.

Extended resource access security — If SMU provided LTERM based and password security is insufficient to meet the installation requirements, the Transaction Authorization Exit (DFSCTRNO) routine may be customized to meet the requirements.

Resource access security (userid based) — RACF enforces transaction authorization security by checking the TIMS/GIMS RACF classes for transaction security profiles. If a security profile for a transaction exists in one of the classes (TIMS or GIMS) the transaction has been secured and protected. RACF checks to see if the userid (or group name) has been authorized to execute the transaction.

Extended resource access security (userid based) — If RACF provided transaction authorization security is insufficient to meet the installation requirements, the DFSCTRNO routine may be customized to meet the requirements. RACF would be called first to determine if the userid/group name is permitted to execute the transaction; and on successful authorization by RACF, DFSCTRNO would be called to perform installation specified transaction authorization security checking.

User customizable (DFSCTRNO) — This exit is enabled within the IMS gen SECURITY macro, and can be tailored to suit any other type of security required. Refer to the appropriate IMS Customization Guide for further details.

24.6 Protecting IMS dependent region(s) and the IMS online system

IMS dependent regions can be protected with Application Group Name (AGN) security. AGN security involved three steps:

1. Deciding which IMS resources will be included in the AGN group and naming the group with the AGN input control card to SMU.
2. Creating a security profile in RACF called AIMS class for the AGN group name and (PERMITting) authorized userids. Or, if RACF is not used to authorized userids, the Resource Access Security Exit routine may be customized by the installation to authorize userids to the AGN group.

3. Determining with IMS dependent region(s) will be allowed to schedule the PSB included in the AGN group. The procedure that is used to start the dependent region contains an **AGN=** parameter in the procedures' JCL. The procedure must specify the correct AGN name in order to schedule the PSB.

24.7 Protecting IMS PSBs and online application programs

PSBs exist for both online application programs and for Batch Message Processing (BMP) programs. PSBs may be protected in several ways.

PSB access security — As you noticed in the AGN example, one method of securing a PSB is by making the PSB part of an AGN group using SMU security facilities. Then RACF or DFSISIS0 can be used to check the user's authorization to use the PSB.

Securing PSB Used by CPI-C Driven Transactions — For Common Programming Interface for Communications (CPIC) driven transactions, security profiles for PSBs may be created in the AIMS class (along with the AGN security profiles). This allows RACF to verify that the CPIC end user's userid is authorized to schedule the PSB. The advantages for the CPIC user are: SMU AGN security is bypassed; and userids (rather than PSB or program name userids) are checked for authorization to schedule the PSB.

24.8 Protecting IMS control program and region application programs

Extended resource protections is used to secure access to the IMS online system. Think of this as securing the IMS control program, the IMS control region, and the VTAM application name for IMS.

The IMS VTAM APPLID may be secured in the RACF APPL class by creating a security profile in the class and authorizing the VTAM APPLID.

Appendix A. Special notices

This publication is intended to help system programmers and application developers who want to gain a general understanding of the IMS product. The information in this publication is not intended as the specification of any programming interfaces that are provided by IMS/ESA. See the PUBLICATIONS section of the IBM Programming Announcement for IMS/ESA for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter 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 the IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Any pointers in this publication to external Web sites are provided for convenience only and do not in any manner serve as an endorsement of these Web sites.

Any performance data contained in this document was determined in a controlled environment, and therefore, the results that may be obtained in other operating environments may vary significantly. Users of this document should verify the applicable data for their specific environment.

Reference to PTF numbers that have not been released through the normal distribution process does not imply general availability. The purpose of including these reference numbers is to alert IBM customers to specific information relative to the implementation of the PTF when it becomes available to each customer according to the normal IBM PTF distribution process.

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

AS/400	AT
C/MVS	CICS
CICS/ESA	Common User Access
CT	CUA
DB2	IBM
IMS	IMS/ESA
MQ	MQSeries
MVS/ESA	Netfinity
OS/390	Parallel Sysplex
RACF	RS/6000
S/370	S/390
SP	System/36
System/360	System/390
VisualAge	VSE/ESA
VTAM	WebExplorer
WebSphere	XT
400	

The following terms are trademarks of other companies:

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

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

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

PC Direct is a trademark of Ziff Communications Company in the United States and/or other countries and is used by IBM Corporation under license.

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

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

SET 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.

Appendix B. Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

B.1 International Technical Support Organization publications

For information on ordering these ITSO publications see “How to get ITSO redbooks” on page 263.

- *IMS/ESA Version 6 Shared Queues*, SG24-5088
- *IMS/ESA Shared Queues: Planning Guide*, SG24-5257
- *IMS e-Business Connect*, SG24-5427
- *IMS Security Guide*, SG24-5363
- *IMS/ESA Data Sharing in a Parallel Sysplex*, SG24-4303
- *IMS/ESA Database Tools Volume I: Database Manager Tools*, SG24-5166
- *IMS/ESA Database Tools Volume II: System Extension Tools*, SG24-5242
- *Connecting IMS to the World Wide Web: A Practical Guide to IMS Connectivity*, SG24-2220
- *IMS/ESA Sysplex Data Sharing: An Implementation Case Study*, SG24-4831
- *IMS Fast Path Solutions Guide*, SG24-4301
- *IMS Version 5 Performance Guide*, SG24-4637

B.2 Redbooks on CD-ROMs

Redbooks are also available on the following CD-ROMs. Click the CD-ROMs button at <http://www.redbooks.ibm.com/> for information about all the CD-ROMs offered, updates and formats.

CD-ROM Title	Collection Kit Number
System/390 Redbooks Collection	SK2T-2177
Networking and Systems Management Redbooks Collection	SK2T-6022
Transaction Processing and Data Management Redbooks Collection	SK2T-8038
Lotus Redbooks Collection	SK2T-8039
Tivoli Redbooks Collection	SK2T-8044
AS/400 Redbooks Collection	SK2T-2849
Netfinity Hardware and Software Redbooks Collection	SK2T-8046
RS/6000 Redbooks Collection (BkMgr Format)	SK2T-8040
RS/6000 Redbooks Collection (PDF Format)	SK2T-8043
Application Development Redbooks Collection	SK2T-8037
IBM Enterprise Storage and Systems Management Solutions	SK3T-3694

B.3 Other publications

These publications are also relevant as further information sources:

- *IMS/ESA V6 Installation Volume 1*, GC26-8736
- *IMS/ESA V6 Installation Volume 2*, GC26-8737
- *IMS/ESA V6 Utilities Ref: System*, SC26-8770
- *IMS/ESA V6 Utilities Ref: Database*, SC26-8034
- *IMS/ESA V6 Utilities Ref: System*, SC26-8770
- *IMS/ESA V6 Customization Guide*, SC26-8020
- *IMS/ESA V6 Administration Guide: System*, GC26-8103
- *IMS/ESA V6 Administration Guide: Database Manager*, GC26-8012
- *IMS/ESA V6 Application Programming: Database Manager*, SC26-8015
- *IMS/ESA V6 Application Programming: Transaction Manager*, SC26-8729
- *DB2 for OS/390 V5 Installation Guide*, GC26-8970

How to get ITSO redbooks

This section explains how both customers and IBM employees can find out about ITSO redbooks, redpieces, and CD-ROMs. A form for ordering books and CD-ROMs by fax or e-mail is also provided.

- **Redbooks Web Site** <http://www.redbooks.ibm.com/>

Search for, view, download, or order hardcopy/CD-ROM redbooks from the redbooks Web site. Also read redpieces and download additional materials (code samples or diskette/CD-ROM images) from this redbooks site.

Redpieces are redbooks in progress; not all redbooks become redpieces and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

- **E-mail Orders**

Send orders by e-mail including information from the redbooks fax order form to:

	e-mail address
In United States	usib6fpl@ibmmail.com
Outside North America	Contact information is in the "How to Order" section at this site: http://www.elink.ibm.link.ibm.com/pbl/pbl/

- **Telephone Orders**

United States (toll free)	1-800-879-2755
Canada (toll free)	1-800-IBM-4YOU
Outside North America	Country coordinator phone number is in the "How to Order" section at this site: http://www.elink.ibm.link.ibm.com/pbl/pbl/

- **Fax Orders**

United States (toll free)	1-800-445-9269
Canada	1-403-267-4455
Outside North America	Fax phone number is in the "How to Order" section at this site: http://www.elink.ibm.link.ibm.com/pbl/pbl/

This information was current at the time of publication, but is continually subject to change. The latest information may be found at the redbooks Web site.

IBM Intranet for Employees

IBM employees may register for information on workshops, residencies, and redbooks by accessing the IBM Intranet Web site at <http://w3.itso.ibm.com/> and clicking the ITSO Mailing List button. Look in the Materials repository for workshops, presentations, papers, and Web pages developed and written by the ITSO technical professionals; click the Additional Materials button. Employees may access MyNews at <http://w3.ibm.com/> for redbook, residency, and workshop announcements.

IBM Redbook Fax Order Form

Please send me the following:

Title	Order Number	Quantity
-------	--------------	----------

[illegible]

First name	Last name
------------	-----------

Company	Revenue	Profit	Assets	Liabilities	Equity
Company A	100	20	50	30	20
Company B	150	30	75	45	30
Company C	200	40	100	60	40
Company D	250	50	125	75	50
Company E	300	60	150	90	60

Address _____

City	Postal code	Country
------	-------------	---------

Telephone number	Telefax number	VAT number
------------------	----------------	------------

☐ Invoice to customer number☐ Credit card number

Credit card expiration date	Card issued to	Signature
-----------------------------	----------------	-----------

We accept American Express, Diners, Eurocard, Master Card, and Visa. Payment by credit card not available in all countries. Signature mandatory for credit card payment.

Glossary

A

advanced program-to-program communication (APPC). (1) IBM's architected solution for program-to-program communication, distributed transaction processing, and remote database access. A transaction program (TP) using the APPC API can communicate with other TPs on systems that support APPC. (2) An implementation of the Systems Network Architecture (SNA) logical unit (LU) 6.2 protocol that enables interconnected systems to communicate and share the processing of programs.

API. See application program interface.

applet. An applet is a piece of Java bytecode that is executed on the workstation, under control of the Web browser's Java Virtual Machine. The applet is downloaded when the browser accesses a page containing an <APPLET> tag.

application. (1) The use to which an information processing system is put; for example, a payroll application or an order-entry application. (2) A collection of defined and extended classes that provides a reusable piece of functionality. An application contains and organizes functionally related classes. It also can contain subapplications and specify prerequisites.

application program interface (API). An architected functional interface supplied by an operating system or other software system. The interface enables an application program written in a high-level language to use specific data or functions of the underlying system.

ASCII. (American Standard Code for Information Interchange), this is the world-wide standard for the code numbers used by computers to represent all the upper and lower-case Latin letters, numbers, punctuation, etc. There are 128 standard ASCII codes each of which can be represented by a 7-digit binary number, 0000000 through 1111111.

authority. The right to do something on the system or to use an object, such as a file or document, in the system.

authorization list. A list that gives a group of users one or more types of access to objects (such as files or programs) or data in the objects (such as records in a file). It consists of a list of two or more user IDs and their authorities for system resources.

B

bandwidth. How much stuff you can send through a connection, usually measured in bits per second. A full page of English text is about 16,000 bits.

Base Primitive Environment (BPE). A system service component that underlies the HWS address space.

baud. In common usage the baud rate of a modem is how many bits it can send or receive per second. Technically, baud is the number of times per second that the carrier signal shifts value; for example a 1200 bit/second modem actually runs at 300 baud, but it moves 4 bits per baud. See also bit, modem.

bit. (binary digit) A single digit number in base 2, in other words, either a 1 or a zero. The smallest unit of computerized data. *Bandwidth* is usually measured in bits per second. See also bandwidth, bps, byte, kilobyte, megabyte.

Bps. (bits per second) A measurement of how fast data is moved from one place to another. ie. A 28.8 modem can move 28,800 bits per second. See also bandwidth, bit.

browser. Software that enables users to browse through the cyberspace of the World Wide Web. See also Client, URL, WWW.

byte. A set of bits that represent a single character. Usually there are 8 bits in a byte, sometimes more, depending on how the measurement is being made.

C

CGI Link. A stand-alone executable program that receives incoming CGI requests and routes them to the VisualAge application. CGI Link runs on the HTTP server, which does not have to be the same as the machine running the VisualAge application.

CGI query. A special kind of HTTP request from a client browser requesting that a server-based program be run&period. A CGI query specifies the name of the program to run, along with any input parameters. See also *Common Gateway Interface*.

client. A software program that is used to contact and obtain data from a server software program on another computer, often across a great distance. Each client program is designed to work with one or more specific kinds of server programs, and each server requires a specific kind of client. A Web browser is a specific kind of client. See also browser, server.

client/server. The model of interaction in distributed data processing in which a program at one location sends a request to a program at another location and awaits a response. The requesting program is called a client, and the answering program is called a server.

Common Gateway Interface. A standard protocol through which a Web server can execute programs running on the server machine. CGI programs are executed in response to requests from Web client browsers.

Common User Access (CUA). An IBM architecture for designing graphical user interfaces that uses a set of standard components and terminology.

configuration. A description of a group of components that identifies, for each component, the component edition or version that is part of the group.

D

database manager. other word for a database management system.

datastore. An IMS TM system that provides transaction and database processing.

domain name. The unique name that identifies an Internet site. Domain names always have two or more parts, separated by dots. The part on the left is the most specific, and the part on the right is the most general. A given machine may have more than one domain name but a given domain name points to only one machine. Usually, all of the machines on a given network will have the same thing as the right-hand portion of their domain names, for example, gateway.mynetwork.com.br, mail.mynetwork.com.br, www.mynetwork.com.br, and so on. It is also possible for a domain name to exist but not be connected to an actual machine. This is often done so that a group or business can have an Internet e-mail address without having to establish a real Internet site. In these cases, some real Internet machine must handle the mail on behalf of the listed domain name. See also IP Number.

dynamic link library (DLL). A file containing data and code objects that can be used by programs or applications during loading or at run time but are not part of the program's executable (.EXE) file.

E

e-mail. (Electronic mail) Messages transmitted over the Internet from user to user. E-mail can contain text, but also can carry with it files of any type as attachments.

F

feature. A major component of a software product that can be ordered separately.

field. A group of related bytes (such as name or amount) that are treated as a unit in a record.

firewall. A combination of hardware and software that protects a local area network (LAN) from Internet hackers. It separates the network into two or more parts and restricts outsiders to the area outside the firewall. Private or sensitive information is kept inside the firewall.

first-in first-out (FIFO). A queuing technique in which the next request to be processed from a queue is the request of the highest priority that has been on the queue for the longest time.

form. An HTML element that can include entry fields, push buttons, and other user-interface controls through which users can enter information&period. Sometimes called a *fill-in form*.

FTP. (File Transfer Protocol) The basic Internet function that enables files to be transferred between computers. You can use it to download files from a remote, host computer, as well as to upload files from your computer to a remote, host computer. (See Anonymous FTP).

G

gateway. A host computer that connects networks that communicate in different languages. For example, a gateway connects a company's LAN to the Internet.

GIF. (Graphics Interchange Format) A graphics file format that is commonly used on the Internet to provide graphics images in Web pages.

Gopher. Gopher is a facility that helps you find resources on the Internet. Gopher presents you simple based menus. Each menu item represents either another Gopher menu, or can take you directly to facilities or services such as viewing or down-loading files, or starting a Telnet session. Because Gopher menus could point to other Gopher servers, you can search for resources across the whole Internet system.

graphical user interface (GUI). A type of interface that enables users to communicate with a program by manipulating graphical elements rather than by entering commands. Typically, a graphical user interface includes a combination of graphics, pointing devices, menu bars, overlapping windows, and icons.

H

host. (1) A computer that "hosts" outside computer users by providing files, services or sharing its resources. (2) Any computer on a network that is a repository for services available to other computers on the network. It is quite common to have one host machine provide several services, such as *WWW* and *USENET*. See also Node, Network.

Host Web Service (HWS). An other short name for ITOC. It is also the prefix of the module and messages. This short name involves that only web clients can use it, but it's not exact: any TCP/IP client can connect to IMS thru HWS.

HTML (hypertext markup language). The basic language that is used to build hypertext documents on the World Wide Web. It is used in basic, plain ASCII-text documents, but when those documents are interpreted (called rendering) by a Web browser such as Netscape, the document can display formatted text, color, a variety of fonts, graphic images, special effects, hypertext jumps to other Internet locations and information forms.

HTTP (hypertext transfer protocol). The protocol for moving hypertext files across the Internet. Requires a HTTP client program on one end, and an HTTP server program on the other end. HTTP is the most important protocol used in the World Wide Web (*WWW*). See also Client, Server, *WWW*.

HTTP request. A transaction initiated by a Web browser and adhering to HTTP. The server usually responds with HTML data, but can send other kinds of objects as well.

hypertext. Text in a document that contains a hidden link to other text. You can click a mouse on a hypertext word and it will take you to the text designated in the link. Hypertext is used in Windows help programs and CD encyclopedias to jump to related references elsewhere within the same document. The wonderful thing about hypertext, however, is its ability to link - using HTTP over the Web - to any Web document in the world, yet still require only a single mouse click to jump clear around the world.

I

icon. A small pictorial representation of an object.

index. A set of pointers that are logically arranged by the values of a key. Indexes provide quick access and can enforce uniqueness on the rows in a table.

Internet. The vast collection of interconnected networks that all use the TCP/IP protocols and that evolved from the ARPANET of the late 1960's and early 1970's.

intranet. A private *network* inside a company or organization that uses the same kinds of software that you would find on the public *Internet*, but that is only for internal use. As the Internet has become more popular, many of the tools used on the Internet are being used in private networks, for example, many companies have Web servers that are available only to employees.

IP. (Internet Protocol) The rules that provide basic Internet functions. See TCP/IP.

IP Number. An Internet address that is a unique number consisting of four parts separated by dots, sometimes called a *dotted quad*. (For example: 198.204.112.1). Every Internet computer has an IP number and most computers also have one or more domain names that are plain language substitutes for the dotted quad.

ISDN (Integrated Services Digital Network). A set of communications standards that enable a single phone line or optical cable to carry voice, digital network services and video. ISDN is intended to eventually replace our standard telephone system.

ITOC (IMS TCP/IP OTMA Connection). The IBM provided software that acts as an OTMA bridge between IMS/OTMA and TCP/IP. Its main use is to connect IMS with the internet.

J

Java. Java is a programming language invented by Sun Microsystems that is specifically designed for writing programs that can be safely downloaded to

your computer through the Internet and immediately run without fear of viruses or other harm to your computer or files. Using small Java programs (called *applets*, Web pages can include functions such as animations, calculators, and other fancy tricks. We can expect to see a huge variety of features added to the Web using Java, since you can write a Java program to do almost anything a regular computer program can do, and then include that Java program in a Web page.

Java Beans. Java Beans is a set of APIs that make it easy to create Java applications from reusable components. It is a platform-neutral, component-based software architecture for the Java Platform and is device and operating system independent.

Java Bytecode. The solution that the Java system adopts to solve the binary distribution problem is a "binary code format" that's independent of hardware architectures, operating system interfaces, and window systems. The format of this system-independent binary code is architecture neutral.

Java Classes. A class is a software construct that defines the data (state) and methods (behavior) of the specific concrete objects that are subsequently constructed from that class. In Java terminology, a class is built out of members, which are either fields or methods.

Java Packages. Java packages are collections of classes and interfaces that are related to each other in some useful way. Such classes need to be able to access each other's instance variables and methods directly.

JavaScript. JavaScript is an easy-to-use object scripting language designed for creating live online applications that link together objects and resources on both clients and servers.

JITOC. The IMS TCP/IP OTMA Connection Connector for Java is a set of Java beans which provide a way to create Java applications that can access IMS transactions. The ITOC Connector for Java provides a Common Connector Framework-compliant Java interface to ITOC.

JPEG. (Joint Photographic Experts Group) The name of the committee that designed the photographic image-compression standard. JPEG is optimized for compressing full-color or gray-scale photographic-type, digital images. It doesn't work well on drawn images such as line drawings, and it does not handle black-and-white images or video images.

K

kbps. (kilobits per second) A speed rating for computer modems that measures (in units of 1024 bits) the maximum number of bits the device can transfer in one second under ideal conditions.

kilobyte. A thousand bytes. Actually, usually 1024 bytes. See also byte, bit.

L

LAN. Local area network. A computer network located on a user's establishment within a limited geographical area. A LAN typically consists of one or more server machines providing services to a number of client workstations. See also Ethernet.

listserv. An Internet application that automatically serves mailing lists by sending electronic newsletters to a stored database of Internet user addresses. Users can handle their own subscribe/unsubscribe actions without requiring anyone at the server location to personally handle the transaction.

Login. The account name used to gain access to a computer system. Not kept secret (unlike password).

M

Mail. The Internet provides electronic mail using the Simple Mail Transfer Protocol (SMTP) and Post Office Protocol (POP). Most electronic mail services provide a gateway to Internet mail so if you have access to Internet mail you E-Mail access to millions of people. A new standard called Multipurpose Internet Mail Extension (MIME) allows you now to send mail that includes binary and multimedia objects.

megabyte. A million bytes. A thousand kilobytes. See also byte, bit, kilobyte.

MIME. (Multipurpose Internet Mail Extensions) A set of Internet functions that extend normal e-mail capabilities and enable nontext computer files to be attached to e-mail. Nontext files include graphics, spreadsheets, formatted word-processor documents, sound files, and so on. Files sent by MIME arrive at their destination as exact copies of the original so that you can send fully formatted word processing files, spreadsheets, graphics images and software applications to other users via simple e-mail. Besides email software, the MIME standard is also universally used by Web servers to identify the files they are sending to Web clients, in this way new file formats can be accommodated simply by updating the browsers' list of pairs of MIME types and appropriate software for handling each type. See also browser, client, server.

N

News. Internet News (also called Usenet) is a discussion or conferencing facility. Thousands of different news groups cover almost any subject you can imagine.

notebook. A view that resembles a bound notebook, containing pages separated into sections by tabbed divider pages. A user can turn the pages of a notebook or select the tabs to move from one section to another.

O

object-oriented programming. A programming methodology built around objects and based on sending messages back and forth between those objects. The basic concepts of object-oriented programming are encapsulation, inheritance, and polymorphism.

Open Transaction Manager Access (OTMA). A transaction-based connectionless client/server protocol using XCF as communication vehicle.

P

parameter. A data element included as part of a message to provide information that the object might need. In Smalltalk, generally referred to as an argument.

password. A code used to gain access to a locked system. Good passwords contain letters and nonletters and are not simple combinations.

PATH_INFO. A CGI variable, usually transmitted to the CGI program in the form of an environment variable. The PATH_INFO variable contains all path information from the URL following the name of the CGI executable. For a Web Connection application, this information is the same as the VisualAge part name.

Port. (1) A place where information goes into or out of a computer, or both. For example, the serial port on a personal computer is where a modem would be connected. (2) On the Internet port often refers to a number that is part of a URL, appearing after a colon (:) right after the domain name. Every *service* on an Internet server listens on a particular port number on that server. Most services have standard port numbers; Web servers normally listen on port 80. Services can also listen on nonstandard ports, in which case the port number must be specified in a URL when accessing the server. (3) Refers to translating a piece of software to bring it from one type of computer system to another. See also domain name, server, URL. (4) In the case of ITOC, HWS address space represents several port numbers; each port will provide access to one of a number of sockets associated with the IMS Transaction Manager systems HWS is connected to.

POST. One of the methods used in HTTP requests. A POST request is used to send data to an HTTP server. See also GET.

protocol. (1) The set of all messages to which an object will respond. (2) Specification of the structure and meaning (the semantics) of messages that are exchanged between a client and a server. (3) Computer rules that provide uniform specifications so that computer hardware and operating systems can communicate. It's similar to the way that mail, in countries around the world, is addressed in the same basic format so that postal workers know where to find

the recipient's address, the sender's return address and the postage stamp. Regardless of the underlying language, the basic protocols remain the same.

proxy. An application gateway from one network to another for a specific network application like Telnet or FTP, for example, a firewall's proxy Telnet server performs authentication of the user and then lets the traffic flow through the proxy as if it were not there. Function is performed in the firewall and not in the client workstation, causing more load in the firewall. Compare with socks.

R

receiver. The object that receives a message. Contrast with sender.

record. A group of related data, fields, or words, treated as a unit, such as name, address, and telephone number.

repository. (1) An organized, shared body of information that can support business and data-processing activities.

reset button. A type of push button that can appear on a form. A reset button restores all input fields to their default states.

return value. An object or data type that a receiver object passes to a sender object in response to a message.

router. A network device that enables the network to reroute messages it receives that are intended for other networks. The network with the router receives the message and sends it on its way exactly as received. In normal operations, they do not store any of the messages that they pass through.

S

script. A series of commands that define the sequence in which they will have to be processed.

sender. An object that sends a message to another object. On the level of code implementation, the sender is considered to be the sending method within the class or instance that issues the message. Contrast with receiver.

server. (1) A computer that provides services to multiple users or workstations in a network; for example, a file server, print server, or mail server. (2) An object that performs one or more tasks on behalf of a client. The server can be a computer (a file server), a specific process on a server, or a distributed object. A single server machine could have several different server software packages running on it, thus providing many different servers to clients on the network. See also client, network.

service. A specific behavior that an object is responsible for exhibiting.

servlet. A servlet is a piece of Java code that runs inside a Java-enabled Webserver, such as the Lotus Domino Go Webserver Release 4.6.1 or IBM HTTP Server 1.3.3 with IBM WebSphere Application Server V2.0, and extends the functions of the server. The server hands requests to the servlet, which replies to them. Servlets are a good substitute for CGI programs because they are faster and more easily manageable.

session. A series of commands that come from the same client and belong to the same logical sequence and period. A session is identified by a unique session key, which is generated by VisualAge. A session begins when a client initially connects (without a session key) and ends when a specified timeout period has elapsed since the last connection.

socket. An end-point to which clients can connect. This address is unique on the entire network. The connection between two sockets provides a full duplex communication path between the two end processes.

socks. Software to intercept and redirect all TCP/IP requests at the firewall. It handles data to and from applications such as Telnet, FTP, Mosaic, and Gopher. Provides users in a secured network access to resources outside the network by directing data through the firewall. Firewall users must use client programs specifically designed to work with the sock server.

structured query language (SQL). A language used to access relational databases.

Systems Network Architecture (SNA). The description of the logical structure, formats, protocols, and operational sequences for transmitting information units through, and controlling the configuration and operation of, networks.

T

TCP/IP. (Transmission Control Protocol/Internet Protocol) The basic programming foundation that carries computer messages around the globe via the Internet. The suite of protocols that defines the Internet. Originally designed for the UNIX operating system, TCP/IP software is now available for every major kind of computer operating system. To be truly on the Internet, your computer must have TCP/IP software.

Telnet. An Internet protocol that lets you connect your PC as a remote workstation to a host computer anywhere in the world and to use that computer as if you were logged on locally. You often have the ability to use all of the software and capability on the host computer, even if it's a huge mainframe.

U

uniform resource locator (URL). A standard identifier for a resource on the World Wide Web, used by a Web browser to initiate a connection. The URL includes the communications protocol to use, the name of the

server, and path information identifying the object to be retrieved on the server. A URL looks like :
http://www.ibm.com, or *telnet://well.sf.ca.us.br*, or *news:new.newusers.questions.br*

user profile. A file that contains the user's password, the list of special authorities assigned to a user, and the objects the user owns. It is used by the system to verify the user's authorization to read or use objects, such as files or devices, or to run the jobs on the system. Each user profile must have a unique name.

V

variable. A storage place within an object for a data element. The data element is an object, such as a number or date, stored as an attribute of the containing object.

W

WAN. (Wide Area Network). Any internet or network that covers an area larger than a single building or campus. See also Internet, LAN, network.

Web Browser. As many other Internet facilities, the Web uses a client-server processing model. The Web browser is the client component. Examples of Web browsers include Mosaic, Netscape and the IBM WebExplorer. The Web browser is responsible for formatting and displaying information, interacting with the user and invoking external viewers for data types that it doesn't support directly.

Web Server. Web servers are responsible for servicing requests for information from Web browsers. The information can be a file retrieved from the servers local disk or generated by a program called by the server to perform a specific application function.

window. A rectangular area of the screen with visible boundaries in which information is displayed. Windows can overlap on the screen, giving the appearance of one window being on top of another.

World Wide Web. (WWW) (W3) (the Web) An Internet client-server distributed information and retrieval system based upon HTTP that transfers hypertext documents across a varied array of computer systems. The Web was created by the CERN High-Energy Physics Laboratories in Geneva, Switzerland in 1991. CERN boosted the Web into international prominence on the Internet.

List of abbreviations

ACB	Access Control Block	DLISAS	Data Language I Separate Address Space
AIB	Application Interface Block	DLI	See DL/1
APA	all points addressable	DL/I	Data Language 1
API	Application Program Interface	DOF	Device Output Format
APPC	Advanced Program-to-Program Communication	DPAGE	Device Page
APPC/MVS	Advanced Program-to-Program Communication/Multiple Virtual Storage	ECSA	Extended Common System Area
BIN	BiNary	EBCDIC	Extended Binary Coded Decimal Interchange Code
BMP	Batch Message Processing Region	EMH	Expedited Message Handling
CA	Control Area (VSAM)	EOD	End Of Data
CD-ROM	(optically read) Compact Disk - Read Only Memory	EOM	End Of Message
CI	Control Interval (VSAM)	EOS	End Of Segment
CICS	Customer Information Control System (IBM)	ESDS	Entry Sequence Data Set
COBOL	Common Oriented Business Language	ETO	Extended Terminal Option
CQS	Common Queue Server	FF	Full Function Database
CPU	Central Processing Unit	FP	Fast Path Database
CRC	Command Recognition Character	FPU	Fast Path Utility
CSA	Common System Area	FSE	Free Space Element
DASD	Direct Access Storage Device	FSEAP	Free Space Element Anchor Point
DBCTL	DataBase Control Subsystem	GSAM	Generalize Sequential Access Method
DB	DataBase	HDAM	Heirarchical Direct Access Method
DB/DC	DataBase manager/Data Communication manager system	HIDAM	Heirarchical Index Direct Access Method
DBD	Database Descriptor Block	HISAM	Heirarchical Index Sequential Access Method
DBMS	DataBase Management System	HSSP	High Speed Sequential Processing
DBRC	DataBase Recovery Control	IBM	International Business Machines Corporation
DBT	DBCTL Thread	IFP	Fast Path Region
DB2	DataBase 2	IMS	Information Management System
DCCTL	Data Communication Control Subsystem	IMS/ESA	Information Management System/Enterprise Systems Architecture
DD	Data Definition JCL statement	INTERNET	a worldwide network of TCP/IP-based networks
DEDB	Data Entry Data Base	IRLM	Inter Region Lock Manager
DIF	Device Input Format	ISC	Inter-System Communications

ITSO	International Technical Support Organization	RAP	Root Anchor Point
ISO	International Organization for Standardization	RBA	Relative Byte Address
ITSO	International Technical Support Organization	REXX	Restructured Extended eXecutor Language
JCL	Job Control Language (MVS and VSE)	RLDS	Recovery Log Data Set
KSDS	Key Sequence Data Set	RSR	Remote Site Recovery
LBG	Load Balancing Group	SB	Sequential Buffering
LPAGE	Logical Page	SLDS	System Log Data Set
LPAR	Logical PARTitioning	SPA	Scratch Pad Area
LTERM	Logical TERMinal	SNA	Systems Network Architecture
LU	Logical Unit	SSA	Segment Search Argument
MID	Message Input Descriptor	SVC	SuperVisor Call routine
MFS	Message Format Services	SYSPLEX	SYStems comPLEX
MOD	Message Output Descriptor	TCB	Task Control Block (MVS control block)
MPP	Message Processing Program	TCP	Transmission Control Protocol (USA, DoD)
MPR	Message Processing Region	TCPIP	Transmission Control Protocol / Internet Protocol
MQ	Message and Queueing (IBM software)	TM	Transaction Manager
MSC	Multiple Systems Coupling	TP	Transaction Program/process (OSI)
MVS	Multiple Virtual Storage (IBM System 370 & 390)	TSO	Time Sharing Option
MVS/ESA	Multiple Virtual Storage/Enterprise Systems Architecture (IBM)	UOW	Unit of Work
NCP	Network Control Program	USERID	USER IDentification
NT	Microsoft Windows NT	VNET	Virtual NETwork
OLDS	Online Log Data Set	VSAM	Virtural Sequential Access Method
OSAM	Overflow Sequential Access Method	VSO	Virtual Storage Option
OS/390	Operating System 390	VT	Virtual Terminal (OSI)
OTMA	Open Transaction Manager Access	VTAM	Virtual Telecommunications Access Method (IBM) (runs)
PCB	Program Communicatoin Block	WADS	Write Ahead Data Set
PROC	PROCedure	XCF	Cross-system Coupling Facility (MVS)
PROCLIB	PROCedure LIBrary (IBM System/360)	XRF	eXtended Recovery Facility
PSB	Program Specification Block		
PTF	Program Temporary Fix		
QSAM	SeQUential Access Method		
RAA	Root Addressable Area		
RACF	Resource Access Control Facility		

Index

Numerics

3270 35, 159, 168

A

abbreviations 271
abnormal termination 148
ACB 79
ACBGEN 140, 141, 145
access intent 209
access paths 64, 71, 88
acronyms 271
AIB 142
AMS 212, 214
API 142
APPC 24, 49
APPLCTN 40, 41
application dependent regions 22
application interface block 142
application program 133, 135, 140, 142, 147, 150, 173
application programming 142
APPLTN 133

B

backup 215
batch message processing 141, 150
batch message processing regions 22
BMH 16, 18
BMP 14, 16, 22, 27, 32, 45, 57, 99, 141, 148, 149, 150, 208

C

CATDS 98
change destination 148
checkpoint 34, 59, 143, 149, 200, 201, 203
CHKP 93, 99, 100, 143, 201, 203
CHNG 148
CICS 6, 14, 18, 25, 49, 142
COMM 54
command codes 177, 185
COMPAT 141, 144
concatenated keys 139
control region 13, 133
conversational processing 58, 149
conversational transaction 143, 149
conversational transactions 151, 154
CQS 15
CSA 23, 33

D

database authorization 209
database calls 142, 173, 174
database descriptor block 140
database manager 3, 5, 14, 23
DB/DC 11, 14, 28

DB2 3, 7, 9, 11, 14, 20, 145
DBA 67, 208
DBCTL 7, 11, 14, 20, 25, 28
DBD 79, 83, 84, 95, 140, 145, 194, 198
DBMS 67, 69
DBRC 15, 20, 207
DBT 16, 17
DCCTL 11, 14, 15, 28
DEDB 7, 15, 79, 81, 82, 86, 89
delete 143, 174
DELETE.LOG INACTIVE 214
deleting segments 143, 183, 196
DLET 143, 174, 183
DLISAS 16, 31
dynamic allocation 213

E

ETO 28, 36

F

Fast Path 20, 22, 39, 47, 82, 89
FORCER 207, 208
FPU 16, 18

G

GENJCL 216
get hold next 142, 174
get hold unique 142, 174
get next 142, 148, 174
get unique 142, 148, 174
GHN 142
GHU 142, 174
GN 142, 148, 174, 179, 180
GSAM 17, 81, 92, 189, 200, 201
GU 142, 148, 174, 179

H

HDAM 79, 81, 83, 84, 89, 94, 179, 197
HIDAM 79, 81, 86, 88, 93, 95, 197
hierachical model 69
HISAM 81, 93
HSSP 16, 18

I

IBM Data Propagator 9
IDCAMS 89
IFP 14, 16, 17
image copy 216
IMS commands 133
IMSGEN 15, 209
IMSID 21
insert 142, 148, 174
inserting segments 142, 184, 196
IRLM 19, 20, 21, 58, 102
ISC 6, 25, 49

ISRT 142, 148, 174, 184
ITOC 51

L

LIST.RECON STATUS 215
log archive 218
log control 217
logging 56, 59, 149
logical relationships 72, 76, 197
LTERM 33, 38, 143, 148, 149

M

message 31, 32, 35, 135
message format service 157
message processing program 133, 141
Message Processing Region 20
message processing region 133
message processing regions 22
message queue 37, 38
messages 166
MFS 33, 36, 154, 157
MPP 14, 16, 32, 33, 38, 55, 57, 133, 141, 143, 147,
148, 149, 150, 153, 154, 157
MPR 20, 22, 32, 42, 43, 133
MQSeries 51
MSC 6, 49
MSDB 89
MSGTYPE 40

N

non-conversational transaction 143
non-response transactions 151

O

ODBA 14
OLDS 98, 149
online transaction 133
OSAM 83, 96
OTMA 6, 23, 40, 49

P

PCB 135, 141, 143, 184, 194
PCB mask 134
PI 56, 102
PRILOG 214, 217
processing intent 144
PROCLIB 20
program control block 135
Program Isolation 34, 102
program specification block 135, 140
PSB 56, 79, 133, 135, 140, 141, 143, 148, 194, 209
PSBGEN 140, 144

R

RACF 24, 28, 54
RBA 86

RECON 9, 15, 210, 212, 214, 215, 216
RECON backup 214, 216, 217
RECON Reorganization 215
RECOVCTL 208
recovery 211
recovery control 212, 217
REPL 143, 174, 182
replace 143, 174
replacing segments 143, 182, 195
Resource Translate Table 145
response transactions 151
Restart 27
retrieving segments 142, 179, 194
REXX 142
RLDS 98
root segment 142
RRA 83
RSR 10, 11, 28
RTT 145

S

Scheduling 41
scheduling 33, 40
search field 194
secondary index 72, 76, 156, 194, 196, 198
segment search arguments 135, 175, 177
share control 212, 217
SHARECTL 208, 212
shared queue 39
shared queues 45
SHISAM 81
Shutdown 28
SLDS 98
SMS 98
SNA 13, 25
SPA 147, 149
spool 54
SSA 135, 174, 175, 176, 184
status code 140, 177, 197
subsystem 211
subsystems 21
Sysplex 39

T

TCP/IP 51
TRAN 33
TRANSACT 40, 41, 42
Transaction 32
transaction 133
transaction manager 3, 5, 14, 23

U

updating segments 143, 182

V

VSAM 83, 86, 88, 89, 93, 96
VSO 91
VTAM 6, 11, 24, 49

W

WADS 98

X

XDFLD 194

XRF 10, 28

XRST 93, 100, 201

ITSO redbook evaluation

IMS Primer
SG24-5352-00

Your feedback is very important to help us maintain the quality of ITSO redbooks. **Please complete this questionnaire and return it using one of the following methods:**

- Use the online evaluation form found at <http://www.redbooks.ibm.com>
- Fax this form to: USA International Access Code + 1 914 432 8264
- Send your comments in an Internet note to redbook@us.ibm.com

Which of the following best describes you?

☐ **Customer** ☐ **Business Partner** ☐ **Solution Developer** ☐ **IBM employee**
☐ **None of the above**

Please rate your overall satisfaction with this book using the scale:
(1 = very good, 2 = good, 3 = average, 4 = poor, 5 = very poor)

Overall Satisfaction _____

Please answer the following questions:

Was this redbook published in time for your needs? Yes___ No___

If no, please explain:

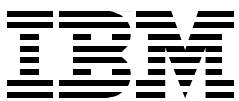
What other redbooks would you like to see published?

Comments/Suggestions: (THANK YOU FOR YOUR FEEDBACK!)

SG24-5352-00

Printed in the U.S.A.

IMS Primer



SG24-5352-00