DFSMS/MVS Version 1 Release 5

DFSMSdfp Advanced Services

SC26-4921-04

DFSMS/MVS Version 1 Release 5

DFSMSdfp Advanced Services

SC26-4921-04

Note!

Before using this information and the product it supports, be sure to read the general information under “Notices” on page xiii.

Fifth Edition (March 1999)

This edition applies to Version 1 Release 5 of DFSMS/MVS (5695-DF1), Release 7 of OS/390 (5647-A01), and any subsequent releases until otherwise indicated in new editions. Make sure you are using the correct edition for the level of the product.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address given below.

A form for readers' comments appears at the back of this publication. If the form has been removed, address your comments to:

International Business Machines Corporation
RCF Processing Department
G26/050
5600 Cottle Road
SAN JOSE, CA 95193-0001
U.S.A.

Or you can send your comments electronically to starpubs@vnet.ibm.com.

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

© **Copyright International Business Machines Corporation 1979, 1999. 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

Notices . . . xiii
Programming Interface Information . . . xiii
Trademarks . . . xiii

About This Book . . . xv
Required Product Knowledge . . . xv
How to Tell if this Book is Current . . . xv
Referenced Publications . . . xvi
References to Product Names Used in DFSMS/MVS Publications . . . xvii
How to Read Syntax Diagrams . . . xviii
Address and Register Conventions . . . xx

Summary of Changes . . . xxiii
Fifth Edition, March 1999 . . . xxiii
Fourth Edition, June 1997 . . . xxiii
Third Edition, December 1995 . . . xxiv
Service Update to Version 1 Release 3, September 1996 . . . xxiv

Chapter 1. Using the Volume Table of Contents . . . 1
VTOC Components . . . 1
Data Set Control Block (DSCB) Types . . . 2
Allocating and Releasing DASD Space . . . 13
The VTOC Index . . . 13
VTOC Index Records . . . 14
Accessing the VTOC with DADSM Macros . . . 15
Requesting DASD Volume Information Using LSPACE . . . 16
Reading DSCBs from the VTOC Using OBTAIN . . . 24
Releasing Unused Space from a DASD Data Set Using PARTREL . . . 27
Creating (Allocating) a DASD Data Set Using REALLOC . . . 34
Accessing the VTOC with CVAF Macros . . . 40
Serializing and Updating . . . 40
Identifying the Volume . . . 41
Generating a CVPL (CVAF Parameter List) . . . 41
Using Buffer Lists . . . 43
Using Macro ICVEDT02 to Map the Extents Area . . . 45
Accessing the DSCB Directly . . . 46
Accessing DSNs or DSCBs in Sequential Order . . . 47
Reading Sets of DSCBs with CVAF Filter . . . 49
VTOC Error Responses . . . 55
Recovering from System or User Errors . . . 56
GTF Trace . . . 56
VTOC and VTOC Index Listings . . . 56

Chapter 2. Managing the Volume Table of Contents . . . 57
Creating the VTOC and VTOC Index . . . 57
Protecting the VTOC and VTOC Index . . . 57
RACF . . . 57
APF . . . 57
Password Protection . . . 58
Copying/Restoring/Initializing the VTOC . . . 58

© Copyright IBM Corp. 1979, 1999

Volumes Containing a Nonindexed VTOC . . . 58
Volumes Containing an Indexed VTOC . . . 58
Accessing the VTOC with RENAME and SCRATCH . . . 58
Deleting a Data Set from the VTOC . . . 59
SCRATCH and CAMLST Macro Specification . . . 61
Renaming a Data Set in the VTOC . . . 64
RENAME and CAMLST Macro Specification . . . 65

Chapter 3. Using Catalog Management Macros . . . 69
Application Program Considerations . . . 69
Catalog Search Order . . . 70
Retrieving Information from a Catalog . . . 70
Retrieving Information by Data Set Name (LOCATE and CAMLST NAME) . 71
Retrieving Information by Generation Data Set Name (LOCATE and CAMLST NAME) . . . 73
Retrieving Information by Alias (LOCATE and CAMLST NAME) . . . 74
Reading a Block by Relative Block Address (LOCATE and CAMLST BLOCK) 76
Building and Deleting Indexes . . . 78
Building an Index (INDEX and CAMLST BLDX) . . . 78
Building a Generation Index (INDEX and CAMLST BLDG) . . . 80
Deleting an Index (INDEX and CAMLST DLTX) . . . 82
Assigning an Alias for an Index (INDEX and CAMLST BLDA) . . . 83
Deleting an Alias for an Index (INDEX and CAMLST DLTA) . . . 84
Connecting and Disconnecting CVOLs . . . 85
Connecting CVOLs (INDEX and CAMLST LNKX) . . . 85
Disconnecting CVOLs (INDEX and CAMLST DRPX) . . . 86
Using Non-VSAM Data Set Catalog Entries . . . 88
Cataloging a Non-VSAM Data Set (CATALOG and CAMLST CAT) . . . 88
Uncataloging a Non-VSAM Data Set (CATALOG and CAMLST UNCAT) . . 90
Recataloging a Non-VSAM Data Set (CATALOG and CAMLST RECAT) . . 91
CVOL Processor Return Codes . . . 94

Chapter 4. Executing Your Own Channel Programs . . . 95
Using EXCP . . . 95
31-Bit IDAW Requirements . . . 96
EXCP Requirements . . . 97
Channel Program . . . 97
Control Blocks Used with EXCP . . . 97
Executing the Channel Program . . . 98
Initiating the Channel Program . . . 99
Virtual and Central Storage Addresses . . . 99
Device Type Considerations . . . 100
Completing Execution . . . 101
Using Executable Macro Instructions with EXCP . . . 105
Initializing Data Control Blocks . . . 105
Executing a Channel Program . . . 106
End of Volume . . . 106
Restoring Data Control Blocks . . . 108
Executing Fixed Channel Programs in Real Storage (EXCPVR) . . . 108
Control Block Fields . . . 110
Data Control Block (DCB) Fields . . . 110
Data Control Block Extension . . . 120
Input/Output Block Fields . . . 120
Input/Output Block Common Extension Fields . . . 123

Input/Output Error Data Block Fields . . . 124
Event Control Block Fields . . . 124
Data Extent Block Fields . . . 126
EXCP and EXCPVR Appendages . . . 126
Making Appendages Available to the System . . . 128
The Authorized Appendage List (IEAAPP00) . . . 128
Start-I/O Appendage . . . 130
Page Fix and EXCPVR Start I/O Appendage . . . 130
Program-Controlled Interruption Appendage . . . 133
End-of-Extent Appendage . . . 134
Abnormal-End Appendage . . . 134
Channel-End Appendage . . . 136
Converting a Relative Track Address to an Actual Track Address . . . 136
Return Codes from the Conversion Routine . . . 137
Converting an Actual Track Address to a Relative Track Address . . . 137
Obtaining the Sector Number of a Block on an RPS Device . . . 138

Chapter 5. Using XDAP to Read and Write to Direct Access Devices . . . 141
Using XDAP . . . 141
Macro Instructions Used with XDAP . . . 142
Executing Direct Access Programs . . . 143
Control Blocks Used with XDAP . . . 145

Chapter 6. Using Password Protected Data Sets . . . 147
Providing Data Set Security . . . 148
PASSWORD Data Set Characteristics . . . 149
Creating Protected Data Sets . . . 150
Protection Feature Operating Characteristics . . . 151
Maintaining the PASSWORD Data Set Using PROTECT . . . 152
Record Format . . . 152
Protection-Mode Indicator . . . 152
PROTECT Macro Specification . . . 153

Chapter 7. Using System Macro Instructions . . . 159
Ensuring Data Security by Validating the Data Extent Block . . . 159
DEBCHK Macro Specification . . . 160
Obtaining I/O Device Characteristics . . . 163
DEVTYPE Macro Specification . . . 164
Reading and Modifying a Job File Control Block . . . 178
RDJFCB Macro Specification . . . 180
DEQ at Demount Facility for Tape Volumes . . . 190
High-Speed Cartridge Tape Positioning . . . 192
OPEN—Initialize Data Control Block for Processing the JFCB . . . 193
Purging and Restoring I/O Requests . . . 194
PURGE Macro Specification . . . 195
RESTORE Macro Specification . . . 198
Performing Track Calculations . . . 198
TRKCALC Macro Specification . . . 199
Determining Version, Release, and Modification Level of DFSMS/MVS . . . 207
Determining DFARELS During Assembler Macro Phase . . . 207

Chapter 8. Displaying Messages on Cartridge Magnetic Tape Subsystems . . . 211
MSGDISP Macro Specification . . . 211

MSGDISP—Displaying a Mount Message 211

Chapter 9. Using DFSMSdfp Callable Services 225
Character Data Representation Architecture (CDRA) APIs 226
Call for DFSMS/MVS Level Determination 226
Format 226
Parameters 227
Return Codes 228
Example 228
Call for Data Set Attribute Retrieval 229
Format 229
Parameters 230
Return Codes 231
Example 231
Call for Data Set Backup-While-Open Support 231
Format 231
Parameters 232
Return Codes 233
Example 233
Using the Backup-While-Open Facility 234
Call for DFSMSdfp Share Attributes 237
Format 237
Parameters 238
Return Codes 238
IGWASYS, IGWASMS, IGWABWO, IGWLSHR Return and Reason Codes 239
Call for Record-Level Sharing Query 240
Format 240
Parameters 240
Return Codes 242
Example 243

Chapter 10. Establishing Installation Defaults for Binder Options 245

Chapter 11. Using the DESERV Exit 247
Describing Task and Global Exits 248
Task Level Exit 248
Global Exit 249
Interactions Between the Task Level and Global Exits 249
Establishing Multiple Task level or Multiple Global Exits 250
Issuing DESERV FUNC=EXIT (invocation environment) 250
Invocation Syntax 251
Installing or Replacing the DESERV Exit 253
Deleting the DESERV Exit 255
Determining If a DESERV Exit Is Active 255
Writing the DESERV Exit 255
Exit Processing 255
Parameters Related to the GET Function 256
Parameters Related to the PUT Function 272
PUT Return and Reason Codes 274
Parameters Related to the DELETE Function 278
Parameters Related to the RENAME Function 280
Parameters Related to the UPDATE Function 282
Entry Environment for Exit Routine 284
Exit Environment for Exit routine 284

Registers on Entry to the DESERV Exit . . . 285
Registers on Return from the DESERV Exit . . . 285
DESERV Exit Return and Reason Codes . . . 285
DESERV FUNC=EXIT Return and Reason Codes . . . 286
Example of the DESERV Exit . . . 288

Chapter 12. Maintaining the System Image Library . . . 295
UCS Images in SYS1.IMAGELIB . . . 297
Examples of UCS Image Coding . . . 299
UCS Image Alias Names . . . 302
UCS Image Tables in SYS1.IMAGELIB . . . 302
Alias Names in UCS Image Tables . . . 303
Adding or Modifying a UCS Image Table Entry . . . 307
Verifying the UCS Image . . . 310
FCB Images in SYS1.IMAGELIB . . . 311
Adding an FCB Image to the Image Library . . . 313
Modifying an FCB Image . . . 316
JES Support for the 3211 Indexing Feature . . . 317

Appendix A. CVAF VTOC Access Macros . . . 319
Coding CVAF Macro Instructions . . . 319
CVAFDIR Macro Overview and Specification . . . 319
CVAFDSM Macro Overview and Specification . . . 330
CVAFFILT Macro Overview and Specification . . . 337
CVAFSEQ Macro Overview and Specification . . . 347
CVAFTST Macro Overview and Specification . . . 359
VTOC Index Error Message and Associated Codes . . . 360
Error Message . . . 360
Codes Put in the CVSTAT Field . . . 361

Appendix B. ANTRQST Macro — Call to the System Data Mover API . 363
Environment . . . 363
Programming Requirements . . . 364
Restrictions . . . 364
Input Register Information . . . 364
Output Register Information . . . 364
Performance Implications . . . 365
Syntax . . . 365
Parameters . . . 371
Subparameters for REQUEST . . . 372
Character Filtering . . . 407
ABEND Codes . . . 407
Return and Reason Codes . . . 407
Example: Getting the Current Consistency Time for XRC Session TUCSON in Assembler . . . 416

Appendix C. Control Blocks . . . 419
Data Extent Block (DEB) Fields . . . 419
Data Facilities Area (DFA) Fields . . . 425

Glossary . . . 429

Index . . . 433

Figures

1. Locating the Volume Table of Contents . . . 1
2. Contents of VTOC—DSCBs Describing Data Sets . . . 3
3. DSCB Format-1 . . . 4
4. DSCB Format-2 . . . 8
5. DSCB Format-3 . . . 10
6. DSCB Format-4 . . . 10
7. DSCB Format-5 . . . 12
8. Example of the Relationship of a VTOC to Its Index . . . 14
9. Format of the LSPACE Parameter List (MF=D) . . . 20
10. DADSM LSPACE Free Space Information MF=(D,MSG) . . . 22
11. DADSM LSPACE Message Return Area Contents . . . 22
12. DADSM LSPACE Free Space Information Format, MF=(D,EXPMSG) . . . 23
13. LSPACE Data Return Area Format . . . 23
14. DADSM OBTAIN Return Codes . . . 25
15. DADSM OBTAIN Return Codes . . . 27
16. DADSM PARTREL Return Codes . . . 32
17. DADSM CREATE Return Codes . . . 39
18. Format of the CVAF Parameter List . . . 42
19. CVFCTN Field of CVPL—Contents and Definitions . . . 43
20. Format of a Buffer List Header . . . 44
21. Format of a Buffer List Entry . . . 45
22. Format of ICVEDT02 Mapping Macro . . . 45
23. Control Blocks Required for CVAF Filter Services . . . 50
24. Format of a Filter Criteria List Header . . . 50
25. Format of a Filter Criteria List Entry . . . 52
26. SCRATCH Return Codes . . . 62
27. SCRATCH Status Codes . . . 63
28. Secondary Status Codes . . . 64
29. DADSM RENAME Return Codes . . . 67
30. RENAME Status Codes . . . 67
31. LOCATE Return Codes . . . 77
32. INDEX Return Codes . . . 88
33. CATALOG Return Codes . . . 93
34. CVOL Processor Return Codes . . . 94
35. DCBOFLGS Usage . . . 107
36. Data Control Block Format for EXCP (After OPEN) . . . 112
37. Device-dependent portion of the DCB with DEVD=DA and DSORG=PS (or DSORG=PO) . . . 117
38. Device-dependent portion of the DCB with DEVD=DA and DSORG=DA 118
39. Device-dependent portion of the DCB with DEVD=TA and DSORG=PS 118
40. Device-dependent portion of the DCB with DEVD=PR and DSORG=PS 119
41. Device-dependent portion of the DCB with DEVD=PC or RD and DSORG=PS . . . 119
42. Input/Output Block (IOB) Format . . . 121
43. Format of an IOBE . . . 124
44. Format of an IEDB . . . 124
45. Event Control Block after Posting of Completion Code . . . 125
46. EXCP Appendages . . . 126
47. Entry Points, Returns, and Available Work Registers for Appendages . 127
48. Form for MBBCCHR . . . 137

© Copyright IBM Corp. 1979, 1999

49. Convert Routine Return Codes 137
50. Registers and Their Use for Converting Actual to Relative 138
51. Registers and Their Use for A Sector Convert Routine 139
52. Channel Program Command Words Used by XDAP 146
53. Parameter List for ADD Function 154
54. Parameter List for REPLACE Function 155
55. Parameter List for DELETE Function 156
56. Parameter List for LIST Function 157
57. PROTECT Return Codes 157
58. Minimum size of area 166
59. Simulated Device UCBTYP Information 171
60. Simulated Device Characteristics Information 173
61. Output from DEVTYPE Macro 175
62. Output from DEVTYPE Macro — DASD Devices 175
63. Examples of Standard Form of the RDJFCB macro. 180
64. Example Code Using RDJFCB Macro 181
65. Return Codes from the RDJFCB Macro 182
66. Processing a Multivolume Direct Data Set with RDJFCB 185
67. Format of the Allocation Retrieval List 187
68. Format of the Allocation Retrieval Area 188
69. Sample Code Retrieving Allocation Information 189
70. Examples of Standard Form of the OPEN TYPE=J Macro 194
71. The IOB Chain 198
72. Sample &IHADFARELS Program 208
73. Example of Determining Symbol Definition 209
74. Example of an IGWASYS Call Statement 229
75. Example of an IGWASMS Call LINK Statement 231
76. Example of IGWABWO Using LOAD and CALL Statements 234
77. Backup-While-Open Indicators 237
78. IGWASYS, IGWASMS, IGWABWO, IGWLSHR Return and Reason Codes 239
79. IGWARLS Return and Reason Codes 242
80. Example of the IGWARLS Query Call Using LOAD and CALL Statements 243
81. Exit Routine Call Sequence 250
82. DESERV Screen Table Structure 254
83. Deleting the DESERV Exit 255
84. Determining If a DESERV Exit Is Active 255
85. DESX Structure Mapping DESERV Exit Parameter List 256
86. Structure of DESP for DESERV GET Invocations 257
87. DESL Structure 260
88. DESN Parameter List 261
89. DESB Parameter List 261
90. SMDE Format 262
91. Directory Entry Name Section 264
92. Directory Entry Notelist Section (PDS Only) 264
93. Directory Entry Token Section 264
94. Directory Entry Primary Name Section 265
95. Directory Entry Name Section 265
96. LSLoader Attributes Unique to Program Objects 269
97. Attributes Unique to Load Modules (PDS only) 271
98. Alias in Unformatted Form 272
99. DESERV PUT DESP Fields 272
100. PUT Return and Reason Codes 276

101. DESD Parameter List 276
102. DESERV DELETE DESP Fields 278
103. DESERV RENAME DESP Fields 280
104. DESERV UPDATE DESP Fields 282
105. Return and Reason Codes for the Exit DESERV Function 288
106. Establishing and Deleting a Task Level DESERV Exit 289
107. Sample DESERV Exit Routine 291
108. SYS1.IMAGELIB Contents 296
109. Code to Add a 1403 UCS Image to SYS1.IMAGELIB 299
110. Code to Add a 3203 UCS Image to SYS1.IMAGELIB 300
111. Sample Code to Add a 3211 UCS Image to SYS1.IMAGELIB 301
112. UCS Image Table Entry Format 303
113. UCS5 Image Table Contents 304
114. UCS6 Image Table Contents 305
115. 3262 Model 5 Print Bands 306
116. Adding a New Band ID to the 4245 UCS Image Table (UCS5) 309
117. Adding a New Default Entry to the 4248 UCS Image Table (UCS6). . . 310
118. Format of the Standard STD1 FCB Image 312
119. Format of the Standard STD2 FCB Image 313
120. Sample Code to Assemble and Add an FCB Load Module to SYS1.IMAGELIB 315
121. Example of CVAFDIR Macro with VTOC 326
122. Example of Using the CVAFDIR Macro with an Indexed VTOC 329
123. Example of the CVAFFILT Macro 343
124. Example of the CVAFSEQ Macro with an Indexed VTOC 353
125. Example of the CVAFSEQ Macro with a Nonindexed VTOC 356
126. How the ANTRQST Macro Relates to Major OS/390 Functions 363
127. Return Codes for the ANTRQST Macro 408
128. Partial Listing of DEB Fields 419
129. DFA Fields 426

Notices

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 that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of the intellectual property rights of IBM may be used instead of the IBM product, program, or service. The evaluation and verification of operation in conjunction with other products, programs, or services, except those expressly designated by IBM, are the responsibility of the user.

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:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
USA

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
Information Enabling Requests
Dept. DWZ
5600 Cottle Road
San Jose, CA 95193

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

Any pointers in this publication to non-IBM Web sites are provided for convenience only, and do not in any manner serve as an endorsement of these Web sites.

Programming Interface Information

This book is intended to help you to modify and extend the data management capabilities of the operating system. General-use programming interfaces allow the customer to write programs that obtain the services of DFSMS/MVS.

Trademarks

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

© Copyright IBM Corp. 1979, 1999

CICS
CICS/ESA
CICS/MVS
DFSMS
DFSMS/MVS
DFSMSdfp
DFSMShsm
ECKD
IBM
IPDS
MVS/ESA
OS/390
PSF
RACF
RETAIN
Systems Application Architecture
VTAM

UNIX is a registered trademark in the United States and/or other countries licensed exclusively through X/Open Company Limited.

Other company, product, and service names, which may be denoted by a double asterisk (**), may be trademarks or service marks of others.

About This Book

This book is intended to help system programmers modify and extend the data management capabilities of the operating system and for programmers to write advanced application programs.

Required Product Knowledge

To use this book effectively, you should be familiar with:

- Assembler language
- Standard program linkage conventions
- The utility programs IEHLIST and IEHPROGM
- Data management access methods and macro instructions
- The storage management functions provided by DFSMSdss and DFSMShsm. DFSMSdss moves data from one device to another, backs up and recovers data sets, and reduces free-space fragmentation on DASD volumes. DFSMShsm manages storage by migrating and recalling data sets through a hierarchy of storage devices.

  To learn about physical storage administration, see *DFSMS/MVS DFSMSdfp Storage Administration Reference*, SC26-4920.

See “Referenced Publications” on page xvi for further information and specific book titles.

How to Tell if this Book is Current

IBM regularly updates its books with new and changed information. When first published, both hardcopy and BookManager softcopy versions of a book are identical, but subsequent updates might be available in softcopy before they are available in hardcopy. Here's how to determine the level of a book:

- Check the book's order number suffix (often referred to as the dash level). A book with a higher dash level is more current than one with a lower dash level. For example, in the publication order number SC26-4930-02, the dash level 02 means that the book is more current than previous levels, such as 01 or 00. Suffix numbers are updated as a product moves from release to release, as well as for hardcopy updates within a given release.
- Check to see if you are using the latest softcopy version. To do this, compare the last two characters of the book's file name (also called the book name). The higher the number, the more recent the book. For example, DGT1U302 is more recent than DGT1U301.
- Compare the dates of the hardcopy and softcopy versions of the books. Even if the hardcopy and softcopy versions of the book have the same dash level, the softcopy could be more current. This will not be apparent from looking at the edition notice. The edition notice number and date remain that of the last hardcopy version. When you are looking at the softcopy product bookshelf, check the date shown to the right of the book title. This will be the date that the softcopy version was created.

© Copyright IBM Corp. 1979, 1999

Also, an asterisk (*) is added next to the new and changed book titles in the CD-ROM booklet and the README files.

Vertical lines to the left of the text indicate changes or additions to the text and illustrations. For a book that has been updated in softcopy only, the vertical lines indicate changes made since the last printed version.

Referenced Publications

In this book, references are made to the following publications:

Publication Title	Order Number
IBM High Level Assembler/MVS & VM & VSE Programmer's Guide	SC26-4941
IBM High Level Assembler/MVS & VM & VSE Language Reference	SC26-4940
OS/390 MVS Assembler Services Guide	GC28-1762
OS/390 MVS Assembler Services Reference	GC28-1910
OS/390 MVS Authorized Assembler Services Guide	GC28-1763
OS/390 MVS Authorized Assembler Services Reference ALE-DYN	GC28-1764
OS/390 MVS Authorized Assembler Services Reference ENF-IXG	GC28-1765
OS/390 MVS Authorized Assembler Services Reference LLA-SDU	GC28-1766
OS/390 MVS Authorized Assembler Services Reference SET-WTO	GC28-1767
OS/390 MVS Planning: Global Resource Serialization	GC28-1759
DFSMS/MVS Program Management	SC26-4916
Enterprise Systems Architecture/390 Principles of Operation	SA22-7201
IBM 2821 Control Unit Component Description	GA24-3312
IBM 3203 Printer Component Description and Operator's Guide	GA33-1515
IBM 3211 Printer, 3216 Interchangeable Train Cartridge, and 3811 Printer Control Unit Component Description and Operator's Guide	GA24-3543
IBM 3262 Model 5 Printer Product Description	GA24-3936
IBM 3800 Printing Subsystem Programmer's Guide	GC26-3846
IBM 3800 Printing Subsystem Model 3 Programmer's Guide: Compatibility	SH35-0051
IBM 3800 Printing Subsystem Models 3 and 8 Programmer's Guide	SH35-0061
3900 Advanced Function Printer Product Description	GA32-0135
IBM 4245 Printer Model 1 Component Description and Operator's Guide	GA33-1541
IBM 4248 Printer Model 1 Description	GA24-3927
IBM 6262 Printer Model 014 Product Description	GA24-4134
IBM 6262 Printer Model 014 User's Guide	GA24-4132
IBM 6262 Printer Print Band Manual	GA24-4131
Device Support Facilities User's Guide and Reference	GC35-0033
OS/390 MVS JCL Reference	GC28-1757
OS/390 MVS JCL User's Guide	GC28-1758
OS/390 JES2 Initialization and Tuning Guide	SC28-1791

Publication Title	Order Number
OS/390 JES3 Initialization and Tuning Guide	SC28-1802
OS/390 MVS Conversion Notebook	GC28-1747
DFSMS/MVS Access Method Services for ICF	SC26-4906
DFSMS/MVS Access Method Services for VSAM	SC26-4905
DFSMS/MVS Checkpoint/Restart	SC26-4907
DFSMS/MVS DFSMSdfp Diagnosis Reference	LY27-9606
DFSMS/MVS Installation Exits	SC26-4908
DFSMS/MVS Macro Instructions for Data Sets	SC26-4913
DFSMS/MVS Planning for Installation	SC26-4919
DFSMS/MVS Using Data Sets	SC26-4922
DFSMS/MVS Using Magnetic Tapes	SC26-4923
DFSMS/MVS Utilities	SC26-4926
MVS/ESA SML: Managing Data	SC26-3124
DFSMS/MVS Implementing System-Managed Storage	SC26-3123
OS/390 MVS Initialization and Tuning Guide	SC28-1751
OS/390 HCD Planning	GC28-1750
OS/390 Security Server (RACF) Introduction	GC28-1912
OS/390 Security Server (RACF) General User's Guide	SC28-1917
OS/390 MVS System Data Set Definition	GC28-1782
OS/390 MVS System Messages, Vol 1 (ABA-ASA)	GC28-1784
OS/390 MVS System Messages, Vol 2 (ASB-EWX)	GC28-1785
OS/390 MVS System Messages, Vol 3 (GDE-IEB)	GC28-1786
OS/390 MVS System Messages, Vol 4 (IEC-IFD)	GC28-1787
OS/390 MVS System Messages, Vol 5 (IGD-IZP)	GC28-1788
OS/390 TSO/E Command Reference	SC28-1969

References to Product Names Used in DFSMS/MVS Publications

DFSMS/MVS publications support DFSMS/MVS, 5695-DF1, as well as the DFSMSdfp base element and the DFSMShsm, DFSMSdss, and DFSMSrmm features of OS/390, 5647-A01. DFSMS/MVS publications also describe how DFSMS/MVS interacts with other IBM products to perform the essential data, storage, program and device management functions of the operating system.

DFSMS/MVS publications typically refer to another IBM product using a generic name for the product. When a particular release level of a product is relevant, the reference includes the complete name of that product. This section explains the naming conventions used in the DFSMS/MVS library for the following products:

MVS can refer to:

- MVS/ESA SP Version 5, 5695-047 or 5695-048
- The MVS base control program (BCP) of OS/390, 5647-A01

All MVS book titles used in DFSMS/MVS publications refer to the OS/390 editions. Users of MVS/ESA SP Version 5 should use the corresponding MVS/ESA book. Refer to *OS/390 Information Roadmap* for titles and order numbers for all the elements and features of OS/390.

For more information about OS/390 elements and features, including their relationship to MVS/ESA SP and related products, please refer to *OS/390 Planning for Installation*.

RACF can refer to:

- Resource Access Control Facility (RACF), Version 2, 5695-039
- The RACF element of the OS/390 Security Server, an optional feature of OS/390

All RACF book titles refer to the Security Server editions. Users of RACF Version 2 should use the corresponding book for their level of the product. Refer to *OS/390 Security Server (RACF) Introduction* for more information about the Security Server.

CICS can refer to:

- CICS/MVS, 5665-403
- CICS/ESA, 5685-083
- The CICS element of the CICS Transaction Server for OS/390, 5665-147

All CICS book titles refer to the CICS Transaction Server for OS/390 editions. Users of CICS/MVS and CICS/ESA should use the corresponding books for those products. Please see *CICS Transaction Server for OS/390: Planning for Installation* for more information.

How to Read Syntax Diagrams

Throughout this library, diagrams are used to illustrate the programming syntax. Keyword parameters are parameters that follow the positional parameters. Unless otherwise stated, keyword parameters can be coded in any order. The following list tells you how to interpret the syntax diagrams:

- Read the diagrams from left-to-right, top-to-bottom, following the main path line. Each diagram begins on the left with double arrowheads and ends on the right with two arrowheads facing each other.

```
►►─┤ Syntax Diagram ├──────────────────────────────────────────►◄
```

- If a diagram is longer than one line, each line to be continued ends with a single arrowhead and the next line begins with a single arrowhead.

```
►►─┤ First Line ├──────────────────────────────────────────────►

►─┤ Second Line ├─────────────────────────────────────────────►

►─┤ Last Line ├───────────────────────────────────────────────►◄
```

- Required keywords and values appear on the main path line. You must code required keywords and values.

```
►►──REQUIRED_KEYWORD──────────────────────────────────────────────────►◄
```

  If several mutually exclusive required keywords or values exist, they are stacked vertically in alphanumeric order.

```
►►──┬─REQUIRED_KEYWORD_OR_VALUE_1─┬──────────────────────────────────►◄
    └─REQUIRED_KEYWORD_OR_VALUE_2─┘
```

- Optional keywords and values appear below the main path line. You can choose not to code optional keywords and values.

```
►►──┬──────────────┬──────────────────────────────────────────────────►◄
    └─KEYWORD──────┘
```

  If several mutually exclusive optional keywords or values exist, they are stacked vertically in alphanumeric order below the main path line.

```
►►──┬──────────────────────────┬──────────────────────────────────────►◄
    ├─KEYWORD_OR_VALUE_1───────┤
    └─KEYWORD_OR_VALUE_2───────┘
```

- An arrow returning to the left above a keyword or value on the main path line means that the keyword or value can be repeated. The comma means that each keyword or value must be separated from the next by a comma.

```
►►──┬─┬──────────────────┬──────────────────────────────────────────►◄
    │ │ ,                │
    │ ▼                  │
    └─REPEATABLE_KEYWORD─┘
```

- An arrow returning to the left above a group of keywords or values means more than one can be selected, or a single one can be repeated.

```
►►──┬──────────────────────────────────────────────────────────────────►◄
    │ ┌─┬──────────────────────────────────┬──┐
    │ │ ,                                  │  │
    │ ▼                                    │  │
    │   ┌─REPEATABLE_KEYWORD_OR_VALUE_1────┤  │
    │   └─REPEATABLE_KEYWORD_OR_VALUE_2────┘  │
    └─────────────────────────────────────────┘
```

- A word in all uppercase is a keyword or value you must spell exactly as shown. In this example, you must code ***KEYWORD***.

```
►►──KEYWORD───────────────────────────────────────────────────────────►◄
```

  If a keyword or value can be abbreviated, the abbreviation is discussed in the text associated with the syntax diagram.

- If a diagram shows a character that is not alphanumeric (such as parentheses, periods, commas, and equal signs), you must code the character as part of the syntax. In this example, you must code ***KEYWORD=(001,0.001)***.

```
►►──KEYWORD=(001,0.001)───────────────────────────────────────────────►◄
```

- If a diagram shows a blank space, you must code the blank space as part of the syntax. In this example, you must code ***KEYWORD=(001 FIXED)***.

```
►►──KEYWORD=(001 FIXED)───────────────────────────────────────────────►◄
```

- Default keywords and values appear above the main path line. If you omit the keyword or value entirely, the default is used.

```
►►─┬─DEFAULT─┬──────────────────────────────────────────────────────►◄
   └─KEYWORD─┘
```

- A word in all lowercase italics is a *variable*. Where you see a variable in the syntax, you must replace it with one of its allowable names or values, as defined in the text.

```
►►─variable──────────────────────────────────────────────────────────►◄
```

- References to syntax notes appear as numbers enclosed in parentheses above the line. Do not code the parentheses or the number.

```
►►─KEYWORD(1)────────────────────────────────────────────────────────►◄
```

Note:
1 An example of a syntax note.

- Some diagrams contain *syntax fragments*, which serve to break up diagrams that are too long, too complex, or too repetitious. Syntax fragment names are in mixed case and are shown in the diagram and in the heading of the fragment. The fragment is placed below the main diagram.

```
►►─┤  Reference to Syntax Fragment ├──────────────────────────────────►◄
```

Syntax Fragment:

```
├─1ST_KEYWORD,2ND_KEYWORD,3RD_KEYWORD──────────────────────────────┤
```

Address and Register Conventions

The notation used to code an operand is as follows:

symbol
The operand can be any valid assembler-language symbol.

(*0*) General register 0 can be used as an operand. When used as an operand in a macro instruction, the register must be specified as the decimal number 0 enclosed in parentheses as shown.

(*1*) General register 1 can be used as an operand. When used as an operand in a macro instruction, the register must be specified as the decimal number 1 enclosed in parentheses as shown. When you use register 1, the instruction that loads it is not included in the macro expansion.

(*2-12*)
The operand specified can be any of the general registers 2 through 12. All registers as operands must be coded in parentheses; for example, if register 3 is coded, it is coded as **(3)**. A register from 2 through 12 can be coded as a decimal number, symbol (equated to a decimal number), or an expression that results in a value of 2 through 12.

RX-Type Address
The operand can be specified as any valid assembler-language RX-type address as shown in the following examples:

Name	Operation	Operand
ALPHA1	L	1,39(4,10)

Name	Operation	Operand
ALPHA2	L	REG1,39(4,TEN)
BETA1	L	2,ZETA(4)
BETA2	L	REG2,ZETA(REG4)
GAMMA1	L	2,ZETA
GAMMA2	L	REG2,ZETA
GAMMA3	L	2,=F'1000'
LAMBDA1	L	3,20(,5)

Both ALPHA instructions specify explicit addresses; REG1 and TEN have been defined as absolute symbols. Both BETA instructions specify implied addresses, and both use index registers. ZETA is a relocatable symbol. Indexing is omitted from the GAMMA instructions. GAMMA1 and GAMMA2 specify implied addresses. The second operand of GAMMA3 is a literal. LAMBDA1 specifies an explicit address with no indexing.

A-Type Address
The operand can be specified as any address that can be written as a valid assembler-language A-type address constant. An A-type address constant can be written as an absolute value, a relocatable symbol, or relocatable expression. Operands requiring an A-type address are inserted into an A-type address constant during the macro expansion process. For more details about A-type address constants, see *High Level Assembler/MVS & VM & VSE Language Reference.*

absexp
The operand can be an absolute value or expression. An absolute expression can be an absolute term or an arithmetic combination of absolute terms. An absolute term can be a nonrelocatable symbol, a self-defining term, or the length attribute reference. For more details about absolute expressions, see *High Level Assembler/MVS & VM & VSE Language Reference.*

relexp
The operand can be a relocatable symbol or expression. If the program containing a relocatable symbol or expression is relocated *n* bytes away from its originally assigned area of storage, the value of a relocatable symbol or expression changes by *n*. For more details about relocatable symbols and expressions, see *High Level Assembler/MVS & VM & VSE Language Reference.*

Summary of Changes

This summary of changes includes specific updates to this book as well as product highlights for previous releases.

Fifth Edition, March 1999

This publication is a minor revision in support of the functional changes introduced with DFSMS/MVS Version 1 Release 5. Technical changes or additions to the text and illustrations are indicated by a vertical line to the left of the change.

This revision also includes maintenance and editorial changes.

The following summarizes the changes to that information.

- Two new flags, DFAALVER and DFAALFOR, have been added to the Data Facilities Area (DFA) fields. Applications can use DFAALVER and DFAALFOR to determine whether the installation ISO/ANSI version level is Version 3 or Version 4 and whether or not the version level is "forced."
- A new parameter, KEYREG, has been added to the DEBCHK macro.

Fourth Edition, June 1997

This publication is a minor revision in support of the functional changes introduced with DFSMS/MVS Version 1 Release 4. This book was last published September, 1996, in softcopy only. Vertical lines to the left of the text and illustrations indicate technical changes or additions since the September softcopy version.

This revision also includes maintenance and editorial changes.

The following summarizes the changes to that information:

- Comments on using the IECSDSL1 macro have been added to “Data Set Control Block (DSCB) Types” on page 2.
- HFS and VSAM data set comments have been added to “Format-3 DSCB” on page 9.
- A new bit (DS4DEVAV) and a byte (DS4DEVAC) have been added to the format-4 DSCB. See “Format-4 DSCB” on page 10.
- In “Creating (Allocating) a DASD Data Set Using REALLOC” on page 34, comments on HFS, extended format, and VSAM data sets have been added.
- “Return Codes from REALLOC” on page 39 has been modified for clarity.
- Definitions of fully-qualified and partially-qualified data set names for catalog have been added to “Retrieving Information from a Catalog” on page 70.
- Figure 63 on page 180 and Figure 70 on page 194 have been redrawn to enhance clarity.
- Chapter 10, “Establishing Installation Defaults for Binder Options” on page 245 has been modified for accuracy.
- Figure 86 on page 257 has been modified for clarity.

© Copyright IBM Corp. 1979, 1999

- In “Data Facilities Area (DFA) Fields” on page 425, several modifications have been made for accuracy and clarity.
- Minor changes to the DESERV exit interface have been documented. If you have implemented a DESERV EXIT routine, these changes may require that you make some coding changes to your exit.
- Throughout this book, "real address" has been changed to "central storage address."

Third Edition, December 1995

This publication is a major revision in support of the functional changes introduced with DFSMS/MVS Version 1 Release 3. Technical changes or additions to the text and illustrations are indicated by a vertical line to the left of the change. For a book that has been updated in softcopy only, the vertical lines indicate changes made since the last printed version.

This revision also includes maintenance and editorial changes.

The following summarizes the changes to that information.

- The book title has been changed from DFSMS/MVS Using Advanced Services for Data Sets to *DFSMS/MVS Version 1 Release 5 DFSMSdfp Advanced Services*
- In Chapter 9, “Using DFSMSdfp Callable Services” on page 225 we have added the new IGWARLS callable service.
- We have added a new Program Management Chapter. (See Chapter 10, “Establishing Installation Defaults for Binder Options” on page 245.)
- We have added a new chapter on the DESERV Installation Exit. (See Chapter 11, “Using the DESERV Exit” on page 247.)
- We have replaced all previous notational conventions with new railroad syntax diagrams. See “How to Read Syntax Diagrams” on page xviii for instructions on reading these new diagrams.
- We have added a glossary for this edition.

Service Update to Version 1 Release 3, September 1996

Service updates have been added to the book.

Chapter 1. Using the Volume Table of Contents

This chapter is intended to help you to use system macros to access and modify the VTOC and VTOC index. The direct access device storage management (DADSM) routines control space allocation on direct access volumes through the volume table of contents (VTOC) for a volume and through the VTOC index if one exists.

VTOC Components

The (VTOC) is a data set that describes the contents of the direct access volume on which it resides. It is a contiguous data set; that is, it resides in a single extent on the volume and starts after cylinder 0, track 0 and before track 65,535. A VTOC's address is located in the VOLVTOC field of the standard volume label (see Figure 1). A VTOC consists of complete tracks.

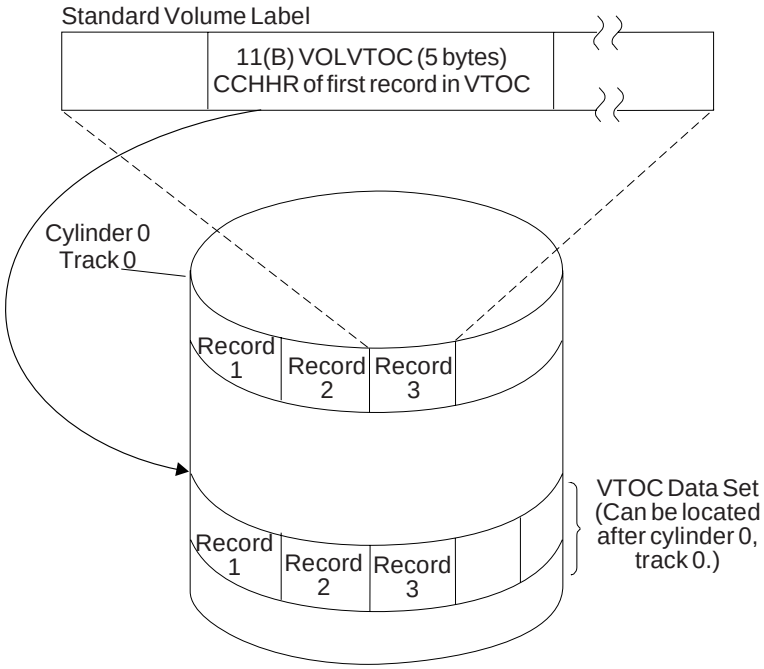


Figure 1. Locating the Volume Table of Contents

The VTOC is composed of 140-byte[1] data set control blocks (DSCBs) that correspond either to a data set or virtual storage access method (VSAM) data space currently residing on the volume, or to contiguous, unassigned tracks on the volume. A special type of data is in an HFS (Hierarchical File System) data set. Each one contains a UNIX type of file system that is compliant with the IEEE POSIX standard and OSF XPG/4.2 standard. Each HFS data set can contain multiple HFS files belonging to multiple users. You can access the files through OS/390 UNIX System Services, BSAM, QSAM, and VSAM. See *DFSMS/MVS Using Data Sets*. DSCBs for data sets or VSAM data spaces describe their charac-

[1] The 140 bytes are divided into a 44-byte key portion followed by a 96-byte data portion. You can refer to the logical 140-byte DSCB or to either of its parts.

© Copyright IBM Corp. 1979, 1999

teristics and location. DSCBs for contiguous, unassigned tracks indicate their location.

Data Set Control Block (DSCB) Types

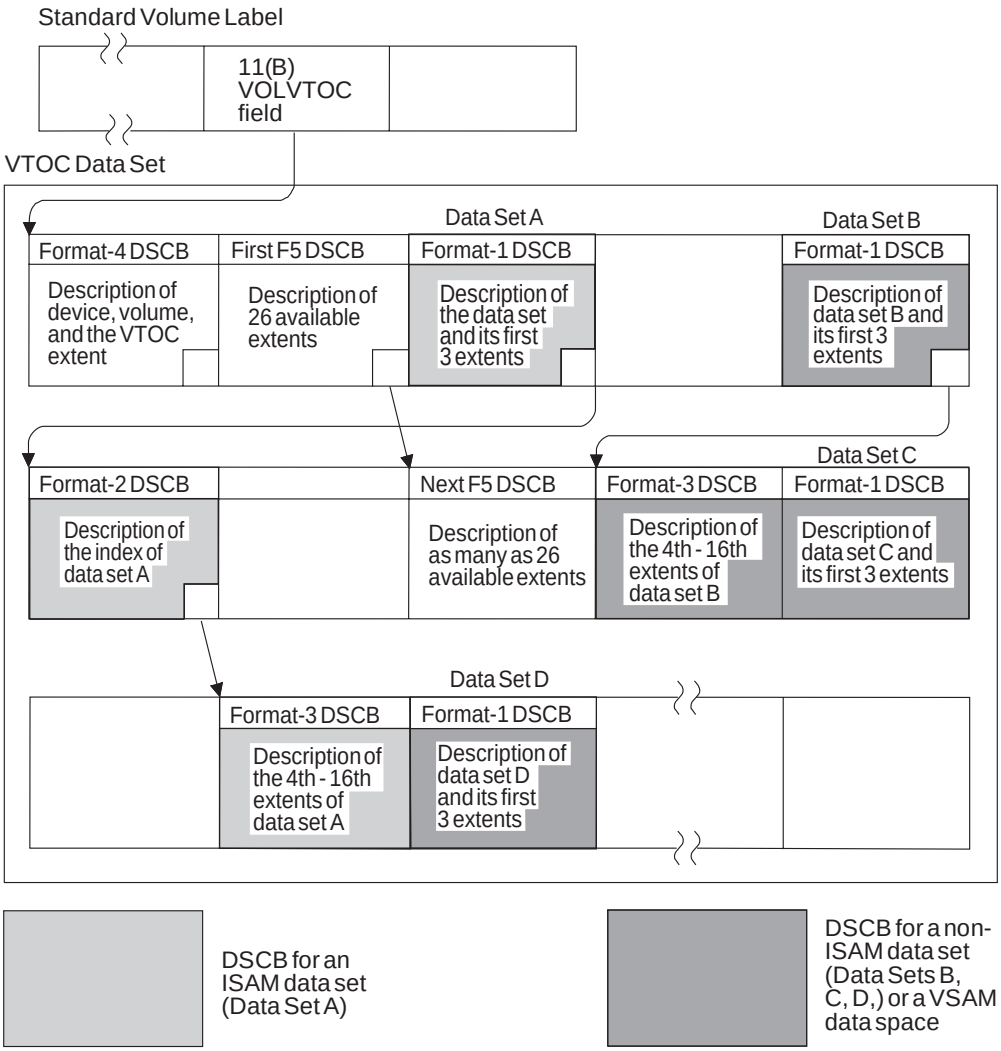
The VTOC contains several kinds of DSCBs. This section describes what the DSCBs are used for, how many exist on a volume, and how to locate them. The DSCB layouts are shown in Figure 3 on page 4 through Figure 7 on page 12.

The first record in every VTOC is the VTOC DSCB (format-4). The record describes the device the volume resides on, the volume attributes, and the size and contents of the VTOC data set. The next DSCB in the VTOC data set is a free-space DSCB (format-5) even if the free space is described by format-7 DSCBs. The third and subsequent DSCBs in the VTOC can occur in any order.

A data set or VSAM data space is defined by one or more DSCBs in the VTOC of each volume on which it resides. The number of DSCBs needed to define a data set or VSAM data space is determined by (1) the organization of the data set (for example, indexed sequential access method (ISAM) data sets need a format-2 DSCB to describe the index) and (2) the number of extents the data set or VSAM data space occupies. One or more format-3 DSCBs are required for data sets with more than three extents, including partitioned data sets extended (PDSEs).

Figure 2 on page 3 shows a VTOC and the DSCBs needed to define two types of data sets (ISAM and non-ISAM). Data set A (in Figure 2) is an ISAM data set; three DSCBs, a format-1, format-2, and format-3, are identified. Data sets B, C, and D could be sequential, partitioned, or direct data sets or they could be VSAM data spaces. Data set B has more than three extents and therefore requires both a format-1 and a format-3 DSCB.

Data sets C and D have three or fewer extents and need only a format-1 DSCB.



Note: Empty boxes in the VTOC data set represent free VTOC Records (Format-0 DSCBs)

Figure 2. Contents of VTOC—DSCBs Describing Data Sets

The mapping macro for the format-1, format-2, format-3, format-4, and format-5 DSCBs is IECSDSL1. Code positional parameters to specify which formats of DSCB to map. For example, the following maps format-1 and format-3:

```
IECSDSL1 1,3
```

The macro does not generate a DSECT statement. This makes it easier to embed it in your own DSECT or CSECT. You can use a technique such as the following to get separate DSECT statements:

```
F1AREA   DSECT
         IECSDSL1 1
F3AREA   DSECT
         IECSDSL1 3
```

The first symbol generated for each format is of the form IECSDSLn, where n is the number of the format.

Format-0 DSCB

Name: Free VTOC Record

Function: Describes an unused record in the VTOC (contains 140 bytes of binary zeros). To delete a DSCB from the VTOC, a format-0 DSCB is written over it.

How Many: One for every unused 140-byte record on the VTOC. The DS4DSREC field of the format-4 DSCB is a count of the number of format-0 DSCBs on the VTOC. This field is not maintained for an indexed VTOC.

How Found: Search on key equal to X'00' (sometimes X'00000000').

Format-1 DSCB

Name: Identifier

Function: Describes the first three extents of a data set or VSAM data space.

How Many: One for every data set or data space on the volume, except the VTOC.

How Found: Use the OBTAIN macro or a CVAF macro.

Figure 3 shows the contents of a format-1 DSCB.

Figure 3 (Page 1 of 5). DSCB Format-1

Offset Dec (Hex)	Type or Bit Mask	Length	Name	Description
0(X'0')	Character	44	DS1DSNAM	Data set name.
44(X'2C')	Character	1	DS1FMTID	Format Identifier (X'F1').
45(X'2D')	Character	6	DS1DSSN	Data set serial number (identifies the first or only volume containing the data set/space).
51(X'33')	Unsigned	2	DS1VOLSQ	Volume sequence number.
53(X'35')	Character	3	DS1CREDT	Creation date ('YDD'), discontinuous binary. Add 1900 and the value in the Y byte to determine the year. For VSAM data sets that are not SMS-managed, the expiration date is in the catalog.
56(X'38')	Character	3	DS1EXPDT	Expiration date ('YDD'), discontinuous binary. Add 1900 and the value in the Y byte to determine the year.
59(X'3B')	Unsigned	1	DS1NOEPV	Number of extents on volume.
60(X'3C')	Unsigned	1	DS1NOBDB	Number of bytes used in last directory block.
61(X'3D')	Bitstring	1	DS1FLAG1	Flags byte
	1		DS1COMPR	Compressible format data set (DS1STRP is also 1).
	. 1		DS1CPOIT	Checkpointed data set.
	. . xx xxxx			Reserved.
62(X'3E')	Character	13	DS1SYSCD	System code.
75(X'4B')	Character	3	DS1REFD [2]	Date last referenced ('YDD' or zero, if not maintained). Add 1900 and the value in the Y byte to determine the year.

[2] For VSAM data sets cataloged in an integrated catalog facility catalog, if the date in the FMT-1 DSCB's DS1REFD field is earlier than the current date, OPEN updates the field with the current date. For a multivolume VSAM data set, this is only done for the first volume of the data component of the base cluster. For a non-VSAM multi-volume data set that is not extended format, OPEN updates the DS1REFD fields on the first volume OPEN has selected to use. For an extended format non-VSAM data set, OPEN updates the DS1REFD fields on all volumes of the data set.

Figure 3 (Page 2 of 5). DSCB Format-1

Offset Dec (Hex)	Type or Bit Mask	Length	Name	Description
78(X'4E')	Bitstring	1	DS1SMSFG	System managed storage indicators.
	1		DS1SMSDS	System managed data set. IEHLIST displays this bit as the letter “S.”
	. 1		DS1SMSUC	Uncataloged system managed data set (the VTOC index is an uncataloged system managed data set as are all temporary data sets on system managed volumes). IEHLIST displays this bit as the letter “U.”
	. . 1		DS1REBLK	System determined the block size and data set can be reblocked (you or the system can reblock the data set). IEHLIST displays this bit as the letter “R.”
	. . . 1		DS1CRSDB	DADSM created original block size and data set has not been opened for output. IEHLIST displays this bit as the letter “B.”
	. . . . 1 . . .		DS1PDSE	Data set is a PDSE or HFS data set(DS1PDSEX is also 1 for HFS). IEHLIST displays this bit as the letter “I” for a PDSE. IEHLIST displays a “?” when it finds an invalid combination of bits.
	. . . . . 1 . .		DS1STRP	Sequential extended-format data set. IEHLIST displays this bit as the letter “E.” IEHLIST displays a “?” when it finds an invalid combination of bits.
	. . . . . 1 .		DS1PDSEX	HFS data set (DS1PDSE must also be 1) IEHLIST displays this bit as the letter “H.”
	 1		DS1DSAE	Extended attributes exist
79(X'4F')	Character	3	DS1SCEXT	Secondary space extension. Valid only when DS1EXT is on (see offset 94(X'5E')).
79(X'4F')	Bitstring	1	DS1SCXTF	Secondary space extension flag byte—only one of the first 4 bits is on.
	1		DS1SCAVB	If 1, DS1SCXTV is the original block length. If 0, DS1SCXTV is the average record length.
	. 1		DS1SCMB	If 1, DS1SCXTV is in megabytes.
	. . 1		DS1SCKB	If 1, DS1SCXTV is in kilobytes.
	. . . 1		DS1SCUB	If 1, DS1SCXTV is in bytes.
	. . . . 1 . . .		DS1SCCP1	If 1, DS1SCXTV has been compacted by a factor of 256.
	. . . . . 1 . .		DS1SCCP2	If 1, DS1SCXTV has been compacted by a factor of 65,536.
80(X'50')	Unsigned	2	DS1SCXTV	Secondary space extension value for average record length or average block length.
82(X'52')	Bitstring	2	DS1DSORG	Data set organization.
			First byte of DS1DSORG	
	1000 000x		DS1DSGIS	Indexed sequential organization.
	0100 000x		DS1DSGPS	Physical sequential organization.
	0010 000x		DS1DSGDA	Direct organization.
	0001 000x		DS1DSGCX	BTAM or QTAM line group.
	. . . . xx . .			Reserved.
	0000 001x		DS1DSGPO	Partitioned organization.
	 1		DS1DSGU	Unmovable; the data contains location dependent information.
			Second byte of DS1DSORG	
	100x 00xx		DS1DSGGS	Graphics organization.
	010x 00xx		DS1DSGTX	TCAM line group.
	001x 00xx		DS1DSGTQ	TCAM message queue.

Figure 3 (Page 3 of 5). DSCB Format-1

Offset Dec (Hex)	Type or Bit Mask	Length	Name	Description
	000x 10xx		DS1ACBM	VSAM data set/space.
	000x 10xx		DS1ORGAM	VSAM data set/space.
	000x 01xx		DS1DSGTR	TCAM 3705.
	. . . x . . xx			Reserved.
84(X'54')	Character	1	DS1RECFM	Record format.
	10		DS1RECFF	Fixed length.
	01		DS1RECFV	Variable length.
	11		DS1RECFU	Undefined length.
	. . 1		DS1RECFT	Track overflow.
	. . . 1		DS1RECFB	Blocked; cannot occur with undefined.
	. . . . 1 . . .		DS1RECFS	Fixed length: X'88' = standard blocks; no truncated blocks or unfilled tracks. Variable length: X'48' = spanned records.
	. . . . . 00 .			No control character.
	. . . . . 10 .		DS1RECFA	ISO/ANSI control character.
	. . . . . 01 .		DS1RECMC	Machine control character.
	. . . . . 11 .			Reserved.
	 x			Reserved.
85(X'55')	Character	1	DS1OPTCD	Option Code.
BDAM OPTCD field assignments (applies only if DS1DSGDA is on):				
	1			Write validity check.
	. 1			Track overflow.
	. . 1			Extended search.
	. . . 1			Feedback.
	. . . . 1 . . .			Actual addressing.
	. . . . . 1 . .			Dynamic buffering.
	. . . . . 1 .			Read exclusive.
	 1			Relative block addressing.
ISAM OPTCD field assignments (applies only if DS1DSGIS is on):				
	1			Write validity check.
	. 1			Accumulate track index entry.
	. . 1			Master indices.
	. . . 1			Independent overflow area.
	. . . . 1 . . .			Cylinder overflow area.
	. . . . . 1 . .			Reserved.
	. . . . . 1 .			Delete option.
	 1			Reorganization criteria.
BPAM, BSAM, QSAM OPTCD field assignments (applies only if DS1DSGPO or DS1DSGPS is on):				
	1			Write validity check.
	. 1			Allow data check (if on printer).
	. . 1			Chained scheduling.
	. . . 1			VSE/MVS interchange feature on tape.
	. . . . 1 . . .			Treat EOF as EOV (tape).
	. . . . . 1 . .			Search direct.
	. . . . . 1 .			User label totaling.
	 1			Each record contains a table reference character.
85(X'55')	Bitstring	1	DS1OPTAM	VSAM OPTCD settings.
VSAM OPTCD field assignments (applies only if DS1ORGAM is on):				
	1		DS1OPTIC	Data set is cataloged in an integrated catalog facility catalog.
	. 1		DS1OPTBC	Data set is an integrated catalog facility catalog.
	. . xx xxxx			Reserved.
86(X'56')	Binary	2	DS1BLKL	Block length (Type F records), or maximum block size (U or V records).

Figure 3 (Page 4 of 5). DSCB Format-1

Offset Dec (Hex)	Type or Bit Mask	Length	Name	Description
88(X'58')	Binary	2	DS1LRECL	Logical record length: Fixed length–record length, Undefined length–zero, Variable unspanned–maximum record length, Variable spanned and < 32757 bytes–maximum record length, Variable spanned and > 32756 bytes–X'8000'.
90(X'5A')	Binary	1	DS1KEYL	Key length (0 to 255).
91(X'5B')	Binary	2	DS1RKP	Relative key position.
93(X'5D')	Character	1	DS1DSIND	Data set indicators.
	1		DS1IND80	Last volume containing data in this data set.
	. 1		DS1IND40	Data set is RACF* defined with a discrete profile.
	. . 1		DS1IND20	Block length is a multiple of 8 bytes.
	. . . 1		DS1IND10	Password is required to read or write, or both; see DS1IND04.
	. . . 1 . x . .		DS1IND04	If DS1IND10 is 1 and DS1IND04 is 1, password required to write, but not to read. If DS1IND10 is 1 and DS1IND04 is 0, password required both to write and to read.
	. . . . x . . .		DS1IND08	Reserved.
	. . . . . 1 .		DS1IND02	Data set opened for other than input since last backup copy made.
			DS1DSCHA	Same as DS1IND02.
	 1		DS1IND01	Secure checkpoint data set.
			DS1CHKPT	Same as DS1IND01.
94(X'5E')	Binary	4	DS1SCALO	Secondary allocation space parameters.
94(X'5E')	Character	1	DS1SCAL1	Flag byte.
	xxxx xxxx		DS1DSPAC	Space request bits, defined as follows:
	00 . 0		DS1DSABS	Absolute track request.
	00 . 1		DS1EXT	Extension to secondary space exists (see DS1SCEXT at offset 79 (X'4F')).
	11		DS1CYL	Cylinder request.
	10		DS1TRK	Track request.
	01 x		DS1AVR	Average block length request.
	01 1		DS1AVRND	Average block and round request.
	. . 1		DS1MSGP	Mass storage vol group (MSVGP - no longer supported).
	. . . . 1 . . .		DS1CONTG	Contiguous request.
	. . . . . 1 . .		DS1MXIG	MXIG request.
	. . . . . 1 .		DS1ALX	ALX request.
	 1		-	Round request.
95(X'5F')	Binary	3	DS1SCAL3	Secondary allocation quantity.
98(X'62')	Binary	3	DS1LSTAR	Last used track and block on track (TTR). Not defined for VSAM, PDSE, and HFS.
101(X'65')	Binary	2	DS1TRBAL	If not extended format, this is the value from TRKCALC indicating space remaining on last track used. For extended format data sets this is the high order two bytes (TT) of the four-byte last used track number. See DS1LSTAR. Zero for VSAM, PDSE and HFS.
103(X'67')	Character	2		Reserved.
105(X'69')	Character	30	DS1EXNTS	Three extent fields.
105(X'69')	Character	10	DS1EXT1	First extent description.
	Character	1		Extent type indicator.
	X'81'			Extent on cylinder boundaries.

Figure 3 (Page 5 of 5). DSCB Format-1

Offset Dec (Hex)	Type or Bit Mask	Length	Name	Description
	X'80'			Extent described is sharing cylinder (no longer supported).
	X'40'			First extent describes the user labels and is not counted in DS1NOEPV.
	X'04'			Index area extent (ISAM).
	X'02'			Overflow area extent (ISAM).
	X'01'			User's data block extent, or a prime area extent (ISAM).
	X'00'			This is not an extent.
		1		Extent sequence number.
		4		Lower limit (CCHH).
		4		Upper limit (CCHH).
115(X'73')	Character	10	DS1EXT2	Second extent description.
125(X'7D')	Character	10	DS1EXT3	Third extent description.
135(X'87')	Character	5	DS1PTRDS	Pointer (CCHHR) to a format-2 or format-3 DSCB, or zero.
140(X'8C')	Character		DS1END	-

Format-2 DSCB

Name: Index

Function: Describes the indexes of an ISAM data set.

How Many: One for each ISAM data set (for a multivolume ISAM data set, a format-2 DSCB exists only on the first volume).

How Found: Chained from a format-1 DSCB that represents the data set.

Figure 4 shows the contents of a format-2 DSCB.

Figure 4 (Page 1 of 2). DSCB Format-2

Offset Dec (Hex)	Type	Length	Name	Description
0(X'00')	Character	1	-	Key identifier (X'02').
1(X'01')	Bitstring	7	DS22MIND	Address of 2nd level master index.
8(X'08')	Bitstring	5	DS2L2MEN	Last 2nd level master index entry.
13(X'0D')	Bitstring	7	DS23MIND	Address of 3rd level master index.
20(X'14')	Bitstring	5	DS2L3MIN	Last 3rd level master index entry.
25(X'19')	Bitstring	11		Reserved.
36(X'24')	Bitstring	8	DS2LPDT	Last prime track on last prime cylinder.
44(X'2C')	Character	1	DS2FMTID	Format identifier (X'F2').
45(X'2D')	Unsigned	1	DS2NOLEV	Number of index levels.
46(X'2E')	Bitstring	1	DS2DVIND	High-level index development indicator.
47(X'2F')	Bitstring	3	DS21RCYL	First data record in cylinder.
50(X'32')	Bitstring	2	DS2LTCYL	Last data track in cylinder.
52(X'34')	Unsigned	1	DS2CYLOV	Number of tracks for cylinder overflow.
53(X'35')	Unsigned	1	DS2HIRIN	Highest 'R' on high-level index track.
54(X'36')	Unsigned	1	DS2HIRPR	Highest 'R' on prime data track.
55(X'37')	Unsigned	1	DS2HIROV	Highest 'R' on overflow data track.
56(X'38')	Unsigned	1	DS2RSHTR	'R' of last data record on shared track.
57(X'39')	Unsigned	1	DS2HIRTI	Highest 'R' on unshared track of track index.
58(X'3A')	Unsigned	1	DS2HIIOV	Highest 'R' for independent overflow data tracks.

Figure 4 (Page 2 of 2). DSCB Format-2

Offset Dec (Hex)	Type	Length	Name	Description
59(X'3B')	Unsigned	2	DS2TAGDT	Tag deletion count.
61(X'3D')	Unsigned	3	DS2RORG3	Non-first overflow reference count.
64(X'40')	Unsigned	2	DS2NOBYT	Number of bytes for highest-level index.
66(X'42')	Unsigned	1	DS2NOTRK	Number of tracks for highest level index.
67(X'43')	Unsigned	4	DS2PRCTR	Prime record count.
71(X'47')	Bitstring	1	DS2STIND	Status indicators.
	. 1			Perform key sequence checking.
	. . 1			Initial load has been completed.
	. . . . . 1 .			Last block full.
	 1			Last track full.
	x . . x xx . .			Reserved.
72(X'48')	Bitstring	7	DS2CYLAD	Address of cylinder index.
79(X'4F')	Bitstring	7	DS2ADLIN	Address of lowest level master index.
86(X'56')	Bitstring	7	DS2ADHIN	Address of highest level master index.
93(X'5D')	Bitstring	8	DS2LPRAD	Address of last prime data record.
101(X'65')	Bitstring	5	DS2LTRAD	Address of last track index entry.
106(X'6A')	Bitstring	5	DS2LCYAD	Address of last cylinder index entry.
111(X'6F')	Bitstring	5	DS2LMSAD	Address of last master index entry.
116(X'74')	Bitstring	8	DS2LOVAD	Address of last independent overflow record.
124(X'7C')	Unsigned	2	DS2BYOVL	Bytes remaining on overflow track.
126(X'7E')	Unsigned	2	DS2RORG2	Tracks remaining in independent overflow area.
128(X'80')	Unsigned	2	DS2OVRCT	Overflow record count.
130(X'82')	Unsigned	2	DS2RORG1	Cylinder overflow area count.
132(X'84')	Bitstring	3	DS2NIRT	Address of dummy track index entry.
135(X'87')	Bitstring	5	DS2PTRDS	Pointer (CCHHR) to a format-3 DSCB, or zero.
140(X'8C')			DS2END	-

Format-3 DSCB

Name: Extension

Function: Describes extents after the third extent of a non-VSAM data set or a VSAM data space.

How Many: One for each data set or VSAM data space on the volume that has more than three extents. There can be as many as ten for a PDSE, HFS, extended format data set, or a VSAM data set cataloged in an integrated catalog facility catalog. PDSEs, HFS, and extended format data sets can have up to 123 extents. Each component of a VSAM data set cataloged in an integrated catalog facility catalog can have up to 123 extents per volume. All other data sets and VSAM data spaces are restricted to 16 extents per volume.

How Found: Chained from a format-2 or a format-1 DSCB that represents the data set or VSAM data space. In the case of a PDSE, a sequential extended-format data set, or a VSAM data set cataloged in an integrated catalog facility catalog, the chain can be from a preceding format-3 DSCB.

Figure 5 shows the contents of a format-3 DSCB.

Figure 5. DSCB Format-3

Offset Dec (Hex)	Type	Length	Name	Description
0(X'00')	Bitstring	4	-	Key identifier (X'03030303').
4(X'04')	Bitstring	40	DS3EXTNT	Four extent descriptions.
		1		Extent type indicator.
		1		Extent sequence number.
		4		Lower limit (CCHH).
		4		Upper limit (CCHH).
44(X'2C')	Character	1	DS3FMTID	Format identifier (X'F3').
45(X'2D')	Bitstring	90	DS3ADEXT	Nine additional extent descriptions.
135(X'87')	Bitstring	5	DS3PTRDS	Pointer (CCHHR) to next format-3 DSCB, or zero.
140(X'8C')			DS3END	-

Format-4 DSCB

Name: VTOC

Function: Describes the extent and contents of the VTOC and provides volume and device characteristics. This DSCB contains a flag indicating whether the volume is SMS managed.

How Many: One on each volume.

How Found: The VOLVTOC field of the standard volume label contains the format-4 address. The format-4 DSCB is always the first record in the VTOC.

Figure 6 shows the contents of a format-4 DSCB.

Note: The format-4 DSCB has a 44-byte key of X'04' bytes not shown in Figure 6.

Figure 6 (Page 1 of 3). DSCB Format-4

Offset Dec (Hex)	Type	Length	Name	Description
0(X'00')	Character	1	DS4IDFMT	Format identifier (X'F4').
1(X'01')	Character	5	DS4HPCHR	Highest address (CCHHR) of a format-1 DSCB.
6(X'06')	Unsigned	2	DS4DSREC	Number of available DSCBs.
8(X'08')	Character	4	DS4HCCHH	CCHH of next available alternate track.
12(X'0C')	Unsigned	2	DS4NOATK	Number of remaining alternate tracks.
14(X'0E')	Bitstring	1	DS4VTOCI	VTOC indicators.
	1		DS4DOSBT	VSE bit. Either invalid format 5 DSCBs or indexed VTOC. Previously DOS(VSE) bit. See DS4IVTOC.
	. 1		DS4DVTOC	Index was disabled.
	. . 1		DS4EFVLD	Extended free-space management flag. When DS4EFVLD is on, the volume is in OSVTOC format with valid free space information in the format-7 DSCBs. See also DS4EFLVL and DS4EFPTR.
	. . . 1		DS4DSTKP	VSE stacked pack.
	. . . . 1 . . .		DS4DOCVT	VSE converted VTOC.
	. . . . . 1 . .		DS4DIRF	DIRF bit. A VTOC change is incomplete.
	. . . . . 1 .		DS4DICVT	DIRF reclaimed.
	 1		DS4IVTOC	Volume uses an indexed VTOC.
15(X'0F')	Unsigned	1	DS4NOEXT	Number of extents in the VTOC(X'01').
16(X'10')	Bitstring	1	DS4SMSFG	System managed storage indicators.
	00		DS4NTSMS	Non-system managed volume.

Figure 6 (Page 2 of 3). DSCB Format-4

Offset Dec (Hex)	Type	Length	Name	Description
	01		DS4SMSCV	System managed volume in initial status.
	10			Reserved.
	11		DS4SMS	System managed volume.
	. . xx xxxx			Reserved.
17(X'11')	Binary	1	DS4DEVAC	Number of alternate cylinders when the volume was formatted. Subtract from first 2 bytes of DS4DEVSZ to get number of useable cylinders (can be 0). Valid only if DS4DEVAV is on.
18(X'12')	Character	14	DS4DEVCT	Device constants. For currently supported devices, these fields do not provide enough information to calculate the amount of space used on a track. Use TRKCALC (see “Performing Track Calculations” on page 198).
18(X'12')	Character	4	DS4DEVSZ	Device size.
18(X'12')	Binary	2	DS4DSCYL	Number of logical cylinders including alternates, if any.
20(X'14')	Binary	2	DS4DSTRK	Number of tracks in a logical cylinder.
22(X'16')	Unsigned	2	DS4DEVTK	Device track length.
24(X'18')	Binary	2	DS4DEVOV	Keyed record overhead.
24(X'18')	Binary	1	DS4DEVI	Non-last-keyed record overhead.
25(X'19')	Binary	1	DS4DEVL	Last keyed record overhead.
26(X'1A')	Binary	1	DS4DEVK	Non-keyed record overhead differential.
27(X'1B')	Bitstring	1	DS4DEVFG	Flag byte.
	. . . 1		DS4DEVAV	Value in DS4DEVAC, number of alternate cylinders is valid. Otherwise, the number is not available in this DSCB.
	. . . . 1 . . .			Keyed record overhead field (DS4DEVOV) is used as a 2-byte field to specify the overhead required for a keyed record.
	. . . . . 1 . .			The CCHH of an absolute address is used as a continuous binary value.
	. . . . . 1 .			The CCHH of an absolute address is used as four separate binary values.
	 1			A tolerance factor must be applied to all but the last block of the track.
	xxxx			Reserved.
28(X'1C')	Binary	2	DS4DEVTL	Device tolerance.
30(X'1E')	Binary	1	DS4DEVDT	Number of DSCBs per track.
31(X'1F')	Binary	1	DS4DEVDB	Number of PDS directory blocks per track.
32(X'20')	Binary	8	DS4AMTIM	VSAM time stamp.
40(X'28')	Character	3	DS4AMCAT	VSAM catalog indicator.
40(X'28')	Bitstring	1	DS4VSIND	VSAM indicators.
	1		DS4VSREF	A VSAM catalog references this volume.
	. 1		DS4VSBAD	The VSAM data sets on this volume are unusable because an MSS CONVERTV command has not completed successfully for the volume. (No longer set.)
	. . xx xxxx			Reserved.
41(X'29')	Unsigned	2	DS4VSCRA	Relative track location of the CRA.
43(X'2B')	Binary	8	DS4R2TIM	VSAM volume/catalog match time stamp.
51(X'33')	Character	5		Reserved.
56(X'38')	Character	5	DS4F6PTR	Pointer (CCHHR) to first format-6 DSCB, or zero. (No longer supported as non-zero).
61(X'3D')	Character	10	DS4VTOCE	VTOC extent description.
71(X'47')	Character	10	Reserved.	VTOC extent description.

Figure 6 (Page 3 of 3). DSCB Format-4

Offset Dec (Hex)	Type	Length	Name	Description
81(X'51')	Character	1	DS4EFLVL	Extended free-space management level. X'00' indicates extended free-space management is not used for this volume. X'07' indicates extended free-space management is in use for this volume (see also DS4EFVLD).
82(X'52')	Character	5	DS4EFPTR	Pointer to extended free-space information. If DS4EFLVL=X'00' this is zero. If DS4EFLVL=X'07' this is the CCHHR of the first FMT-7 DSCB and no format-5 DSCBs contain free space information.
87(X'57')	Character	9		Reserved.
96(X'60')			DS4END	-

Format-5 DSCB

Name: Free Space

Function: On a nonindexed VTOC, describes the space on a volume that has not been allocated to a data set or to a VSAM data space (available space). For an indexed VTOC, a single empty format-5 DSCB resides in the VTOC; free space is described in the index and DS4IVTOC is normally on.

How Many: One for every 26 noncontiguous extents of available space on the volume for a nonindexed VTOC; for an indexed VTOC, there is only one.

How Found: The first format-5 DSCB on the volume is always the second DSCB of the VTOC. If there is more than one format-5 DSCB, it is chained from the previous format-5 DSCB using the DS5PTRDS field.

Figure 7 shows the contents of a format-5 DSCB.

Figure 7. DSCB Format-5

Offset Dec (Hex)	Type	Length	Name	Description
0(X'00')	Bitstring	4	DS5KEYID	Key identifier (X'05050505').
4(X'04')	Bitstring	5	DS5AVEXT	Available extent.
		2		Relative track address of the first track in the extent. Relative to the beginning of the volume.
		2		Number of unused cylinders in the extent.
		1		Number of additional unused tracks.
9(X'09')	Bitstring	35	DS5EXTAV	Seven available extents.
44(X'2C')	Character	1	DS5FMTID	Format identifier (X'F5').
45(X'2D')	Bitstring	90	DS5MAVET	Eighteen available extents.
135(X'87')	Bitstring	5	DS5PTRDS	Pointer (CCHHR) to next format-5 DSCB, or zero.
140(X'8C')			DS5END	-

Format-7 DSCB

Name: Free space for certain devices

Only one field in the format-7 DSCB is an intended interface. This field indicates whether the DSCB is a format-7 DSCB. You can reference that field as DS1FMTID or DS5FMTID. A character 7 indicates that the DSCB is a format-7 DSCB, and your program should not modify it.

If you are diagnosing a problem, see *DFSMS/MVS DFSMSdfp Diagnosis Reference* for the layout of the Format-7 DSCB.

Allocating and Releasing DASD Space

DADSM allocate and extend routines assign tracks and cylinders on direct access volumes for new data sets and VSAM data spaces. The DADSM extend routine gets additional space for a data set or VSAM data space that has exceeded its primary allocation. The DADSM scratch and partial release routines release space that is no longer needed on a direct access volume.

The DADSM routines allocate and release space by adding, deleting, and modifying the DSCBs and updating the VTOC index. When space is released, the scratch routines free the DSCBs of the deleted data set or data space.

The VTOC Index

The VTOC index enhances the performance of VTOC access. The VTOC index is a physical-sequential data set on the same volume as the related VTOC. It consists of an index of data set names in format-1 DSCBs contained in the VTOC and volume free space information.

Note: An SMS-managed volume *requires* an indexed VTOC; otherwise, the VTOC index is highly recommended. For additional information about SMS-managed volumes, see *DFSMS/MVS Implementing System-Managed Storage*.

In MVS/DFP 3.3 the VTOC index support was improved to make more efficient use of space in an index. An index that was in use prior to that release might have wasted space. It might run out of space unnecessarily. If that is a concern, you can rebuild the index to ensure that it is organized more efficiently. Use ICKDSF. For information about creating a VTOC index, see “Creating the VTOC and VTOC Index” on page 57.

MVS does support sharing a non-SMS-managed volume that contains a VTOC index with a non-MVS system. If the other system updates the VTOC and turns on the DS4DOSBT, then later when MVS is used to modify the VTOC, DADSM can detect that the index is no longer valid.

Device Support Facilities (ICKDSF) initializes a VTOC index into 2048-byte physical blocks named VTOC index records (VIRs). VIRs are used in several ways. A VTOC index contains the following kinds of VIRs:

VTOC index entry record (VIER) identifies the location of format-1 DSCBs and the format-4 DSCB.

VTOC pack space map (VPSM) identifies the free and allocated space on a volume.

VTOC index map (VIXM) identifies the VIRs that have been allocated in the VTOC index.
VTOC map of DSCBs (VMDS) identifies the DSCBs that have been allocated in the VTOC.

A format-1 DSCB in the VTOC contains the name and extent information of the VTOC index. The name of the index must be `'SYS1.VTOCIX.xxxxxxxx'`, where `'xxxxxxxx'` conforms to standard data set naming conventions and is usually the serial number of the volume containing the VTOC and its index. The name must be unique within the system to avoid ENQ contention.

You can only create (allocate) one data set whose name begins with `'SYS1.VTOCIX.'` on a volume. To rename a VTOC index data set when the VTOC index is active, use a name beginning with `'SYS1.VTOCIX.'`. If a `'SYS1.VTOCIX.'` data set already exists on a volume, you cannot rename another data set on the volume to a name with those qualifiers. If the VTOC index is active, you cannot scratch the VTOC index data set.

The relationship of a VTOC to its index is shown in Figure 8.

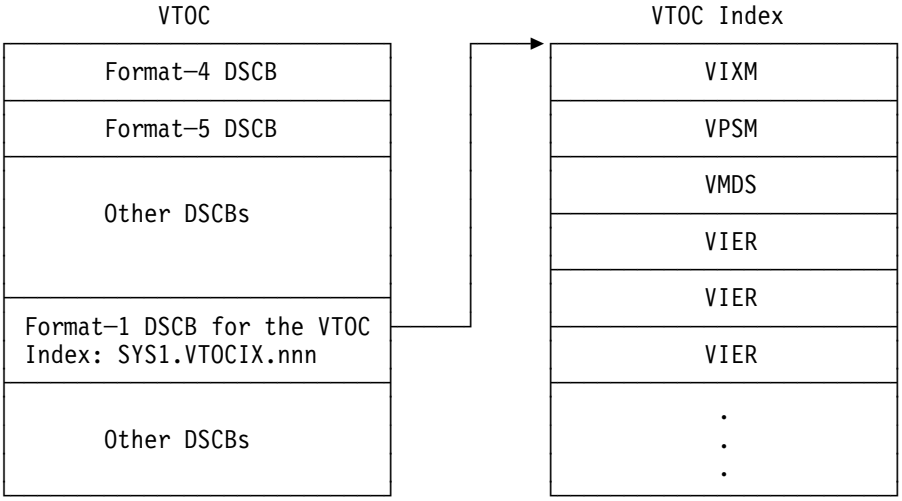


Figure 8. Example of the Relationship of a VTOC to Its Index

VTOC Index Records

VTOC index records consist of the following types:

VTOC Index Entry Record

The first level-one VIER is created with the VTOC index. Subsequent VIERs are created whenever a VIER is too full to allow a data set name to be added to the VTOC index. VIERs have the following characteristics:

- A VIER uses one VIR and contains variable-length index entries. The number of VIERs in an index depends upon the number of data sets on the volume.
- All index entries within a VIER are at the same index level. VIERs in a VTOC index can be on several levels and have a hierarchic relationship. Index entries in higher-level VIERs point to lower-level VIERs. Index entries in level-one

VIERs (those at the lowest level) point to format-1 DSCBs for data sets on the volume.

- Whenever a fourth VIER is created on the same level, a VIER at a higher level is created. Once a higher-level VIER is filled with pointers to lower-level VIERs, another VIER at the same level is created.

VTOC Pack Space Map

The VPSM shows the allocated space on a volume and the space that remains free. The map contains bit maps of the cylinders and tracks on the volume. A value of one indicates that the cylinder or track has been allocated; zero, that it is unallocated. Tracks assigned as alternate tracks are marked as allocated.

VTOC Index Map

The VIXM contains a bit map in which each bit represents one VTOC index record. The status of the bit indicates whether the VIR is allocated (1), or unallocated (0).

An area of the VIXM is reserved for VTOC recording facility (VRF) data. (This facility allows detection of and recovery from some errors in an indexed VTOC.)

Structure of an Indexed VTOC

Indexed and nonindexed VTOCs have similar structures with the following differences for an indexed VTOC:

- Only a single, empty format-5 DSCB exists.
- Some format-4 DSCB data (the number of format-0 DSCBs and the CCHHR of the highest format-1 DSCB) is not maintained by DADSM. The VSE bit (bit 0 in field DS4VTOCI), set to 1 in the format-4 DSCB, indicates that the contents of these fields (and the format-5 DSCB) are not valid. The index bit (bit 7 in field DS4VTOCI) is set in the format-4 DSCB; it indicates that a VTOC index exists.

Accessing the VTOC with DADSM Macros

You can use either DADSM or common VTOC access facility (CVAF) macros to access a VTOC and its index. (CVAF access is described in “Accessing the VTOC with CVAF Macros” on page 40.) The DADSM macros and tasks discussed in this chapter consist of the following:

LSPACE provides free space, volume fragmentation, and VTOC status information for a DASD volume. LSPACE returns the information to any of four user-specified areas.
OBTAIN reads a DSCB from the VTOC.
PARTREL releases unused space from a sequential or partitioned data set or a PDSE.
REALLOC allocates DASD space.

To read a DSCB into virtual storage use the OBTAIN and CAMLST macro instructions. Identify the DSCB to be read using the name of the data set associated with the DSCB, or the absolute track address of the DSCB. Provide a 140-byte data area in virtual storage to contain the DSCB. When you specify the name of the data set, an identifier (format-1 or format-4) DSCB is read into virtual storage. To read a DSCB other than a format-1 or a format-4 DSCB, specify an absolute track address (see the example on page “Example” on page 26). (see the example on topic “Example” on page 26).

Note: You cannot use the OBTAIN macro instruction with either a SYSIN or SYSOUT data set.

To release unused space from a sequential, partitioned, or key sequenced data set or a PDSE, use the PARTREL macro instruction. Your program must be APF authorized.

The following macro instruction descriptions include coding examples, programming notes, and exception return code descriptions.

Requesting DASD Volume Information Using LSPACE

You can use the LSPACE macro to get free space, volume fragmentation, and VTOC status information for a DASD volume. LSPACE normally returns status information (such as LSPACE subfunction, return code, and reason code) to the parameter list. The return code is also returned in register 15. The format of the LSPACE parameter list is shown in Figure 9 on page 20. You can have LSPACE return additional information such as the total number of free extents on the volume, or the fragmentation index. This additional information can be returned in the:

Message return area: described on page 22
Expanded message return area: described on page 22
Data return area: described on page 23
Format-4 DSCB return area: described on page 23.

Message return area: described on 22
Expanded message return area: described on 22
Data return area: described on 23
Format-4 DSCB return area: described on 23.

The calling program must ensure that the volume to be processed remains mounted during LSPACE processing. The volume need not be allocated.

LSPACE—Standard Form

The format of the standard form of the LSPACE macro is:

```
►►──┬──────────┬──LSPACE──┬──────────────┬──UCB=──┬─addr──┬──────────────────────►
    └─label────┘          └─MF──=──I─────┘        └─(reg)─┘

►──┬─,MSG=──┬─addr──┬──┬──────────────┬──┬──────────────────────────────────────►
   │        ├─(reg)─┤  │              │  └─,SMF=──┬─TEST──┬──┘
   │        └─0─────┘  │              │           ├─YES───┤
   ├─,EXPMSG=──┬─addr──┬──┤              │           └─NONE──┘
   │           ├─(reg)─┤  │              │
   │           └─0─────┘  │              │
   └─,DATA=──┬─addr──┬────┘              │
             ├─(reg)─┤                   │
             └─0─────┘                   │

►──┬──────────────────────────────────────────────────────────────────────────►◄
   └─,F4DSCB=──┬─addr──┬──┘
               ├─(reg)─┤
               └─0─────┘
```

MF=I
specifies the standard form of the LSPACE macro.

I specifies the standard form of the macro. This generates a standard parameter list containing the required variables, loads the address of the parameter list in register 1, and issues a supervisor call. The standard form is the default.

Note: UCB must be specified when MF=I is used.

LSPACE—Execute Form

The format of the execute form of the LSPACE macro is:

```
►►──┬──────────┬──LSPACE──MF=──┬─(E,addr)──┬──┬──────────────────────────►
    └─label────┘               └─(E,(reg))─┘  └─,UCB=──┬─addr──┬──┘
                                                        └─(reg)─┘

►──┬──────────────────────────┬──┬──────────────────────────────────────►
   ├─,MSG=──┬─addr──┬──────────┤  └─,SMF=──┬─TEST──┬──┘
   │        ├─(reg)─┤          │           ├─YES───┤
   │        └─0─────┘          │           └─NONE──┘
   ├─,EXPMSG=──┬─addr──┬────────┤
   │           ├─(reg)─┤        │
   │           └─0─────┘        │
   └─,DATA=──┬─addr──┬──────────┘
             ├─(reg)─┤
             └─0─────┘

►──┬──────────────────────────┬──────────────────────────────────────►◄
   └─,F4DSCB=──┬─addr──┬────────┘
               ├─(reg)─┤
               └─0─────┘
```

MF=(E,*addr*) or **(E,(*reg*))**
specifies the execute form of the LSPACE macro.

The MF=L form is usually issued before the execute form to create the parameter list.

(E,*addr*)
loads the address of the parameter list specified by *addr* into register 1 and then issues a supervisor call.

(E,(*reg*))
loads the address of the parameter list specified by *(reg)* into register 1 and then issues a supervisor call.

UCB=*addr* or **(*reg*)**
specifies the address of the UCB for the volume whose free space information you are requesting. The UCB address can be for a captured UCB, or for an actual UCB above or below the 16MB line. For 31-bit callers, the high-order byte is part of the UCB address and must be cleared to zeros if a 24-bit UCB address is being passed.

addr—RX-type address
specifies the address of a fullword containing the UCB address.

(*reg***)**—(2-12)
specifies a register containing the UCB address for the device. Note that this differs from the *addr* form of the parameter.

When using the standard (MF=I) form of the macro, you must provide a UCB address.

Notes:

1. The LSPACE macro will accept the address of a UCB or UCB copy. Unauthorized programs can get a copy of the UCB by using the UCBSCAN macro and specifying the COPY and UCBAREA keywords. The UCB copy can be above or below the 16MB line and on a word boundary. Refer to *OS/390 HCD Planning* for details.
2. LSPACE does not support VIO UCBs.

MSG=*addror (reg)or***0** or **EXPMSG=***addr or (reg) or* **0** or **DATA=** *addror (reg***)** or **0**
specifies the way LSPACE is to return free space information. (optional)

Note: The MSG, EXPMSG, and DATA parameters are mutually exclusive.

MSG=*addr* or **(***2-12***)** or **0**
specifies the address of a caller-provided 30-byte message return area into which LSPACE returns either a free space message or, for unsuccessful requests, status information. For a description of this area, see “Message Return Area” on page 22.

addr—RX-type address
specifies the address of the message return area.

(*reg***)**—(2-12)
specifies a register containing the address of the message return area.

0 specifies that you do not want the free space message. This is the default for all forms of the macro except execute.

EXPMSG=*addr* or **(***reg***)** or **0**
specifies the address of a caller-provided 40-byte expanded message return area into which LSPACE returns either a free space message or, for unsuccessful requests, status information. For a description of this area, see “Expanded Message Return Area” on page 22.

addr—RX-type address
specifies the address of the message return area.

(*reg***)**—(2-12)
specifies a register containing the address of the message return area.

0 specifies that you do not want the free space message. This is the default for all forms of the macro except execute.

DATA=*addr* or **(***reg***)** or **0**
specifies the address of a caller-provided data return area into which LSPACE returns free space and volume information. For a description of this area, see “Data Return Area” on page 23.

addr—RX-type address
specifies the address of the data return area.

(*reg***)**—(2-12)
specifies a register containing the address of the data return area.

0 specifies that you do not want the free space and volume information.

SMF=TEST or **YES** or **NONE**
specifies the type of SMF processing.

TEST
specifies that LSPACE is to test for an active SMF system and whether SMF volume information is desired. If these conditions are met, an SMF record is written. *Only programs executing in supervisor state, protect key 0-7, or APF authorized can specify this operand*.

YES
specifies that you want an SMF record containing volume information to be written. *Only programs executing in supervisor state, protect key 0-7, or APF authorized can specify this operand*.

NONE
specifies that you do not want an SMF record containing volume information to be written. This is the default for all forms of the macro except execute.

F4DSCB=*addr* or **(***reg***)** or **0**
specifies the address of a 96-byte DSCB return area provided by the calling program, into which LSPACE returns the volume's format-4 DSCB. For a description of the format-4 DSCB fields, see Figure 6 on page 10.

addr—RX-type address
specifies the address of the format-4 DSCB return area.

(*reg***)**—(2-12)
specifies a register containing the address of the format-4 DSCB return area.

0 specifies that you do not want the data portion of the format-4 DSCB for the volume. This is the default for all forms of the macro except execute.

LSPACE—List Form

The format of the list form of the LSPACE macro is:

```
►►──┬──────────┬──LSPACE──MF=──┬─L──────────┬──┬──────────────────────────────►
    └─label────┘               ├─(L,MSG)────┤  ├─,MSG=──┬─addr─┬──────────┤
                               ├─(L,EXPMSG)─┤  │         └─0────┘          │
                               └─(L,DATA)───┘  ├─,EXPMSG=──┬─addr─┬────────┤
                                               │           └─0────┘        │
                                               └─,DATA=────┬─addr─┬────────┘
                                                           └─0────┘

►──┬──────────────────┬──┬──────────────────┬──►◄
   └─,SMF=──┬─TEST─┬──┘  └─,F4DSCB=──┬─addr─┬──┘
            ├─YES──┤                 └─0────┘
            └─NONE─┘
```

MF=Lor (L,MSG)or (L,EXPMSG)or (L,DATA)
specifies the list form of the LSPACE macro.

L generates the required constants in the calling program. You can then issue the execute form of the macro, which uses these constants.

(L,MSG)
generates the required message return area constants in the calling program. No other parameters are allowed.

(L,EXPMSG)
generates the required expanded message return area constants in the calling program. No other parameters are allowed.

(L,DATA)
generates the required data return area constants in the calling program. No other parameters are allowed.

LSPACE—DSECT Form

The format of the DSECT form of the LSPACE macro is:

```
►►──┬──────────┬──LSPACE──MF=──┬─D──────────────────────────────────►◄
    └─label────┘               ├─(D,MSG)──────┤
                               ├─(D,EXPMSG)────┤
                               └─(D,DATA)──────┘
```

MF=Dor (D,MSG)or (D,EXPMSG)or (D,DATA)
specifies the DSECT form of the LSPACE macro.

D generates a DSECT that maps the LSPACE parameter list. No other parameters are allowed. See Figure 9 for the format of the LSPACE parameter list.

(D,MSG)
generates a DSECT that maps the message return area. No other parameters are allowed. For the format of the area, see “Message Return Area” on page 22.

(D,EXPMSG)
generates a DSECT that maps the expanded message return area. No other parameters are allowed. For the format of the area, see “Expanded Message Return Area” on page 22.

(D,DATA)
generates a DSECT that maps the data return area. No other parameters are allowed. For the format of the area, see “Data Return Area” on page 23.

Figure 9 (Page 1 of 2). Format of the LSPACE Parameter List (MF=D)

Name	Offset	Bytes	Description
LSPAPL			
LSPAPLID	0(X'00')	4	EBCDIC 'LSPA'.
LSPANGTH	4(X'04')	2	Length of parameter list.
LSPAFLAG	6(X'06')	1	Parameter flag byte.

Figure 9 (Page 2 of 2). Format of the LSPACE Parameter List (MF=D)

Name	Offset	Bytes	Description
LSPASMFY		10	SMF=YES.
LSPASMFT		01	SMF=TEST.
LSPADATA		. . 1 0 0 . . .	Free space data request.
LSPAMSG		. . 0 1 0 . . .	Message data returned.
LSPAEMSG		. . 0 0 1 . . .	Expanded message data requested.
LSPARSVB		 xxx	Reserved.
LSPARSVD	7(X'07')	1	Reserved.
LSPAERCD	8(X'08')	1	LSPACE return code. See Figure 10 on page 22.
LSPASFID	9(X'09')	1	LSPACE subfunction code.
LSPASFPC		X'00'	Processing complete.
LSPASFVP		X'01'	Validate parameters.
LSPASFUS		X'02'	Check UCB status.
LSPASFNQ		X'03'	ENQ on SYSZDMNT.
LSPASF45		X'04'	Read F4 and first F5 DSCB (EXCP).
LSPASFN5		X'05'	Read next F5 DSCB (EXCP).
LSPASFRV		X'06'	Read volume label (EXCP).
LSPASF4X		X'80'	Read F4 and maps (CVAFDIR).
LSPASFEX		X'81'	Get free extents (CVAFDSM).
LSPASFF0		X'82'	Get F0 count (CVAFDSM).
LSPASFVR		X'83'	Get VIR count (CVAFDSM).
LSPASFVD		X'84'	Check for VRF (CVAFVRF).
LSPASFRT	10(X'0A')	1	Subfunction return code.
LSPASFRS	11(X'0B')	1	Subfunction reason code.
LSPARS01		X'01'	Invalid parameter list storage key.
LSPARS02		X'02'	Invalid parameter list ID.
LSPARS03		X'03'	Invalid LSPACE flag.
LSPARS04		X'04'	Invalid authorization for SMF flag.
LSPARS05		X'05'	Invalid message or data return area storage key.
LSPARS06		X'06'	Invalid format-4 DSCB return area storage key.
LSPARS07		X'07'	Invalid UCB address.
LSPARS08		X'08'	Invalid virtual UCB address.
LSPARS09		X'09'	Invalid zero VTOC pointer.
LSPAUCB	12(X'0C')	4	UCB address.
LSPAFRSP	16(X'10')	4	Address of message or data return area.
LSPAFMT4	20(X'14')	4	Address of format-4 DSCB.

Note: For more information about the LSPACE return code, subfunction code, and the subfunction return and reason codes, see Figure 11 on page 22 and *DFSMS/MVS DFSMSdfp Diagnosis Reference*.

Return Codes from LSPACE

Control returns to the instruction following the instructions generated by the LSPACE macro.

See Figure 11 on page 22 for a description of the LSPACE return codes.

LSPACE returns four bytes of diagnostic information in register 0. See the “DADSM/CVAF Diagnostic Aids” section of *DFSMS/MVS DFSMSdfp Diagnosis Reference* for a description of this information.

LSPACE Information Return Areas

The LSPACE macro returns status information to the parameter list and, optionally, returns volume information to any of the four following caller requested return areas.[3]

Message Return Area: LSPACE returns information to a 30-byte message return area (Figure 10). If you provide a message return area with the MSG option, LSPACE returns EBCDIC text, qualified by return codes as shown in Figure 11.

```
LSPMSG   DSECT           Message Area
LSPMTEXT DS    CL30      Message Text
```

Figure 10. DADSM LSPACE Free Space Information MF=(D,MSG)

Figure 11. DADSM LSPACE Message Return Area Contents

Return Code	Description
0(X'00')	Text: SPACE=*aaaa*,*bbbb*, *cccc*/*dddd*,*eeee* where: *aaaa* = Total number of free cylinders *bbbb* = Total number of additional free tracks *cccc* = Total number of free extents *dddd* = Number of cylinders in largest free extent *eeee* = Number of additional tracks in largest free extent **Notes:** 1. For *aaaa, bbbb, and cccc,* the maximum value indicated is 9999 even if the actual value is greater. 2. Note that there is a blank after each of the three commas (after the variables in the text)
4(X'04')	Text: LSPACE—PERMANENT I/O ERROR
8(X'08')	Text: LSPACE—NON-STANDARD OS VOLUME
12(X'0C')	Text: LSPACE—UCB NOT READY
	Text: LSPACE—UCBVTOC IS ZERO
	Text: LSPACE—INVALID PARAMETER
	Text: LSPACE—NOT A DIRECT ACCESS VOL
16(X'10')	No text returned (invalid parameter list or SMF indicator)

Expanded Message Return Area: LSPACE returns information to a 40-byte expanded message return area (Figure 12 on page 23). By providing an expanded message return area with the EXPMSG option, LSPACE returns EBCDIC text, qualified by return codes as shown in Figure 11.

Note: The expanded message return area displays the same return code information as the standard message return area except that the space information provided for return code zero consists of six-digit values (*aaaaaa* instead of *aaaa*).

[3] Requests for the MSG, EXPMSG, and DATA areas are mutually exclusive. LSPACE checks to ensure that the storage key of each information return area is equal to the caller's key or that the caller is authorized prior to its use.

```
LSPMSG   DSECT           Expanded Message Area
LSPETEXT DS    CL40      Expanded Message Text
```

Figure 12. DADSM LSPACE Free Space Information Format, MF=(D,EXPMSG)

Data Return Area: If you provide a data return area with the DATA option, LSPACE returns information as described in Figure 13.

Figure 13. LSPACE Data Return Area Format

Name	Offset	Bytes	Description
LSPDRETN	0(X'00')	1	Return area status byte.
LSPDSPAC		1...	Returned space information.
LSPDF0CN		.1..	Returned format-0 DSCB count.
LSPDVRCN		..1.	Returned free VIR count.
LSPDRRES		. . . x xxxx	Reserved.
LSPDSTAT	1(X'01')	1	Status byte.
LSPDIXDS		X'80'	VTOC index data set exists. See the LSPDIXAC flag to determine if the index is active.
LSPDIXAC		X'40'	Index VTOC active.
LSPDIRES		. . xx xxxx	Reserved.
LSPDRSV1	2(X'02')	2	Reserved.
LSPDNEXT	4(X'04')	4	Number of free extents.
LSPDTCYL	8(X'08')	4	Total free cylinders.
LSPDTTRK	12(X'0C')	4	Total additional free tracks.
LSPDLCYL	16(X'10')	4	Number of cylinders in largest free extent.
LSPDLTRK	20(X'14')	4	Number of additional tracks in largest free extent.
LSPDF0S	24(X'18')	4	Format-0 DSCB count.
LSPDVIRS	28(X'1C')	4	Free VIR count.
LSPDFRAG	32(X'20')	4	Fragmentation index. The fragmentation index is a numeric representation of the relative size and distribution of free space on the volume. A large index value indicates a high degree of fragmentation.

Format-4 DSCB Return Area: If you provide a format-4 DSCB return area with the F4DSCB option, LSPACE returns information to it as described in Figure 6 on page 10.

LSPACE Examples

The following example returns free space information in the message return area.

```
LSPAMFIM LSPACE MSG=MYMSG,UCB=(R10),MF=I
```

The following example returns free space information in the data return area.

```
LSPAMFID LSPACE DATA=MYDATA,UCB=(R10),MF=I
```

The following example uses the list form of the macro to define the parameter list, and the execute form to refer to the same parameter list.

```
LSPALIST LSPACE MSG=MYDATA,MF=L
        .
        .
        .
LSPAEX   LSPACE MF=(E,LSPALIST),UCB=(R10)
```

Reading DSCBs from the VTOC Using OBTAIN

The following section discusses using the OBTAIN routine to read a DSCB. You can specify either the data set name or the absolute device address.

OBTAIN does not support HFS files. You will receive unpredictable results if you issue OBTAIN for an HFS file.

Reading a DSCB by Data Set Name

If you specify a data set name using OBTAIN and the CAMLST SEARCH option, the OBTAIN routine reads the 96-byte data portion of the format-1 DSCB and the absolute track address of the DSCB into virtual storage. The absolute track address is a 5-byte field in the form CCHHR that contains zeros for VSAM and VIO data sets.

Because the VTOC lacks a format-1 DSCB for some VSAM entries, an OBTAIN request that searches the VTOC for such a DSCB will fail to find it. If the VSAM entry information is available in the catalog, OBTAIN uses information from the catalog to build a pseudo format-1 DSCB, setting its CCHHR to zeros.

The format of the OBTAIN and CAMLST macros is:

```
►►──┬──────────┬──OBTAIN──listname_addrx──────────────────────────►◄
    └─label────┘
```

```
►►──listname──CAMLST──SEARCH──,dsname_relexp──,vol_relexp──────────►
►──,wkarea_relexp──────────────────────────────────────────────────►◄
```

listname_addrx
: points to the parameter list (labeled listname) set up by the CAMLST macro instruction.

SEARCH
: code this operand as shown.

dsname_relexp
: specifies the virtual storage location of a fully-qualified data set name. The area that contains the name must be 44 bytes long.

Note: A DSNAME of 44 bytes of X'04' (X'040404...04') can be used to read a format-4 DSCB.

vol_relexp
: specifies the virtual storage location of the 6-byte volume serial number on which the DSCB is located.

wkarea_relexp
specifies the virtual storage location of a 140-byte work area that you must define.

Example: In the following example, the format-1 DSCB for data set A.B.C is read into virtual storage using the SEARCH option. The serial number of the volume containing the DSCB is 770655.

```
          OBTAIN    DSCBABC           READ DSCB FOR DATA
*                                     SET A.B.C INTO DATA
*                                     AREA NAMED WORKAREA

DSCBABC   CAMLST    SEARCH,DSABC,VOLNUM,WORKAREA
DSABC     DC        CL44'A.B.C'       DATA SET NAME
VOLNUM    DC        CL6'770655'       VOLUME SERIAL NUMBER
WORKAREA  DS        140C              140-BYTE WORK AREA
```

Note: Check the return codes.

The OBTAIN macro instruction points to the CAMLST parameter list. SEARCH, the first operand of CAMLST, specifies that a DSCB be read into virtual storage using the data set name at the address indicated in the second operand. DSABC specifies the virtual storage location of a 44-byte area containing the fully-qualified name of the data set whose format-1 DSCB is to be read. VOLNUM specifies the virtual storage location of a 6-byte area in which you have placed the serial number of the volume containing the required DSCB. WORKAREA specifies the virtual storage location of a 140-byte work area into which the DSCB is to be returned.

Control is returned to your program at the next executable instruction following the OBTAIN macro instruction. If the DSCB has been successfully read into your work area, register 15 contains zeros. Otherwise, register 15 contains one of the return codes shown in Figure 14.

Return Codes from OBTAIN (Reading by Data Set Name)

Figure 14. DADSM OBTAIN Return Codes

Return Code	Description
0(X'00')	Successful completion of OBTAIN routine.
4(X'04')	The required volume was not mounted.
8(X'08')	The format-1 DSCB was not found in the VTOC of the specified volume.
12(X'0C')	A permanent I/O error was encountered, or an invalid format-1 DSCB was found when processing the specified volume, or an unexpected error return code was received from CVAF (common VTOC access facility).
16(X'10')	Invalid work area pointer.

After executing these macro instructions, the first 96 bytes of the work area contain the data portion of the format-1 or format-4 DSCB; the next 5 bytes contain the absolute track address (CCHHR) of the DSCB. For VSAM or VIO data sets, these 5 bytes contain zeros.

Reading a DSCB by Absolute Device Address

You can read a DSCB from a VTOC using OBTAIN and the CAMLST SEEK option. Specify the SEEK option by coding SEEK as the first operand of the CAMLST macro and by providing the absolute device address of the DSCB you want to read, unless the DSCB is for a VIO data set. Only the SEARCH option can be used to read the DSCB of a VIO data set.

The format of the OBTAIN and CAMLST macros is:

```
►►──┬──────────┬──OBTAIN──listname_addrx──────────────────────────────►◄
    └──label───┘
```

```
►►──listname──CAMLST──SEEK──,cchhr_relexp──,vol_relexp──────────────►
►──,wkarea_relexp──────────────────────────────────────────────────►◄
```

listname_addrx
points to the parameter list (labeled listname) set up by the CAMLST macro instruction.

SEEK
code this operand as shown.

cchhr_relexp
specifies the virtual storage location of the 5-byte absolute device address (CCHHR) of a DSCB.

vol_relexp
specifies the virtual storage location of the 6-byte volume serial number on which the DSCB is located.

wkarea_relexp
specifies the virtual storage location of a 140-byte work area that you must define.

Example: In the following example, the DSCB at actual-device address X'00 00 00 01 07' is returned in the virtual storage location READAREA, using the SEEK option. The DSCB resides on the volume with the volume serial number 108745.

```
         OBTAIN   ACTADDR         READ DSCB FROM
*                                 LOCATION SHOWN IN CCHHR
*                                 INTO STORAGE AT LOCATION
*                                 NAMED READAREA

ACTADDR  CAMLST   SEEK,CCHHR,VOLSER,READAREA
CCHHR    DC       XL5'0000000107' ABSOLUTE TRACK ADDRESS
VOLSER   DC       CL6'108745'     VOLUME SERIAL NUMBER
READAREA DS       140C            140-BYTE WORK AREA
```

Note: Check the return codes.

The OBTAIN macro points to the CAMLST parameter list. SEEK, the first operand of CAMLST, specifies that a DSCB be read into virtual storage. CCHHR specifies the storage location containing the 5-byte actual-device address of the DSCB. VOLSER specifies the storage location containing the serial number of the volume where the DSCB resides. READAREA specifies the storage location to which the 140-byte DSCB is to be returned.

Control is returned to your program at the next executable instruction following the OBTAIN macro instruction. If the DSCB has been read into your work area, register 15 contains zeros. Otherwise, register 15 contains one of the return codes shown in Figure 15.

Return Codes from OBTAIN (Reading by Absolute Device Address)

Figure 15. DADSM OBTAIN Return Codes

Return Code	Description
0(X'00')	Successful completion of OBTAIN routine.
4(X'04')	The required volume was not mounted.
8(X'08')	The format-1 DSCB was not found in the VTOC of the specified volume.
12(X'0C')	A permanent I/O error was encountered or an unexpected error return code was received from CVAF.
16(X'10')	Invalid work area pointer.
20(X'14')	The SEEK option was specified and the absolute track address (CCHHR) is not within the boundaries of the VTOC.

Releasing Unused Space from a DASD Data Set Using PARTREL

DADSM supports the release of unused space that is allocated to sequential or partitioned data sets, PDSEs, sequential extended-format data sets, and VSAM extended format data sets. The partial release function is called when:

- The data set is closed (if the RLSE subparameter of SPACE was specified on its DD statement or the storage administrator specified an appropriate value for the partial release option in the management class definition).
- A restart is processing from a checkpoint taken before the data set was extended.
- DFSMShsm performs a space management cycle and an appropriate value is specified in the management class definition.
- A PARTREL macro is issued.

The PARTREL Macro

The PARTREL macro supports sequential and partitioned data sets on volumes with or without indexed VTOCs. It supports PDSEs and extended format data sets on SMS-managed volumes for which indexed VTOCs are required. PARTREL does not support HFS files. You will receive unpredictable results if you issue PARTREL for HFS files.

PARTREL can be coded in the execute, DSECT, and list forms, but not the standard form. The calling program:

- Must be APF authorized.

- Must have allocated the volume to this task and must ensure it stays mounted during the PARTREL function.
- Must ensure that the data set is not open.
- Must not hold any locks or an ENQ on the VTOC.
- Must provide the address of an available standard register save area in general register 13.
- Can provide the associated parameter list and parameters in storage either above or below 16MB virtual.
- Can be in any storage key.
- Can run in either supervisor or problem program state.
- Can be in either 24 or 31-bit addressing mode.
- Requires the address of a UCB, not a UCB copy.

PARTREL—Execute Form

The format of the execute form of the PARTREL macro is:

```
►►─┬──────────┬─PARTREL─MF=─┬─(E,address)─┬─┬──────────────────────────►
   └─label────┘             └─(E,(reg))───┘ └─,DSN=─┬─addr─┬─┘
                                                    └─(reg)┘

►─┬──────────────────────────┬─┬──────────────────────────┬──────────────►
  └─,ERASE=─┬─TEST─┬─┬─┬─┘ └─,MODE=─┬─PGM─┬─┬─┬─┘
            ├─YES──┤ │ │             └─SUP─┘ │ │
            └─NO───┘ │ │                     │ │

►─┬──────────────────────────┬─┬──────────┬─┬──────────────────────────┬──►
  └─,TIOT=─┬─ENQ──┬─┬─┬─┘ └─,UCB=(reg)─┘ └─,VSAMTYPE=─┬─KSDS─┬─┬─┬─┘
           └─NOENQ─┘ │ │                               ├─RRDS─┤ │ │
                     │ │                               ├─LDS──┤ │ │
                     │ │                               └─VRRDS┘ │ │

►─┬──────────────────────────┬─┬──────────────────────────────┬──────────►
  └─,VSAMFMT=─┬─EXTD─┬─┬─┬─┘ └─,VSAMCLLR=─┬─VSAMCLSE─┬─┬─┬─┘
              └─STD──┘ │ │                 └─NOTVSAM──┘ │ │
                       │ │                              │ │

►─┬──────────────────────────┬─┬──────────────────────────┬──────────►◄
  └─,VSAMSEC=─┬─BYPASS───┬─┬─┬─┘ └─,VSAMTRKS=─┬─addr─┬─┬─┬─┘
              └─NOBYPASS─┘ │ │                └─(reg)┘ │ │
                           │ │                         │ │
```

Except for MODE, all parameters default to the current contents of the parameter list. The MODE parameter defaults to PGM.

Descriptions of these parameters include information about DADSM processing. The descriptions use the term *value* to designate the parameter value passed to DADSM. The value can be:

- Specified as a parameter on the PARTREL macro
- Provided as the parameter's associated value in the parameter list
- Defined by DADSM from the information provided in the request.

MF=(E,*addr*) or **(E,(*reg*))**
specifies the execute form of the macro and the address of an existing PARTREL parameter list.

addr—RX-type address, **(***reg***)**—(0-12)
specifies the PARTREL parameter list address.

DSN=*addr* or **(***reg***)**
specifies the address of a 44-byte area that contains the data set name. The name must be left-justified, with any unused bytes defined as blanks.

addr—RX-type address, **(***reg***)**—(0), (2-12)
You must provide a value for DSN.

ERASE=YES or **NO** or **TEST**
specifies a residual data erase attribute (see "Deleting a Data Set from the VTOC" on page 59 for a description of erase attributes). ERASE=YES and ERASE=NO are mutually exclusive. The default is ERASE=TEST. If you specify VSAMTYPE, ERASE is ignored.

ERASE=YES
specifies that the area being released should be erased (overwritten with zeros) before it is made available for new allocations.

ERASE=NO
specifies that the area should not be erased. This specification overrides the RACF erase attribute.

ERASE=TEST
specifies that the associated RACF erase attribute is to be used.

MODE=PGM or **SUP**
specifies that PARTREL is requested by a caller in problem program state (MODE=PGM) or in supervisor state (MODE=SUP). MODE=PGM is the default.

If the calling program is in supervisor state (and wants control returned in supervisor state), the value of MODE must be SUP. If the calling program is in problem program state, the value of MODE must be PGM.

TIOT=ENQ or **NOENQ**
specifies the desired SYSZTIOT and SYSDSN ENQ processing within partial release. The default is ENQ. If you specify VSAMTYPE, TIOT is ignored.

TIOT=ENQ
specifies that partial release is to do its normal, exclusive ENQ on SYSZTIOT and SYSDSN. If either of these ENQ requests fails, PARTREL terminates the request with a return code of `X'08'`.

TIOT=NOENQ
specifies that the caller has provided the necessary serialization. If partial release finds that the caller does not have exclusive use of SYSDSN, PARTREL terminates the request with a return code of `X'24'`.

UCB=(*reg***)**
specifies the address of the UCB for the volume on which the subject data set resides. The UCB address can be for a captured UCB, or for an actual UCB above or below the 16MB line. For 31-bit callers, the high-order byte is part of the UCB address and must be cleared to zeros if a 24-bit UCB address is being passed. The volume must be mounted, and you must ensure that it remains mounted. Use the address of a UCB, not a UCB copy.

(*reg***)**—(0), (2-12)
specifies a register containing the UCB address for the device.

VSAMTYPE=KSDS or **ESDS** or **RRDS** or **LDS** or **VRRDS**
specifies the type of VSAM data set.

Currently, an extended key-sequenced data set (KSDS) is the only type of VSAM data set which is supported. You receive an invalid data set type return code if you specify any other type.

If you specify VSAMTYPE, you *must* also specify VSAMFMT, VSAMCLLR, VSAMSEC, and VSAMTRKS.

If you specify VSAMTYPE, the ERASE= parameter is ignored.

VSAMTYPE=KSDS
specifies a VSAM key-sequenced data set.

VSAMTYPE=ESDS
specifies a VSAM entry-sequenced data set.

VSAMTYPE=RRDS
specifies a VSAM fixed-length relative record data set.

VSAMTYPE=LDS
specifies a VSAM linear data set.

VSAMTYPE=VRRDS
specifies a VSAM variable-length relative record data set.

VSAMFMT=EXTD or **STD**
specifies the format of the data set.

VSAMFMT=EXTD
specifies that this is an extended format data set.

VSAMFMT=STD
specifies that this is *not* an extended format data set. If you specify STD catalog services returns an error to partial release.

VSAMCLLR=VSAMCLSE or **NOTVSAM**
specifies the caller.

VSAMCLLR=VSAMCLSE
specifies that the caller is VSAM CLOSE.

VSAMCLLR=NOTVSAM
specifies that the caller is other than VSAM CLOSE.

VSAMSEC=BYPASS or **NOBYPASS**
specifies if security checking is to be performed

VSAMSEC=BYPASS
specifies that security checking is *not* to be performed.

VSAMSEC=NOBYPASS
specifies that security checking is to be performed.

VSAMTRKS=*addr* or **(***reg***)**
specifies the address of a 4-byte area that is used to contain the number of tracks released for this data set.

You must provide an address for VSAMTRKS only if you code VSAMTYPE.

addr—RX-type address, **(***reg***)**—(0), (2-12)
For a VSAM extended format data set, PARTREL returns a value which is the sum of all the space released for all the parts of the data set.

PARTREL—List Form

The format of the list form of the PARTREL macro is:

```
►►─┬──────────┬─PARTREL──MF=L──┬──────────────┬──┬──────────────────────────────┬──►
   └─label────┘                └─,DSN=addr────┘  │                              │
                                                 └─,ERASE=─┬─TEST─┬─┬──────────┬─┘
                                                           ├─YES──┤ │
                                                           └─NO───┘ │
►──┬──────────────────────────┬──┬──────────────────────────┬──────────────────────►
   │                          │  │                          │
   └─,MODE=─┬─PGM─┬─┬──────┬─┘  └─,TIOT=─┬─ENQ───┬─┬──────┬─┘
            └─SUP─┘ │      │             └─NOENQ─┘ │      │
►──┬──────────────────────────┬──┬──────────────────────────┬──────────────────────►
   │                          │  │                          │
   └─,VSAMTYPE=─┬─KSDS──┬─┬──┬─┘  └─,VSAMFMT=─┬─EXTD─┬─┬──┬─┘
                ├─RRDS──┤ │  │                └─STD──┘ │  │
                ├─LDS───┤ │  │                         │  │
                └─VRRDS─┘ │  │                         │  │
►──┬──────────────────────────┬──┬──────────────────────────┬──────────────────────►
   │                          │  │                          │
   └─,VSAMCLLR=─┬─VSAMCLSE─┬─┬─┬─┘  └─,VSAMSEC=─┬─BYPASS───┬─┬─┬─┘
                └─NOTVSAM──┘ │ │                └─NOBYPASS─┘ │ │
►──┬──────────────────────────┬──────────────────────────────────────────────────►◄
   │                          │
   └─,VSAMTRKS=addr───────────┘
```

Notes:

1. The execute form of the UCB parameter cannot be specified on the list form.
2. The list form MODE parameter is for documentation only. The value of MODE is as specified or defaulted on the execute form.

For an explanation of the parameters, see the execute form.

PARTREL—DSECT Form

The format of the DSECT form of the PARTREL macro is:

```
►►─┬──────────┬─PARTREL──MF=D──────────────────────────────────────────────────►◄
   └─label────┘
```

Example: An example of the DSECT form's expansion is:

```
 PRELPL   PARTREL MF=D
+PRELPL   DSECT         DSECT FOR PARAMETER LIST
+PRLPLID  DS    CL4     EBCDIC 'PREL' FOR PARTREL
+PRLNGTH  DS    AL2     LENGTH OF PARAMETER LIST
+PRERRCDE DS    H       ERROR CODE RETURNED FROM
+*                        PARTIAL RELEASE
+PRLFLAG  DS    XL1     PARAMETER FLAG BYTE
+PRLPGM   EQU   X'00'   MODE=PGM (PROBLEM PROGRAM)
+PRLSUP   EQU   X'80'   MODE=SUP (SUPERVISOR STATE)
```

```
+PRLTIOT  EQU   X'40'   TIOT=NOENQ
+PRLNERAS EQU   X'20'   ERASE=NO
+PRLERASE EQU   X'10'   ERASE=YES
+PRLVCLOS EQU X'08'     VSAMCLLR=VSAMCLSE
+PRLVCNV  EQU X'04'     VSAMCLLR=NOTVSAM
+PRLBYPS  EQU X'02'     VSAMSEC=BYPASS
+PRLNBYPS EQU X'01'     VSAMSEC=NOBYPASS
+PRLFLAG2 DS XL1
+PRLVKSDS EQU X'80'     VSAMTYPE=KSDS
+PRLVESDS EQU X'40'     VSAMTYPE=ESDS
+PRLVRRDS EQU X'20'     VSAMTYPE=RRDS
+PRLVVRRD EQU X'10'     VSAMTYPE=VRRDS
+PRLVLDS  EQU X'08'     VSAMTYPE=LDS
+PRLEXTD  EQU X'04'     VSAMFMT=EXTD
+PRLSTD   EQU X'02'     VSAMFMT=STD
+PRLRSVD  DS XL2        RESERVED
+PRLDSN   DS A          DATA SET NAME POINTER
+PRLUCB   DS A          UCB POINTER
+PRLTRKS  DS A          ADDRESS OF NUMBER OF TRACKS RETURNED
                         (ONLY VALID FOR VSAM REQUESTS)
+PRLCTGR  DS F          CATALOG REASON CODE
+PRLEND   EQU   *       END OF PARAMETER LIST
+PRLENGTH EQU   PRLEND-PRELPL       LENGTH OF PARAMETER LIST
```

Return Codes From PARTREL

Control returns to the instruction following the last instruction generated by the PARTREL macro. Register 15 contains the applicable PARTREL return code.

If an error occurs, PARTREL issues message IEC614I, consisting of failure-related status information.

- If an error results from a CVAF function, the subfunction return code field contains the CVAF return code, and the subfunction reason code field contains the CVAF status code (CVSTAT).
- If an error results from the execution of an EXCP channel program, the subfunction return code and reason code fields contain either:
  - The ECB completion code and the CSW channel status, if the ECB completion code is not X'41' and the channel status is not zero.
  - The sense bytes (two) from the IOB, if the ECB completion code is X'41' and there is no channel status.
- If an error results from an RACF invocation, the subfunction return code and reason code fields contain the RACF return code and reason code.

Figure 16 describes the conditions indicated by the PARTREL return code.

Note: This is a cumulative list of DADSM partial release return codes. Some of these codes might not apply to the PARTREL macro.

Figure 16 (Page 1 of 2). DADSM PARTREL Return Codes

Return Code	Description
0(X'00')	If the 4 bytes of diagnostic information returned in register 0 are all zeros, this indicates successful partial release. If they are nonzero, see the “DADSM/CVAF” section of *DFSMS/MVS DFSMSdfp Diagnosis Reference* to determine the failure-related conditions.

Figure 16 (Page 2 of 2). DADSM PARTREL Return Codes

Return Code	Description
2(X'02')	Unable to find extent in the VTOC.
4(X'04')	Unable to find extent in the VTOC.
8(X'08')	Either the required SYSZTIOT or SYSDSN ENQ failed, or an unrelated DEB indicates that another DCB is open to the data set.
12(X'0C')	Invalid parameter list.
16(X'10')	One of the following conditions occurred: • A permanent I/O error occurred. • CVAF provided an unexpected return code. • An installation exit rejected the request. • An I/O error occurred while the tracks to be released were being erased (for ERASE-on-SCRATCH).
20(X'14')	DSN, or DSN pointer is invalid.
24(X'18')	Invalid UCB pointer.
28(X'1C')	Specified DSORG is not supported.
32(X'20')	No room in the VTOC.
36(X'24')	Invalid TIOT=NOENQ request; caller does not have exclusive use of SYSDSN.
40(X'28')	An error occurred while SMS was processing the request.
44(X'2C')	CLOSE is the caller, user rejected the partial release using the PREEXIT routine.
48(X'30')	An error occurred during conversion from CCHH to relative track address.
52(X'34')	An error occurred during sort conversion of the extent descriptor table.
56(X'38')	An error occurred during conversion from new extent descriptor table to old extent descriptor table.
60(X'3C')	An error occurred during conversion from relative track address to CCHH.
64(X'40')	An error occurred during conversion from Format-7 to extent descriptor table.
68(X'44')	An error occurred during conversion from format-5 to extent descriptor table.
72(X'48')	Input DSCB is not a format-5 or a format-7 DSCB.
76(X'4C')	An error occurred during conversion from extent descriptor table to format-7.
80(X'50')	An error occurred during conversion from extent descriptor table to format-5.
84(X'54')	VSAM standard format data set incorrectly specified. If the data set is VSAM, it must be extended format.
88(X'58')	VSAMTRKS address parameter must be specified when VSAMTYPE is specified.
92(X'5C')	The name of the specified VSAM data set is not a VSAM cluster data component.
96(X'60')	Catalog call to partial release failed.
100(X'64')	Request was directed to a VSAM data set defined with guaranteed space.
104(X'68')	Unexpected reason code received back from a call to catalog services.

Creating (Allocating) a DASD Data Set Using REALLOC

The REALLOC macro builds a parameter list to allocate a new data set. You can code the macro in the execute, DSECT, and list forms, but not in the standard form. The calling program:

- Must be APF authorized
- Must allocate the volume to this address space and must ensure it stays mounted during the REALLOC function
- Must not hold any locks
- Can provide the associated parameter list and parameters in storage either above or below 16MB
- Can use any storage key
- Can run in either supervisor or problem program state
- Can be in either 24 or 31-bit addressing mode
- Must note that REALLOC does not call RACF or catalog management
- Must note that REALLOC cannot create data sets on SMS-managed volumes (SMS-managed data sets can be created through JCL or dynamic allocation)
- Must note that REALLOC cannot create PDSEs, HFS data sets, or extended format data sets
- Requires the address of a UCB, not a UCB copy.

The calling program must provide the REALLOC macro with one or more model DSCBs. You can use the OBTAIN macro to get the DSCBs from other data sets and modify them for the request. DADSM uses these model DSCBs to validate the allocation request, and to construct the DSCBs written to the VTOC for the requested allocation.

The ALLOC parameter for the REALLOC macro defines the allocation request as either absolute (ABS) or movable[4] (MOV).

An absolute request is limited to a single volume with indexed VTOC support. An absolute request provides a set of allocation parameters, a full format-1 DSCB, and optional format-2 and format-3 DSCBs that describe the attributes of the desired data set:

- Support is provided for data sets with a user label extent and for ISAM data set allocations,[5] but is not limited to these two types.
- The number of extents to be allocated, and their absolute placement on the volume, are defined by the format-1 DSCB and one (optional) format-3 DSCB.

---

[4] The requested data set's allocation is not sensitive to its placement on the volume. This is not a reference to the format-1 DSCB bit DS1DSGU (unmovable bit), that can be either on or off in an ALLOC=MOV request's partial DSCB. That is, the data set can subsequently contain location-dependent information.

[5] See the description of the F2DSCB parameter in the REALLOC macro execute form for more information.

A movable request is limited to a single volume with or without indexed VTOC support. A movable request provides a set of allocation parameters and a partial DSCB[6] that describe the attributes of the desired data set:

- ISAM data sets and absolute track allocated data sets are not supported.
- The maximum number of extents that can be allocated is determined by the data set organization (PD1DSORG) and the data set indicator (PD1DSIND) bytes in the partial DSCB. If PD1DSORG indicates a VSAM data set organization and PD1DSIND indicates that the data set is cataloged in an integrated catalog facility catalog, the maximum number of extents is 255 across multiple volumes, 123 on a single volume.

REALLOC—Execute Form

The format of the execute form of the REALLOC macro is:

```
►►─┬──────────┬─REALLOC─MF=─┬─(E,addr)─┬──┬──────────────────────────────►
   └─label────┘             └─(E,(reg))─┘  │          ┌─MOV─┐          │
                                           └─,ALLOC=─┴─ABS─┴──────────┘
►─┬──────────────────────┬─┬──────────────────────┬──────────────────────►
  └─,DSSIZE=─┬─addr─┬────┘ └─,F2DSCB=─┬─addr─┬────┘
             └─(reg)─┘                └─(reg)─┘
►─┬──────────────────────┬─┬──────────────────────┬──────────────────────►
  └─,F3DSCB=─┬─addr─┬────┘ └─,MINAU=─┬─addr─┬────┘
             └─(reg)─┘                └─(reg)─┘
►─┬──────────────────────┬─┬──────────────────────┬─┬──────────────┬──►◄
  └─,PDSCB=─┬─addr─┬────┘ └─,PDSDIR=─┬─addr─┬────┘ └─,UCB=(reg)──┘
            └─(reg)─┘                └─(reg)─┘
```

All parameters except ALLOC default to the current contents of the referenced parameter list. The ALLOC parameter defaults to MOV.

MF=(E,*addr*) or **(E,(*reg*))**
specifies the execute form of the macro and the address of a REALLOC parameter list.

addr—RX-type address, **(*reg*)**—(0-12)
specifies the address of the REALLOC parameter list.

ALLOC=ABS or **MOV**
specifies that the REALLOC request is for absolute extents (ALLOC=ABS) or for a movable allocation (ALLOC=MOV). ALLOC=MOV is the default.

DSSIZE=*addr* or **(*reg*)**
specifies the size of the data set to be allocated in tracks. The DSSIZE parameter is invalid for an ALLOC=ABS request.

addr—RX-type address
specifies the address of a word that contains the data set size.

[6] The partial DSCB (mapped by the IECPDSCB macro) consists of the first 106 bytes of a format-1 DSCB followed by 10 bytes in which the primary space request and number of directory blocks are specified.

(*reg***)**—(0), (2-12)
specifies a register that contains the size of the data set.

Provide a value for DSSIZE for an ALLOC=MOV request. The PDPRIQTY field of the partial DSCB is ignored.

REALLOC assumes that you have provided the value of DSSIZE in tracks even if the PD1SCALO flag byte of the partial DSCB indicates a cylinder request (`X'C0'`), or an average block request, (`X'40'`).

If the PD1SCALO flag byte of the partial DSCB indicates a cylinder request (`X'C0'`), or an average block with round request (`X'41'`), the value of DSSIZE is rounded up to the next full cylinder.

F2DSCB=*addr* or **(***reg***)**
specifies the in-storage address of a format-2 DSCB. This DSCB is used as a model to construct the allocated data set's format-2 DSCB.

The F2DSCB parameter is invalid for an ALLOC=MOV request.

addr—RX-type address, (*reg*)—(0), (2-12)

You can provide a value for F2DSCB in an ALLOC=ABS request when the DS1DSORG flag byte of the given format-1 DSCB is `X'80'` (indexed sequential organization).

Because REALLOC allocates on the basis of a single volume for each request and because a multivolume ISAM data set is defined with one format-2 DSCB (on the first volume only), the associated DADSM allocation processing routines do not require a format-2 DSCB (that is, multiple REALLOC requests can be used to allocate a multivolume ISAM data set).

F3DSCB=*addr* or **(***reg***)**
specifies the in-storage address of a format-3 DSCB. This DSCB is used as a model to construct the allocated data set's format-3 DSCB.

The F3DSCB parameter is invalid for an ALLOC=MOV request.

addr—RX-type address, (*reg*)—(0), (2-12)

Provide a value for F3DSCB in an ALLOC=ABS request when the DS1NOEPV byte of the format-1 DSCB indicates more than three extents (or when the DS1NOEPV byte indicates more than two extents and the DS1EXT1 extent type indicator is `X'40'`, a user label extent).

The REALLOC request is limited to a maximum of 16 extents when the F3DSCB keyword is specified. No more than one format-3 DSCB can be specified.

Enter a value of zero for the F3DSCB in an ALLOC=ABS request when the DS1NOEPV byte of the format-1 DSCB indicates that there are less than four extents (or when the DS1NOEPV byte indicates that there are less than three extents and the DS1EXT1 extent type indicator is `X'40'`, a user label extent).

MINAU=*addr* or **(***reg***)**
specifies the size of the minimum allocation unit in tracks. All primary extents for this data set are in multiples of this minimum allocation unit. This minimum does not apply to subsequent extensions of the data set.

The MINAU parameter is invalid on an ALLOC=ABS request.

addr—RX-type address
specifies the address of a word containing the minimum allocation unit.

(*reg***)**—(0), (2-12)
specifies a register containing the minimum allocation unit.

The MINAU parameter has no effect on the requested allocation if:

- You provide a value of zero.
- The PD1SCALO flag byte of the partial DSCB indicates either a cylinder request (`X'C0'`) or an average block with round request (`X'41'`).

Otherwise, the value of DSSIZE must be a multiple of the value of MINAU.

PDSCB=*addr* or **(***reg***)**
specifies the address of a partial DSCB (for ALLOC=MOV) or the in-storage address of a full format-1 DSCB (for ALLOC=ABS). This DSCB is used as a model to construct the allocated data set's format-1 DSCB.

addr—RX-type address, (*reg*)—(0), (2-12)

Provide a value for the PDSCB parameter and initialize the PD1FMTID field to `X'F1'`.

PDSDIR=*addr* or **(***reg***)**
specifies the number of 256-byte directory blocks for a partitioned data set (PDS).

addr—RX-type address
specifies an in-storage address of a full word containing the number of 256-byte PDS directory blocks.

(*reg***)**—(0), (2-12)
specifies a register containing the number of 256-byte PDS directory blocks.

Provide a value for PDSDIR when partitioned organization is indicated:

- The DS1DSORG flag byte of the format-1 DSCB is `X'02'` (ALLOC=ABS).
- The PD1DSORG flag byte of the partial DSCB is `X'02'` (ALLOC=MOV).

For an ALLOC=MOV request, you can specify the value of PDSDIR in the PDDIRQTY field of the partial DSCB. The PDDIRQTY field is used only if the REALLOC parameter list value of PDSDIR is zero.

Do not specify a value for PDSDIR if a PDS is not indicated.

UCB=(*reg***)**
specifies the address of the UCB for the volume on which the data set is to be allocated. The UCB address can be for a captured UCB, or for an actual UCB above or below the 16MB line. For 31-bit callers, the high-order byte is part of the UCB address and must be cleared to zeros if a 24-bit UCB address is being passed. The volume must be mounted, and remain mounted. Use the address of a UCB, not a UCB copy.

(*reg***)**—(0), (2-12)
specifies a register containing the UCB address for the device.

REALLOC—List Form

The format of the list form of the REALLOC macro follows. See the execute form for an explanation of the parameters.

```
►►──┬──────────┬──REALLOC──MF=L──┬──────────────────────────┬──┬──────────────────┬──►
    └─label────┘                 └──,ALLOC=──┬─MOV─┬──┘  └──,F2DSCB=addr──┘
                                             └─ABS─┘
►──┬──────────────────┬──┬──────────────────┬──────────────────────────────────────►◄
   └──,F3DSCB=addr────┘  └──,PDSCB=addr─────┘
```

Notes:

1. The execute form parameters DSSIZE, MINAU, PDSDIR, and UCB cannot be specified on the list form.
2. The list form's ALLOC parameter affects the tests made by the REALLOC macro during assembly, and the contents of the parameter list.
3. The value of ALLOC is as specified or defaulted on the execute form.

REALLOC—DSECT Form

The format of the DSECT form of the REALLOC macro is:

```
►►──┬──────────┬──REALLOC──MF=D──────────────────────────────────────────►◄
    └─label────┘
```

REALLOC Parameter List

The parameter list for the DSECT form expansion is:

```
 REALPL   REALLOC MF=D
+REALPL   DSECT                      DSECT FOR PARAMETER LIST
+RALPLID  DS    CL4                  EBCDIC 'REAL' FOR REALLOC
+RALNGTH  DS    AL2                  LENGTH OF PARAMETER LIST
+RAERRCDE DS    H                    ERROR CODE RETURNED FROM
*                                      ALLOCATE (SVC 32)
+RALFLAG  DS    XL1                  PARAMETER FLAG BYTE
+RALMOV   EQU   X'00'                ALLOC=MOV
+RALABS   EQU   X'80'                ALLOC=ABS
+RALFRES  EQU   X'7F'                RESERVED
+RALRSVD  DS    XL3                  RESERVED
+RALDSSZ  DS    F                    DATA SET SIZE
+RALMAU   DS    F                    MINIMUM ALLOCATION UNIT
+RALPDSCB DS    A                    PARTIAL DSCB POINTER
+RALUCB   DS    A                    UCB POINTER
+RALDQTY  DS    F                    PDS DIRECTORY QUANTITY
+RAL2DSCB DS    A                    FORMAT 2 DSCB POINTER
+RAL3DSCB DS    A                    FORMAT 3 DSCB POINTER
+RALEND   EQU   *                    END OF PARAMETER LIST
+RALENGTH EQU   RALEND-REALPL        LENGTH OF PARAMETER LIST
```

Return Codes from REALLOC

Control returns to the instruction following the instructions generated by the REALLOC macro. Register 15 contains the applicable REALLOC return code as shown in Figure 17.

REALLOC returns 4 bytes of diagnostic information in register 0. See *DFSMS/MVS DFSMSdfp Diagnosis Reference*.

If an error occurs, REALLOC issues message IEC614I, consisting of failure-related status information. See Figure 17 for descriptions of the conditions indicated by the REALLOC return code.

Note: This is a cumulative list of DADSM create return codes. Some of these codes might not apply to the REALLOC macro.

Figure 17 (Page 1 of 2). DADSM CREATE Return Codes

Return Code	Description
00(X'00')	If the 4 bytes of diagnostic information returned in register 0 are all zeros, this indicates successful data set creation. If they are nonzero, see the DADSM/CVAF Diagnostic Aids section of *DFSMS/MVS DFSMSdfp Diagnosis Reference* to determine the failure-related conditions.
04(X'04')	Duplicate data set name.
08(X'08')	No room in VTOC or VTOC index.
12(X'0C')	Permanent I/O error or CVAF error.
16(X'10')	Requested absolute track not available.
20(X'14')	Requested quantity not available.
24(X'18')	Average record length exceeds 65,535 bytes.
28(X'1C')	Invalid DSORG for ISAM index request.
32(X'20')	No prime area requested for ISAM.
36(X'24')	ISAM overflow area requested before prime.
40(X'28')	ISAM space request does not start on cylinder boundary.
44(X'2C')	Duplicate ISAM data set element.
48(X'30')	Invalid DADSM REALLOC parameter list.
52(X'34')	Invalid JFCB or partial DSCB pointer.
56(X'38')	Not enough space on volume for directory.
60(X'3C')	REALLOC ALLOC=ABS is not supported on OS VTOCs.
64(X'40')	Invalid user label request.
68(X'44')	Invalid UCB pointer. Requires the address of a UCB, not a UCB copy.
72(X'48')	VSE VTOC cannot be converted to an OS VTOC.
76(X'4C')	No space parameter given for a new data set or zero space requested at absolute track zero.
80(X'50')	Invalid request for ISAM index.
84(X'54')	ISAM multivolume index is not allowed.
88(X'58')	Invalid ISAM data set element.
92(X'5C')	ISAM multivolume overflow is not allowed.
96(X'60')	ABSTR and CYL requests conflict (ISAM).
100(X'64')	CYL and CONTIG request conflict (ISAM).
104(X'68')	Invalid space subparameter.
108(X'6C')	Zero primary space request for ISAM data set or ABSTR request.
112(X'70')	ISAM index request duplication.
116(X'74')	User labels not supported.
120(X'78')	Invalid combination of DSSIZE and MINAU in REALLOC parameter.
124(X'7C')	DSSIZE not a multiple of MINAU.
128(X'80')	Directory space requested is larger than primary space.
132(X'84')	Space request must be ABSTR for a VSE volume.
136(X'88')	Invalid FMT3 DSCB pointer.

Figure 17 (Page 2 of 2). DADSM CREATE Return Codes

Return Code	Description
140(X'8C')	Index must be requested before prime area.
144(X'90')	Invalid third DD card for ISAM allocation.
148(X'94')	Overlapping extents in the VTOC.
156(X'9C')	DADSM CREATE terminated because of possible VTOC errors.
160(X'A0')	ISAM CREATE terminated because of possible VTOC errors.
164(X'A4')	Allocation terminated because of VSE stacked pack format.
168(X'A8')	RACDEF failed, data set already defined.
172(X'AC')	User not authorized to define data set.
176(X'B0')	Installation exit rejected this request with return code 8.
180(X'B4')	Installation exit rejected the request with return code 4.
184(X'B8')	RACF define with modeling specified and model not found.
188(X'BC')	Invalid FMT2 DSCB pointer.
192(X'C0')	Requested data set creation was not allowed by SMS.
196(X'C4')	Requested data set creation was not possible. Register 0 contains additional diagnostic information.
200(X'C8')	The PDSE directory could not be built.
204(X'CC')	VTOC ENQ related failure.
208(X'D0')	I/O error occurred during the allocation of a data set. The data set will be deleted.
212(X'D4')	Request failed due to a presence of split cylinder data set(s) on the volumes.
216(X'D8')	For this type of data set, the primary quantity requested cannot exceed 65,535 tracks.

Accessing the VTOC with CVAF Macros

The common VTOC access facility (CVAF) macros and tasks discussed in this section consist of the following:

CVAFDIR directly accesses DSCBs.
CVAFDSM obtains volume free space information.
CVAFFILT reads sets of DSCBs for one or more DASD data sets.
CVAFSEQ retrieves the following:

- Data set names from an active VTOC index
- DSCBs in physical-sequential order
- DSCBs in data set name order (index required).

CVAFTST determines if a DASD volume has an active VTOC index.

Appendix A, “CVAF VTOC Access Macros” on page 319, contains descriptions of these macros and examples of their use.

When calling CVAF, your program can be in either 24-bit or 31-bit addressing mode. If it is in 31-bit mode, the control blocks shown in Figure 23 on page 50 might reside above the 16MB line. All these areas must be accessible in your program's storage key.

Serializing and Updating

CVAF requires that you provide all necessary system resource serialization for your request. You can ensure the integrity of multiple data elements (sets of DSCBs or VIRs) returned by CVAF only if you adequately serialize system resources and avoid multiple CVAFFILT requests for a set of DSCBs or VIRs. Weigh possible system performance loss because of serialization against the potential loss of data integrity.

Updating without adequate serialization might compromise the integrity of the volume's VTOC, the VTOC index, or any associated data set.

CVAF only complies with requests to modify the volume's VTOC or index from authorized programs.

CVAF assumes that an authorized program holds an exclusive RESERVE (or ENQ) on the qname (major name) of SYSVTOC, and the rname (minor name) of the volume's serial number, with the scope of SYSTEMS. This RESERVE can be made more efficient if Global Resource Serialization or a functional equivalent is active.

The SYSVTOC qname does not serialize access to the format-1 DSCB for a data set. You can provide serialization by allocating the data set with disposition OLD, MOD, or NEW (not SHR). This causes the proper ENQ, ensuring that no other job can update that data set's format-1 DSCB.

Identifying the Volume

If authorized, you can identify the volume to the CVAFDIR, CVAFDSM, CVAFFILT, and CVAFSEQ macros by specifying the address of the UCB. These macros do not accept the address of a UCB copy. If your program is not authorized, specify the address of a SAM or EXCP DEB opened to the volume's VTOC.

The data extent block (DEB) can be obtained by opening a DCB for INPUT, using the RDJFCB and OPEN TYPE=J macros. (After issuing an OPEN TYPE=J macro, the DCB's DCBDEBA field contains the DEB's address.) The DCB's DDNAME must identify a DD statement allocated to the unit whose VTOC is to be accessed. Once your program issues the RDJFCB macro, it must initialize the JFCBDSNM field with the data set name of the format-4 DSCB: 44 bytes of `X'04'`. The RDJFCB macro is described under “RDJFCB Macro Specification” on page 180; the OPEN macro is described under “OPEN—Initialize Data Control Block for Processing the JFCB” on page 193.

If a CVAF macro call specifies IOAREA=KEEP, a subsequent CVAF call using a different CVAF parameter list (CVPL) might omit the UCB and DEB keywords and supply the IOAREA address from the other CVPL by using the IOAREA keyword.

Generating a CVPL (CVAF Parameter List)

The CVAFDIR, CVAFDSM, CVAFFILT, and CVAFSEQ macros use the CVPL to pass parameters to CVAF. The CVAFTST macro expands to provide its only parameter (UCB address) in register 1, and calls the applicable CVAF module. CVAF returns information related to the CVAF request in the CVPL. The CVAFTST macro will accept the address of a UCB or UCB copy. Unauthorized programs can get a copy of the UCB by using the UCBSCAN macro and specifying the COPY, UCBAREA, CMXTAREA, and DCEAREA keywords. The UCB copy, including the extension, must be below the 16MB line and on a word boundary. Data accessed with the DCEAREA can be above the 16MB line. Refer to *OS/390 HCD Planning* for details.

Specifying the CVAFDIR, CVAFDSM, CVAFFILT, or CVAFSEQ macro with MF=L or MF=I (the default) as a subparameter generates a CVPL. Upon return the CV1IVT bit in the CVPL indicates whether the accessed VTOC was indexed or nonindexed. If an error occurs, the CVSTAT field contains feedback data. The CVAF I/O area address is returned in the CVIOAR field and the CVAF filter save

area address is returned in the CVFSA field. To use the generated CVPL to execute a different function than was originally specified, use the MF=E keyword.

To specify a CVAF filter request, use a CVPL generated by the CVAFFILT macro. The CVAFFILT macro generates a CVPL 4 bytes longer (total length = X'44') than that generated by the other CVAF macros (total length = X'40').

The ICVAFPL macro maps the CVPL. The format of the CVPL is shown in Figure 18.

Figure 18 (Page 1 of 2). Format of the CVAF Parameter List

Name	Offset	Bytes	Description
CVPL			
CVLBL	0 (X'00')	4	EBCDIC 'CVPL'
CVLTH	4 (X'04')	2	Length of CVPL: 64 (X'40') for macros other than CVAFFILT. 68 (X'44') for CVAFFILT macro.
CVFCTN	6 (X'06')	1	Function byte (See Figure 19 on page 43).
CVSTAT	7 (X'07')	1	Status information.
CVFL1	8 (X'08')	1	First flag byte.
CV1IVT		1	Indexed VTOC accessed.
CV1IOAR		. 1	IOAREA=KEEP
CV1PGM		. . 1	BRANCH=(YES,PGM)
CV1MRCDS		. . . 1	MAPRCDS=YES
CV1IRCDS		. . . . 1 . . .	IXRCDS=KEEP
CV1MAPIX		 100	MAP=INDEX
CV1MAPVT		 010	MAP=VTOC
CV1MAPVL		 001	MAP=VOLUME
CVFL2	9 (X'09')	1	Second flag byte.
CV2HIVIE		1	HIVIER=YES
CV2VRF		. 1	VRF data exists.
CV2CNT		. . 1	COUNT=YES
CV2RCVR		. . . 1	RECOVER=YES
CV2SRCH		. . . . 1 . . .	SEARCH=YES
CV2DSNLY		. . . . . 1 . .	DSNONLY=YES
CV2VER		. . . . . 1 .	VERIFY=YES
CV2NLEVL		 1	New highest level VIER (output).
CVFL3	10 (X'0A')	1	Third flag byte.
CV3FILT		1	FLTAREA=KEEP
CV3IXERR		. 1	Index error found.
CV3PTR31		. . 1	AMODE31
CV3AVT		. . . 1	Bypass write inhibit error.
CV3ENQD		. . . . 1 . . .	CVAF caller has serialized with another task that has reserved the VTOC.
CV3NOPIN		. . . . . 1 . .	Caller has requested no UCBPIN
CV3HLIXP		. . . . . 1 .	Higher level index pruning done.
CV3RTA4B		 1	CVAFDSM RTA4BYTE specified.
	11 (X'0B')	1	Reserved.
CVUCB	12 (X'0C')	4	UCB address can be captured or actual above or below the 16MB line.
CVDSN	16 (X'10')	4	Data set name address.
CVFCL	16 (X'10')	4	Filter criteria list address.
CVBUFL	20 (X'14')	4	Buffer list address.

Figure 18 (Page 2 of 2). Format of the CVAF Parameter List

Name	Offset	Bytes	Description
CVIRCDS	24 (X'18')	4	Index VIR's buffer list address.
CVMRCDS	28 (X'1C')	4	Map VIR's buffer list address.
CVIOAR	32 (X'20')	4	I/O area address.
CVDEB	36 (X'24')	4	DEB address.
CVARG	40 (X'28')	4	Argument address.
CVSPACE	44 (X'2C')	4	SPACE parameter list address.
CVEXTS	48 (X'30')	4	Extent table address.
CVBUFL2	52 (X'34')	4	New VRF VIXM buffer list address.
CVVRFDA	56 (X'38')	4	VRF data address.
CVCTAR	60 (X'3C')	4	Count area address.
CVFSA	64 (X'40')	4	Filter save area.

The possible contents of the CVFCTN field in the CVPL and their meanings are as follows:

Figure 19. CVFCTN Field of CVPL—Contents and Definitions

Value	Name	Description	
X'01'	CVDIRD	CVAFDIR	ACCESS=READ
X'02'	CVDIWR	CVAFDIR	ACCESS=WRITE
X'03'	CVDIRLS	CVAFDIR	ACCESS=RLSE
X'04'	CVSEQGT	CVAFSEQ	ACCESS=GT
X'05'	CVSEQGTE	CVAFSEQ	ACCESS=GTEQ
X'06'	CVDMIXA	CVAFDSM	ACCESS=IXADD
X'07'	CVDMIXD	CVAFDSM	ACCESS=IXDLT
X'08'	CVDMALC	CVAFDSM	ACCESS=ALLOC
X'09'	CVDMRLS	CVAFDSM	ACCESS=RLSE
X'0A'	CVDMMAP	CVAFDSM	ACCESS=MAPDATA
X'0E'	CVFIRD	CVAFFILT	ACCESS=READ
X'0F'	CVFIRES	CVAFFILT	ACCESS=RESUME
X'10'	CVFIRLS	CVAFFILT	ACCESS=RLSE
X'AA'	CVDMMAPX	CVAFDSM	ACCESS=MAPDATA RTA4BYTE=YES

Using Buffer Lists

A buffer list consists of one or more chained control blocks, each with a header and buffer list entries, obtained and initialized by your program before calling CVAF. The header indicates whether the buffer list is for DSCBs. The entries point to and describe the buffers. You can create buffer lists in two ways:

- Directly, when you fill in the arguments and buffer addresses of DSCBs to be read or written
- Indirectly (by CVAF), when you code the IXRCDS=KEEP and/or MAPRCDS=YES keywords.

The ICVAFBFL macro maps CVAF buffer lists. Figure 20 on page 44 shows the format of a buffer list header. Figure 21 on page 45 shows the format of a buffer list entry.

Buffer List Header

The buffer list header indicates whether the buffer list describes buffers for DSCBs. The DSCB bit must be set to 1 and the VIR bit to zero for CVAF to process a request to read or write a DSCB. Provide buffer lists and buffers in your program's protect key. CVAF uses the protect key and subpool fields in the buffer list header only if you code ACCESS=RLSE.

Each buffer list header contains a count of the number of entries in the buffer list that directly follows the header.

The forward chain address chains buffer lists together. The format of the buffer list header is shown in Figure 20.

Figure 20. Format of a Buffer List Header

Name	Offset	Bytes	Description
BFLHDR	0 (X'00')	8	Buffer list header.
BFLHNOE	0 (X'00')	1	Number of entries.
BFLHFL	1 (X'01')	1	Key and flag byte.
BFLHKEY		xxxx	Protect key of buffer list and buffers.
BFLHVIR		. . . . 1 . . .	Reserved.
BFLHDSCB		. . . . . 1 . .	Buffer list entries describe DSCBs.
		 xx	Reserved.
	2 (X'02')	1	Reserved.
BFLHSP	3 (X'03')	1	Identifies the subpool of buffer list and buffers.
BFLHFCHN	4 (X'04')	4	Forward chain address of next buffer list.

Buffer List Entry

A buffer list contains one or more entries. Each entry provides the buffer address, the length of the DSCB buffer, the argument, and indicates whether the argument is an RBA, a TTR, or a CCHHR. The fields and bit uses are listed below.

- For a DSCB buffer, the RBA bit must be 0, and either the TTR or CCHHR bits must be set to 1 (they must not both be 1).
- The BFLESKIP bit causes an entry to be ignored.
- The BFLEIOER bit is an output indicator from CVAF that indicates an I/O error occurred during reading or writing of the DSCB.
- The BFLELTH field is the length of the buffer; for a DSCB buffer, the length must be 96 or 140.
- The BFLEARG field is the argument (address) of the DSCB. Specify the format of the 5-byte field by setting the BFLECHR, BFLETTR, or BFLERBA bit to 1. The respective BFLEARG values and formats are as follows:

Value	Format
CCHHR	5-byte CCHHR
TTR	0TTR0
RBA	1 byte of 0 followed by a 4-byte relative byte address.

The values for BFLEARG depend on the variables associated with a given request. These are described in the following request-oriented topics.

The format of the buffer list entry is shown in Figure 21 on page 45.

Figure 21. Format of a Buffer List Entry

Name	Offset	Bytes	Description
BFLE	0 (X'00')	12	Buffer list entry.
BFLEFL	0 (X'00')	1	Flag byte.
BFLERBA		100	Argument is RBA.
BFLECHR		010	Argument is CCHHR.
BFLETTR		001	Argument is TTR.
BFLEAUPD		. . . 1	CVAF updated argument field.
BFLEMOD		. . . . 1 . . .	Data in buffer has been modified.
BFLESKIP		. . . . . 1 . .	Skip this entry.
BFLEIOER		. . . . . 1 .	I/O error.
BFLENOVR		 1	If using CVAFDIR to write a 96-byte DSCB, bypass comparing the DSCB key to the data set name.
	1 (X'01')	1	Reserved.
BFLELTH	2 (X'02')	1	Length of DSCB buffer.
BFLEARG	3 (X'03')	5	Argument of DSCB.
BFLEATTR	4 (X'04')	3	TTR of DSCB.
BFLEARBA	4 (X'04')	4	Reserved.
BFLEBUF	8 (X'08')	4	Buffer address.

Using Macro ICVEDT02 to Map the Extents Area

The ICVEDT02 mapping macro is used to map the extent area when you use the CVAFDSM macro and specify RTA4BYTE=YES for indexed VTOCs or nonindexed VTOCs.

The format of the ICVEDT02 mapping macro follows.

Figure 22. Format of ICVEDT02 Mapping Macro

Name	Offset	Bytes	Description
DT2X7EYE	0 (X'00')	8	Identifier="ICVEDT02"
DT2X7LEN	8 (X'08')	4	Length of ICVEDT02.
DT2X7LEV	12 (X'0C')	1	Level number.
DT2X7FLG	13 (X'0D')	1	Flag byte.
DT2X7NF0	14 (X'0E')	2	Number of format-0 DSCBs created.
DT2X7CSR	16 (X'10')	5	CCHHR cursor field. Used by the system. Initialized to zero by caller—then do NOT modify.
DT2X7RES	21 (X'15')	3	Reserved.
DT2X7RTA	24 (X'18')	4	Relative Track Address (RTA) cursor field. Used by the system. Initialized to zero by caller—then do NOT modify.
DT2X7RE2	28 (X'1C')	4	Reserved.
DT2X7ENT	32 (X'20')	4	Number of extent descriptor entries.
-	-	-	Extent descriptor array.
DT2ENTRY	36 (X'24')	8	One entry for each extent.
DT2RTAST	-	4	RTA of start of extent
DT2RTAED	-	4	RTA+1 of end of extent

Accessing the DSCB Directly

The CVAFDIR macro can be used to read or write a DSCB and is described in “CVAFDIR Macro Overview and Specification” on page 319. After a CVAFDIR call, you can test the CV1IVT bit in the CVPL to determine whether the VTOC is indexed or nonindexed.

If the DSCB buffer is 96 bytes, CVAF issues a channel program to verify that the key in the DSCB matches the 44-byte data set name you entered, unless the operation is a write and the BFLENOVR bit is on.

If the buffer is for a 96-byte write and the BFLENOVR bit in the BFLEFL is set to 1, CVAF skips the key verification, improving performance. If you are not certain that the data set name you provide is correct, set the BFLENOVR bit to 0. If the BFLENOVR bit is set to 0, CVAF does not execute the write unless the keys match.

If CVAF is performing key verification, and the DSCB key does not match the data set name you supply, CVAF ignores any specified BFLEARG and writes the DSCB using the rules described in the following section, “Specifying a Data Set Name to Read or Write a DSCB.”

Specifying a Data Set Name to Read or Write a DSCB

To read or write a single DSCB by specifying only the data set name (that is, BFLEARG is zero), specify either ACCESS=READ or ACCESS=WRITE.

Specify the address of the data set name in the DSN keyword. Specify the address of the buffer list in the BUFLIST keyword. Each of these areas and the associated buffers must be in your program's protect key.

The buffer list must contain at least one buffer list entry with the skip bit off and a pointer to a 96-byte buffer. Do not use 140-byte buffers. You can chain buffer lists together, but CVAF uses only the first eligible entry.

For an indexed VTOC, CVAF searches the index for the data set name. If the data set name is found, the DSCB argument is put into the buffer list entry and used to read or write the DSCB. If the data set name cannot be found in the index, CVAF performs a key search of the VTOC.

For a nonindexed VTOC, CVAF uses a channel program to perform a key search of the VTOC to locate the data set name and read or write the DSCBs. If the data set name is found, CVAF puts the DSCB argument into the buffer list entry.

The DSCB argument returned in the buffer list entry is in the format determined by the BFLECHR or BFLETTR bits in the buffer list entry.

If CVAF does not find the data set name in the VTOC, a return code of 4 is indicated in register 15, and an error code of 1 in the CVSTAT field.

Specifying the DSCB Location

To read or write a DSCB by specifying the DSCB's location (that is, BFLEARG), specify either ACCESS=READ or ACCESS=WRITE.

Specify the address of the data set name in the DSN keyword and the address of the buffer list in the BUFLIST keyword. Each of these areas and the associated buffers must be in your program's protect key.

The buffer list must have a buffer list entry with the skip bit off and a pointer to a 96-byte or 140-byte buffer. You can chain buffer lists together, but CVAF uses only the first eligible entry.

If the buffer is for a 140-byte read or write, CVAF issues a channel program to read or write the DSCB at the location specified in the buffer list entry. CVAF ignores the specified data set name. If you specify VERIFY=YES, CVAF verifies that the designated DSCB is a format-0 DSCB before issuing the write channel program.

Releasing Buffers and Buffer Lists Obtained by CVAF

You can release buffers and buffer lists acquired by CVAF in the following ways:

- Issue a CVAF call with ACCESS=RLSE, and specify a buffer list address with the BUFLIST keyword.
- Free a MAP records buffer list by coding MAPRCDS=NO or MAPRCDS=(NO,*addr*) and specifying any ACCESS.
- Free an index records buffer list by coding IXRCDS=NOKEEP or IXRCDS=(NOKEEP,*addr*) and specifying any ACCESS.

CVAF frees all eligible buffers and any buffer lists that become empty. Eligible buffers are those pointed to by buffer list entries with the skip bit off. CVAF frees a buffer list if none of its buffer list entries have the skip bit on. If buffer lists are chained together, CVAF checks and frees all appropriate buffer lists. Do not request CVAF to release the same buffer list twice by specifying its address in more than one place.

Accessing DSNs or DSCBs in Sequential Order

You can use the CVAFSEQ macro to request the return of:

- One format-1 or format-4 DSCB in indexed (data set name) order.
- One or more DSCBs in physical-sequential order. (If you are unauthorized, you can only request one DSCB.)
- The next data set name in the index.

CVAF reads the DSCBs into buffers identified by the BUFLIST keyword. See “CVAFSEQ Macro Overview and Specification” on page 347 for additional information about the macro.

Use the buffer list to specify the argument of each DSCB to be read. For indexed access, request 96-byte DSCBs in the buffer list. For physical-sequential access, request 140-byte DSCBs.

If you select indexed order, CVAF returns each format-1 or format-4 DSCB pointed to by the index. To return only the data set names in the index (not the DSCBs), specify DSNONLY=YES. CVAF updates the DSN area specified in the CVAFSEQ

macro with the data set name of each DSCB read, every time you issue CVAFSEQ. CVAF also returns the CCHHR of the DSCB in the argument area supplied with the ARG keyword.

Initiating Indexed Access (DSN Order)

To initiate indexed access (DSN order), either supply 44 bytes of binary zeros in the DSN area (to indicate the first data set name in the index) or specify the data set name that is the starting point for the index search.

The name returned in the DSN area is equal to or greater than the DSN supplied, depending on the ACCESS keyword selection.

If you specify DSNONLY=NO, CVAF returns the DSCB and argument using the buffer list provided with the BUFLIST keyword. CVAF uses the first entry in the buffer list with the skip bit set to 0 and a nonzero buffer address. You can specify the argument format by setting either the TTR or CCHHR bit in the buffer list entry to 1. If neither bit is set, CVAF returns a CCHHR argument. For indexed access, the DSCB size in the buffer list entry must be 96 bytes.

If you specify DSNONLY=YES, specify the CCHHR argument in the ARG area.

The data set name of the format-4 DSCB is in the index and CVAF might return its name (44 bytes of `X'04'`). The format-4 DSCB's name is likely to be the first data set name in the VTOC index.

Initiating Physical-Sequential Access

To initiate physical-sequential access, either specify DSN=0, or do not specify the DSN parameter at all. To begin the read, initialize the argument field in the first buffer list entry to zero or to the argument of the DSCB. If the argument is zero, CVAF uses the argument of the start of the VTOC.

The ACCESS keyword determines whether CVAF reads the DSCB whose argument is supplied or the DSCB following it. For example, to read the first DSCB (the format-4 DSCB) in the VTOC, you can set the BFLEARG in the first buffer list entry to zero and specify ACCESS=GTEQ in the CVAFSEQ macro. If you subsequently specify ACCESS=GT, CVAF reads the second DSCB (the first format-5 DSCB). Set the DSCB size to 140 in buffer list entries.

If your program is authorized, CVAF reads as many DSCBs as there are entries in the buffer list for a single CVAF call; if it is not authorized, CVAF reads only one DSCB.

CVAF uses one buffer list and does not inspect a second buffer list chained from the first. If your program is authorized, CVAF uses all entries in the buffer list; if it is not authorized, CVAF uses only the first entry. CVAF does not inspect the skip bit. Each entry must contain a buffer address and have the length field set to 140. CVAF updates the argument field of each buffer list entry with the argument of the DSCB. You can specify the argument format by setting either the TTR or CCHHR bit in the buffer list entry to 1. If neither bit is set, CVAF returns a CCHHR argument.

CVAF uses only the argument in the first entry to begin the search and does not inspect arguments in subsequent entries. If you specify a nonzero argument value in the first entry, there must be a DSCB with that argument.

CVAF indicates an end-of-data condition by providing return code 4 in register 15, and a value of X'20' in the CVSTAT field. CVAF sets the argument fields of all buffer list entries following the last DSCB read, to zero (the first entry is zero if CVAF does not read any DSCBs).

CVAF reads all DSCBs, including format-0 DSCBs. You cannot be certain that you have read all DSCBs until CVAF has read the entire VTOC. For a nonindexed VTOC, the DS4HPCHR field of the format-4 DSCB contains the CCHHR of the last format-1 DSCB. DSCBs other than format-1 can reside beyond that location. For an indexed VTOC, the VMDS contains information about which DSCBs are format-0 DSCBs.

Reading Sets of DSCBs with CVAF Filter

You can use the CVAFFILT macro to retrieve sets of DSCBs into buffers provided by the calling program. CVAF filter service supports both indexed and nonindexed VTOCs. “CVAFFILT Macro Overview and Specification” on page 337 describes the macro's format and parameters. The following section summarizes this service and its requirements.

- Request DSCBs by specifying either one or more fully-qualified data set names, or one partially-qualified name. See “Filter Criteria List (FCL)” on page 50 and “Partially-Qualified Names for CVAFFILT” on page 342 for further information.
- Identify a single DASD volume in the CVPL.
- For each of the qualifying data sets, CVAF filter returns DSCBs in the order that they are chained in the VTOC: format-1, format-2, then format-3. CVAF does not return DSCBs of other formats.
- CVAF filter service returns DSCB information for one or more qualifying data sets into caller-provided buffers. See “Example of CVAFFILT Macro Sequences” on page 53 and “Example of Using the CVAFFILT Macro” on page 342 for further information. CVAF filter service does not return a partial DSCB chain:
  - If you do not provide enough buffers to hold the requested DSCBs, CVAF filter service returns one or more complete DSCB chains and/or a status code (CVSTAT in the CVPL). The status code indicates if a RESUME CVAF call can be used to retrieve the rest (or more) of the DSCBs. See “RESUME Capability” for specific information.
  - If the total number of buffers is insufficient to contain a data set's complete DSCB chain, CVAF filter service sets the FCLDSNST byte in the FCL, ignores the data set, and processes the next qualifying data set. To avoid this situation, provide a minimum of eleven DSCB buffers (enough for a data set at the 123 extent limit).

RESUME Capability

If CVAF filter service terminates because you failed to provide sufficient buffers, the information necessary for a RESUME function is saved in the filter save area. (Specifying FLTAREA=KEEP on the initial CVAFFILT allocates the filter save area.)

To allow RESUME processing to execute correctly, you must maintain the relationship between the requested volume (identified by CVDEB, CVUCB, or a kept IOAREA), your FCL, and CVAF's FSA. If you observe this requirement, you can initiate and resume multiple CVAF filter service operations asynchronously on one

or more DASD volumes. You can ensure this relationship by providing a unique CVPL and FCL for the duration of the READ/RESUME/RELEASE sequence associated with each logical request.

Issuing an ACCESS=RESUME without having previously specified FLTAREA=KEEP causes CVAF filter service to produce return code 4 in register 15 and 66 (decimal) in the CVSTAT field.

If you specify FLTAREA=KEEP, issue a subsequent CVAFFILT call with the ACCESS=RLSE keyword to release filter save area storage.

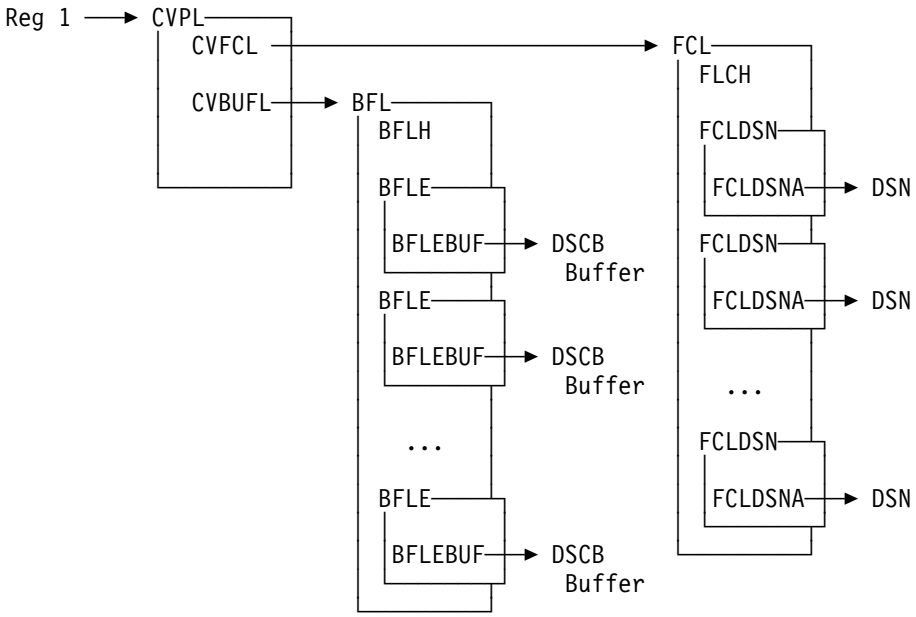


Figure 23. Control Blocks Required for CVAF Filter Services

Filter Criteria List (FCL)

The filter criteria list (FCL) consists of a list header and a variable number of list entries. The list entries follow the header, and each entry represents a data set name to be processed by CVAF filter. The header and entries, shown in Figure 24 and Figure 25 on page 52, are mapped by the ICVFCL macro. The format of the FCL header is shown in Figure 24.

Figure 24 (Page 1 of 2). Format of a Filter Criteria List Header

Name	Offset	Bytes	Description
FCLID	0 (X'00')	4	EBCDIC 'FCL*bb*' (*bb* here represents a blank.)
FCLCOUNT	4 (X'04')	2	Number of data set name entries provided in the list.
FCLDSCBR	6 (X'06')	2	Number of DSCBs returned.
FCL1FLAG	8 (X'08')	1	Request flag byte.
FCL1LIST		1	List contains fully-qualified data set names.
FCL1ORDR		. 1	FCL data set name order requested.
FCL1EQF1		. . 1	Return format-1 DSCBs only.
		. . . x xxxx	Reserved.

Figure 24 (Page 2 of 2). Format of a Filter Criteria List Header

Name	Offset	Bytes	Description
FCL2FLAG	9 (X'09')	1	Status flag byte.
FCL2SEQ		1	CVAFFILT executed sequential VTOC access.
FCL2SDIR		. 1	CVAFFILT executed sequential VTOC access, but did at least one direct DSCB read.
		. . xx xxxx	Reserved.
FCLDRSV	10 (X'0A')	6	Reserved.

FCLID
Must be a 4-character EBCDIC constant of `'FCL`*bb*`'`. (*bb* here represents a blank.)

FCLCOUNT
Specifies the number of data set name entries (FCLDSN) supplied in the list. Do not change this parameter between the initial CVAFFILT call and any subsequent RESUME operations.

- If you specify a partially-qualified data set name, specify FCLCOUNT = 1. See “Partially-Qualified Names for CVAFFILT” on page 342 for the format of partially-qualified data set names.
- If you specify a list of fully-qualified names, CVAFFILT processes only the number of names specified in FCLCOUNT.

FCLDSCBR
Indicates the total number of DSCB entries (including format-1, format-2, and format-3) returned to the caller's buffers by a single CVAFFILT call.

If CVAF encounters an error after successfully processing a data set, you can:

1. Initialize FCLDSCBR to 0 before each READ and RESUME call.
2. Upon return from CVAF filter service, process the number of DSCBs indicated by FCLDSCBR.
3. Then, interpret the CVAF return code and CVSTAT.

FCL1FLAG
Define your request for ACCESS=READ with this flag byte. Any subsequent RESUME requests refer to a copy of these bits in the filter save area (FSA).

FCL1LIST
If you specify a list of fully-qualified data set names, set this bit to 1. If you specify a single partially-qualified data set name, set this bit to 0.

FCL1ORDR
If you specify that CVAF is to return DSCB chains in the data set name sequence implied by the placement of the FCLDSN elements, set this bit to 1. Note that:

- It can improve performance to allow CVAF to determine the sequence of return for format-1 DSCBs.

- CVAF returns DSCBs for a given data set in format-1, format-2, format-3 order.
- If you specify a single partially-qualified data set name, this field is not used.

FCL1EQF1
To have CVAF return only the format-1 DSCBs for the data set names, set this bit to 1.

FCL2FLAG
CVAF filter indicates the following status conditions in this byte.

FCL2SEQ
This bit is set to 1 if a sequential VTOC access path is most efficient. If CVAF filter selects the direct VTOC access path, it sets this field to 0.

FCL2SDIR
This bit is set to 1 if storage limitations within the sequential VTOC access path require direct DSCB reads. CVAF initializes this bit to 0 on each ACCESS=READ and ACCESS=RESUME request. Testing this bit when CVAF filter returns control can indicate if you need to change the storage limitation.

The format of the FCL entry is shown in Figure 25.

Figure 25. Format of a Filter Criteria List Entry

Name	Offset	Bytes	Description
FCLDSN		8	Data set name information entry.
FCLDSNST	0 (X'00')	1	Data set name status.
		X'00'	Data set name not yet processed.
		X'01'	DSCBs returned successfully.
		X'02'	Data set name not found.
		X'03'	Error in DSCB chain. RESUME function recommended.
		X'04'	Error in CVAFFILT processing. RESUME not recommended.
		X'05'	Insufficient user buffer list elements. RESUME function recommended.
FCLDSNLG	1 (X'01')	1	Data set name length.
FCL3FLAG	2 (X'02')	1	Flag byte.
FCL3UPDT		1	This data set name processed during this invocation.
		. xxx xxxx	Reserved.
FCLDSNRV	3 (X'03')	1	Reserved.
FCLDSNA	4 (X'04')	4	Data set name address.

FCLDSN
Contains data set name information. This and the following fields are repeated in the FCL as a set as many times as indicated by the value in FCLCOUNT.

FCLDSNST
Indicates DSCB retrieval status.

- CVAF filter initializes this byte to 0 for ACCESS=READ requests.

- After processing the data set name for either ACCESS=READ or ACCESS=RESUME, CVAF filter updates this byte.
- ACCESS=RESUME requests do not process data set names whose FCLDSNST field is nonzero; therefore, results can be unpredictable if you alter this field.
- For partially-qualified data set name requests, CVAF filter does not post the FCLDSNST field until it has returned all DSCB chains for all qualifying data sets. CVAF filter posts the highest numeric value that applied during its processing.
- For fully-qualified data set name requests, CVAF filter returns a FCLDSNST byte for each data set name. If the value is greater than 1, CVAF filter has not returned any DSCBs for the associated data set name.

See Figure 25 on page 52 for an explanation of the values in this field.

FCLDSNLG
Indicates the length of the data set name. This value is required.

FCL3FLAG
The status flag byte associated with the data set name pointed to by FCLDSNA.

FCL3UPDT
This bit indicates that CVAF filter processed the associated data set name during the current invocation of CVAFFILT.

- When initializing for either a READ or RESUME request, CVAF filter sets this bit to 0.
- When CVAF filter has completed processing for the associated data set name, it sets this bit to 1.

FCL3DSNRV
Reserved, unused.

FCLDSNA
Specifies the address of a fully-qualified data set name, or, if this is the only data set name and FCL1LIST is 0, a partially-qualified data set name. You must provide both this address and the storage area to which it points.

Example of CVAFFILT Macro Sequences

The following example demonstrates the order that you might issue CVAFFILT macro calls to:

1. Request the DSCBs for a list of data sets.
2. Resume CVAFFILT processing interrupted because of insufficient user buffers.
3. Release the kept filter save area.

The example assumes the following conditions:

- You are an authorized caller (that is, you are specifying a UCB address and IOAREA=KEEP).
- You have initialized a CVAF buffer list with the following characteristics:
  - Four buffers

  - The buffer list address in your program has the label BUFADDR
  - The same buffer list is used for ACCESS=READ and ACCESS=RESUME processing.
- You have initialized a filter criteria list as follows:
  - FCLCOUNT = 6 (You are requesting DSCB chains for six data set names.)
  - FCL1LIST = '1'B (The data set names are fully qualified.)
  - FCL1ORDR = '1'B (You want the DSCB chains returned in the order implied by data set name elements in the FCL.)
  - The six data set name elements are initialized so that they form a list requesting SYS1.A, SYS2.B, SYS3.C, SYS4.D, SYS5.E, and SYS6.F.
- The first five data sets have DSCB chain lengths or 1, 5, 2, 3, and 1, respectively, on the volume.
- The sixth data set (SYS6.F) is not defined on the volume.

To obtain an initialized CVPL, you could issue the following CVAFFILT macro (list form—does not call CVAF). This example requests the branch entry to CVAF and specifies that the caller is in supervisor state.

```
CVPLIST  CVAFFILT BRANCH=(YES,SUP),MF=L
```

To obtain the first set of DSCB chains, you could issue the following CVAFFILT macro (execute form—calls CVAF). This example specifies that the filter save area is to be kept to allow for ACCESS=RESUME calls. The IOAREA is to be kept for improved efficiency.

```
CVAFFILT ACCESS=READ,BUFLIST=bufaddr,FCL=fcladdr,
         UCB=ucbaddr,FLTAREA=KEEP,IOAREA=KEEP,
         MF=(E,CVPLIST)
```

This CVAFFILT call returns the following DSCBs:

```
Buffer    Contents of Buffer

   1      Format-1 DSCB, SYS1.A
   2      Format-1 DSCB, SYS3.C
   3      Format-3 DSCB, SYS3.C
   4      Undefined (unused)
```

CVAF filter produces return code = 4, CVSTAT = X'40' (RESUME recommended), and FCLDSCBR = 3. (CVAF returns a total of three DSCBs for the two data sets.) CVAF would not return DSCBs for data set SYS2.B because its chain contains more DSCBs than the total number of buffers provided. To retrieve the DSCBs for SYS2.B, you need to specify at least five buffers and execute another ACCESS=READ. (Even though CVAF allows you to specify a different buffer list for each READ or RESUME, or to modify the existing list between READ and RESUME calls, modifying the FCL would cause unpredictable results.) Buffer entry 4 does not have any DSCBs returned, because SYS4.D's DSCB chain size is larger than the number of remaining buffers. The FCL status information would be as follows:

```
DSN      FCLDSNST   FCL3UPDT   Comments
SYS1.A       1          1      DSCBs returned from this call
SYS2.B       5          1      DSCB chain exceeds total buffers
SYS3.C       1          1      DSCBs returned from this call
SYS4.D       0          0      DSCBs can be returned by RESUME
SYS5.E       0          0      DSCBs can be returned by RESUME
SYS6.F       0          0      DSCBs can be returned by RESUME
```

Because this CVAFFILT invocation recommends RESUME, and you specified FLTAREA=KEEP, you could use the following execute form of CVAFFILT to obtain more DSCB chains:

```
CVAFFILT ACCESS=RESUME,MF=(E,CVPLIST)
```

This CVAFFILT call returns DSCBs as follows:

```
Buffer   Contents of Buffer
   1     Format-1 DSCB, SYS4.D
   2     Format-2 DSCB, SYS4.D
   3     Format-3 DSCB, SYS4.D
   4     Format-1 DSCB, SYS5.E
```

CVAF filter produces return code = 0, CVSTAT = 0 (request completed), and updates the FCL status as follows:

```
DSN      FCLDSNST   FCL3UPDT   Comments
SYS1.A       1          0      DSCBs returned from prior call
SYS2.B       5          0      DSCB chain exceeds total buffers
SYS3.C       1          0      DSCBs returned from prior call
SYS4.D       1          1      DSCBs returned from this call
SYS5.E       1          1      DSCBs returned from this call
SYS6.F       2          1      Data set name not found
```

FCLDSCBR would contain 4. (This CVAFFILT call returns a total of four DSCBs.) CVAF filter does not return any DSCBs for SYS6.F, because its format-1 DSCB cannot be found on the volume (FCLDSNST = `'2'`).

Because this status indicates that CVAF filter has returned all requested DSCBs, and you requested FLTAREA=KEEP and IOAREA=KEEP on the previous call, request the RLSE function as follows:

```
CVAFFILT ACCESS=RLSE,FLTAREA=NOKEEP,IOAREA=NOKEEP,
         MF=(E,CVPLIST)
```

VTOC Error Responses

These actions are taken if an error occurs:

- If an index structure error is detected and if the address space is enqueued on the VTOC, then DADSM or CVAF disables the VTOC index. The indexed VTOC bit is zeroed in the format-4 DSCB, a software error record is written to SYS1.LOGREC, and a system dump is taken at the next call to DADSM create or extend. The VTOC is converted to a nonindexed format at the next DADSM create or extend call.
- If a program check, machine check, or other error occurs while using a VTOC access macro, a SYS1.LOGREC message is written, and a system dump is taken.

- An error code is put in the CVSTAT field of the CVPL. The values and explanations of these error codes are listed in “VTOC Index Error Message and Associated Codes” on page 360.

Recovering from System or User Errors

Because an unauthorized user cannot modify a VTOC, neither the VTOC nor the VTOC index need be recovered from an error caused by an unauthorized user.

A system error can affect a VTOC and VTOC index by interrupting DADSM while it is updating, leaving the VTOC or the VTOC index (or both) in a partially-updated state. Both the VTOC and the VTOC index allow DADSM to recover from such an interruption.

For a nonindexed VTOC (or a VTOC with an index that has been disabled), a subsequent call to DADSM create or extend, causes VTOC convert routines to reestablish the free space DSCB chain.

For an indexed VTOC, a subsequent call to any DADSM function causes the recovery of the previous interrupt (either by backing out or completing the interrupted function).

GTF Trace

A trace function exists to trace all CVAF calls for VTOC index output I/O, all VTOC output I/O, and all VTOC index and space map modifications. For information on this function, see *DFSMS/MVS DFSMSdfp Diagnosis Reference*.

VTOC and VTOC Index Listings

You can obtain dump, formatted, or abridged listings of the VTOC and the VTOC index by using the LISTVTOC command of the IEHLIST utility program. The DFSMSdss print command also provides VTOC and index listing options. The ISMF data set application displays information about the data sets represented in a VTOC.

Chapter 2. Managing the Volume Table of Contents

This chapter discusses the programs and procedures used to create and manage a volume table of contents (VTOC).

Creating the VTOC and VTOC Index

To prepare a volume for activity by initializing it, use the Device Support Facilities (ICKDSF) utility to build the VTOC. You can create a VTOC index when initializing a volume by using the ICKDSF INIT command and specifying the INDEX keyword.

Convert a nonindexed VTOC to an indexed VTOC by using the BUILDIX command and specifying the IXVTOC keyword. The reverse operation can be performed by using the BUILDIX command and specifying the OSVTOC keyword. See *ICKDSF R16 User's Guide* for details.

Protecting the VTOC and VTOC Index

Use the following methods to protect the VTOC and VTOC index:

RACF

You can protect the VTOC and VTOC index using the Resource Access Control Facility (RACF) by defining the volume serial entity under the RACF DASDVOL class. For you to modify a protected VTOC and VTOC index, programs must be authorized at the following levels:

- UPDATE level to open for output processing a VTOC or any data set with a name beginning with SYS1.VTOCIX.
- ALTER level to allocate, rename, or scratch any data set with a name beginning with SYS1.VTOCIX or to rename a data set to a name beginning with SYS1.VTOCIX.

Neither the VTOC nor the VTOC index is protected from being opened for input processing by the DASDVOL/volume serial entity. Neither the VTOC nor the VTOC index can be protected through the RACF DATASET class. For additional information on using RACF, see *OS/390 Security Server (RACF) General User's Guide*.

APF

The APF must authorize a program in order for you to:

- Open a VTOC for output processing
- Open for output processing, allocate, rename, or scratch any data set whose name begins with SYS1.VTOCIX
- Rename a data set to a name that begins with SYS1.VTOCIX.

For additional information on using APF, see *OS/390 MVS Authorized Assembler Services Guide*.

© Copyright IBM Corp. 1979, 1999

Password Protection

The VTOC index data set can be password protected. Protection is the same as for a password-protected data set. Password checking is bypassed if the volume containing the VTOC index is protected by RACF via the DASDVOL class. For additional information, see Chapter 6, “Using Password Protected Data Sets” on page 147.

Copying/Restoring/Initializing the VTOC

The following sections discuss VTOC considerations when updating volumes.

Volumes Containing a Nonindexed VTOC

Restoring a Volume from a Dump Tape: There are no operational requirements if you change the volume serial number or do a partial restore that does not modify the VTOC. If you do a restore and change the VTOC size without changing the volume serial number, you must vary the volume offline after it is restored. Do not try to restore on a volume with an indexed VTOC.
Copying a Volume: There are no operational requirements if you change the volume serial number or do not modify the VTOC of the receiving volume. If you do a copy and change the VTOC size without changing the volume serial number, you must vary the volume offline after it is copied. Do not attempt to copy from a volume with an indexed VTOC.
Shared DASD Considerations: In shared DASD environments, if the VTOC index is relocated or the volume is changed from indexed VTOC to OS VTOC or from OS VTOC to indexed VTOC, you must vary the device offline to the sharing system or systems before beginning the operation.

Volumes Containing an Indexed VTOC

Use Device Support Facilities to convert a VTOC to a nonindexed format to update the volume. If you do not, take note of the following information:

Initializing a Volume: If you do not change the volume serial number, you must vary the volume offline before starting the job.
Restoring a Volume from a Dump Tape: There are no operational requirements if you change the volume serial number or do a partial restore that does not modify the VTOC or VTOC index. If you do a restore, and modify the VTOC or VTOC index without changing the volume serial number, you must vary the volume offline after it is restored.
Copying a Volume: There are no operational requirements if you change the volume serial number of the receiving volume or do a partial dump without modifying the VTOC or VTOC index. If you modify the VTOC or VTOC index without changing the volume serial number, you must vary the receiving volume offline after it is copied.

Accessing the VTOC with RENAME and SCRATCH

You can use the RENAME and SCRATCH DADSM macros to access and modify VTOC and VTOC index information about a data set.

Deleting a Data Set from the VTOC

You can use the SCRATCH and CAMLST macro to delete a non-VSAM data set or a temporary VSAM data set. SCRATCH processing makes the space occupied by the data set available for reallocation. This process does not automatically erase data from the disk. See “Erasing Sensitive Data” for further information.

Specifying the Volumes Affected

When deleting a data set, build a volume list in virtual storage. The volume list consists of an entry for each volume on which the data set resides. If you are deleting an SMS-managed data set, specify at least one SMS-managed volume in the list. The first two bytes of the list indicate the number of entries in the list. Each 12-byte entry consists of a 4-byte device code (the UCBTYP field from the volume's UCB), a 6-byte volume serial number, and 2 bytes of scratch status information consisting of a secondary status code and a status code, both of which must be initialized to zero.

Volumes are processed according to their order in the volume list. If a volume is not mounted, a message is issued to the operator requesting that the volume be mounted. This only occurs when you indicate the direct access device on which unmounted volumes are to be mounted by loading register 0 with the address of the UCB of the device. (The device must be allocated to your job.) **If you do not load register 0 with a UCB address, its contents must be zero**, and at least one of the volumes in the volume list must be mounted before the SCRATCH macro instruction is issued. Use the address of a UCB, not a UCB copy, in register 0 with this macro.

If the requested volume cannot be mounted, the operator replies by indicating that the request cannot be fulfilled. A status code is then set in the last byte of the volume list entry (the second byte of the scratch status code) for the unavailable volume, and the next volume in the volume list is processed.

Erasing Sensitive Data

You should erase data sets that contain sensitive data by overwriting them with zeros before their space is made available. This can either be done before issuing the SCRATCH macro, or be requested in scratch processing by performing one of the following:

- Providing an associated RACF profile ERASE attribute
- Activating bit 21 (X'00 00 04 00') of the SCRATCH parameter list.

Authorized callers of SCRATCH can prevent erasure of the data by setting bit 22 to 1, which overrides the RACF profile ERASE attribute. For information on RACF-defined data sets and volumes, see *OS/390 Security Server (RACF) Introduction*.

System-Managed-Storage Considerations

SMS screens all data set SCRATCH requests. If the volumes in your volume list are SMS managed, SMS does a catalog LOCATE to determine the actual volume serial numbers, and deletes the data set from all volumes on which it resides. SMS coordinates the required changes to the VTOC, the VTOC index, and the catalog.

If DADSM encounters a processing error when SMS is active and all the volumes in your list are SMS managed, SMS determines the volume on which the failure

occurred. The first entry in your list will be overlaid with the entry for the volume on which the request failed.

You might find that a volume indicated as being in error was not specified in the volume list your program provided. This occurs if the volumes in your list are different from the volumes in the data set's catalog entry.

If SMS is not active, you cannot delete SMS-managed data sets.

You can delete SMS-managed VSAM data sets using the access method services DELETE command. See *DFSMS/MVS Access Method Services for ICF* for further information.

General Considerations and Restrictions

A data set cannot be deleted if the expiration date in the format-1 DSCB has not passed unless you override the expiration date. You can request SCRATCH to ignore the expiration date by specifying the OVRD option in the CAMLST macro instruction. SCRATCH processing supports three never-scratch dates. To prevent a data set from being scratched, specify one of the following expiration dates:

1999.365
1999.366
1999.999

To delete a virtual input/output (VIO) data set, the data set must be allocated for use by your job step.

You cannot use the SCRATCH macro with either a SYSIN or SYSOUT data set.

SCRATCH does not support HFS files. You will receive unpredictable results if you use SCRATCH for HFS files.

If you attempt to delete a password-protected data set that is not also RACF protected, the operating system issues message IEC301A to the operator at the console, or the terminal operator of a TSO console, to enter the password. The data set will be scratched if the password supplied is associated with a WRITE protection mode indicator. The protection mode indicator is described in Chapter 6, “Using Password Protected Data Sets” on page 147. See *OS/390 Security Server (RACF) Introduction* for information on RACF-defined data sets and volumes.

If a data set is RACF defined (indicated in its format-1 DSCB or described by a RACF profile) or the volume upon which it resides is RACF defined, only a program with ALTER access authority to either the data set/volume serial in the DATASET class or to the volume serial in the DASDVOL class can scratch the data set.

Use the STOW macro to delete or rename a member of a PDS or PDSE. STOW is described in *DFSMS/MVS Macro Instructions for Data Sets* and *DFSMS/MVS Using Data Sets*. You can also use the IEHPROGM utility to delete a member (see *DFSMS/MVS Utilities*).

Note: For an SMS-managed non-VSAM data set, you need RACF authority to the data set or to the catalog to delete it.

For a non-VSAM data set that is not SMS managed, DADSM invokes RACF to verify authorization. If you have ALTER access authority to the data set/volume serial in the DATASET class, DADSM deletes the data set from the volume. If you

have ALTER authority to the data set/volume serial in the DATASET class, or UPDATE access authority to the catalog/volume serial in the DATASET class, you can delete the catalog entry.

SCRATCH and CAMLST Macro Specification

The format of the SCRATCH and CAMLST macros is:

```
►►──┬──────────┬──SCRATCH──listname_addrx──────────────────────────────►◄
    └──label───┘
```

```
►►──listname──CAMLST──SCRATCH──,dsname_relexp──,,vol_list_relexp──────►
►──┬──────────┬──────────────────────────────────────────────────────►◄
   └──,,OVRD───┘
```

listname_addrx
: points to the parameter list (labeled listname) set up by the CAMLST macro instruction.

SCRATCH
: code this operand as shown.

dsname_relexp
: specifies the virtual storage location of a fully-qualified data set name. The area that contains the name must be 44 bytes long.

vol list_relexp
: specifies the virtual storage location of an area that contains a volume list. The area must begin on a halfword boundary.

OVRD
: when coded as shown, specifies that the expiration date in the DSCB should be ignored.

Example

In the following example, data set A.B.C is deleted from two volumes. The expiration date in the identifier (format-1) DSCB is ignored.

```
         SR       0,0              SHOW NO UCB IS SUPPLIED
         SCRATCH  DELABC           DELETE DATA SET A.B.C
*                                  FROM TWO VOLUMES,
*                                  IGNORING EXPIRATION
*                                  DATE IN THE DSCB

DELABC   CAMLST   SCRATCH,DSABC,,VOLIST,,OVRD
DSABC    DC       CL44'A.B.C'      DATA SET NAME
VOLIST   DC       H'2'             NUMBER OF VOLUMES
         DC       X'3030200E'      3380 DISK DEVICE CODE
         DC       CL6'000017'      VOLUME SERIAL NO.
         DC       H'0'             SCRATCH STATUS CODE
         DC       X'3030200E'      3380 DISK DEVICE CODE
         DC       CL6'000018'      VOLUME SERIAL NO.
         DC       H'0'             SCRATCH STATUS CODE
```

Note: Check the return codes and SCRATCH status codes.

The SCRATCH macro instruction points to the CAMLST macro instruction. The SCRATCH operand specifies that a data set be deleted. DSABC specifies the virtual storage location of a 44-byte area containing the fully-qualified name of the data set to be deleted. VOLIST specifies the virtual storage location of the volume list you have built. OVRD specifies that the expiration date in the DSCB of the data set to be deleted should be ignored.

Return Codes from SCRATCH

Control returns to the instruction following the instructions generated by the SCRATCH macro. Register 15 contains the SCRATCH return code as shown in Figure 26.

SCRATCH returns 4 bytes of diagnostic information in register 0. If an error occurs, DADSM issues message IEC614I, consisting of failure-related information including the return code and the 4 bytes of diagnostic information. See *DFSMS/MVS DFSMSdfp Diagnosis Reference* for a description of this information.

The last two bytes of a volume list entry contain the secondary status code and the scratch status code. The secondary status codes are shown in Figure 28 on page 64 and the scratch status codes are shown in Figure 27 on page 63. To determine if the data set has been deleted from each volume, check the scratch status code. (Even if the scratch status code is zero, the secondary status code might be nonzero for the first entry in the volume list.)

Figure 26 describes the conditions indicated by the SCRATCH return code.

Figure 26 (Page 1 of 2). SCRATCH Return Codes

Return Code	Description
000 (X'00')	If the 4 bytes of diagnostic information returned in register 0 are all zeros, the data set was successfully deleted. If they are nonzero, use the SCRATCH diagnostic information tables in *DFSMS/MVS DFSMSdfp Diagnosis Reference* to determine the failure-related conditions.
004 (X'04')	No volume containing any part of the data set was mounted. The data set might be a VIO data set that was not allocated to your jobstep.

Figure 26 (Page 2 of 2). SCRATCH Return Codes

Return Code	Description
008 (X'08')	An unusual condition was encountered on one or more volumes.
012 (X'0C')	One of the following conditions occurred: • The SCRATCH parameter list is not valid. • The volume list is not valid. • At entry to SCRATCH, register 0 was not zero and did not point to a valid UCB. The address must be that of a UCB, not a UCB copy. The SCRATCH status code will not have been set.

Status Codes from SCRATCH

After the SCRATCH macro instruction is executed (for SCRATCH return codes 0,4,and 8 only), the last byte of each 12-byte entry in the volume list indicates one of the following conditions:

Figure 27 (Page 1 of 2). SCRATCH Status Codes

Status Code	Meaning
0 (X'00')	The data set has been deleted from this volume.
1 (X'01')	The VTOC of this volume does not contain the format-1 DSCB to be deleted.
2 (X'02')	One of the following conditions occurred: • The data set could not be scratched because the correct password was not specified in the two attempts allowed. • The user tried to scratch a VSAM data space or an integrated catalog facility VSAM data set. • The user tried to scratch the VTOC index data set. • An SMS-validation failure occurred. • The verify of the last referenced date failed.
3 (X'03')	The data set was not deleted because either the OVRD option was not specified or the retention cycle had not expired.
4 (X'04')	One of the following conditions occurred: • An invalid format-1 DSCB was encountered when processing this volume. • An unexpected CVAF error return code was encountered. • An installation exit rejected the request. • An I/O error occurred while the DASD tracks occupied by the data set were being erased. Either the ERASE option was specified in the scratch parameter list or the ERASE attribute was specified for an RACF-defined data set.
5 (X'05')	It could not be verified that this volume was mounted, nor was there a unit available for mounting the volume.
6 (X'06')	The operator was unable to mount this volume.
7 (X'07')	The data set was not deleted because it was open.

Figure 27 (Page 2 of 2). SCRATCH Status Codes

Status Code	Meaning
8 (X'08')	The format-1 DSCB indicates the data set is defined to RACF, but either you are not authorized to the data set or volume, or the data set is a VSAM data space.

After the SCRATCH macro instruction is executed, the next to last byte of the first entry in the volume list indicates one of the following conditions:

Figure 28. Secondary Status Codes

Status Code	Meaning
0 (X'00')	No secondary status for this volume.
128 (X'80')	The data set was RACF protected and the calling program was authorized by the RACF DATASET class to scratch the data set. This means that at least one volume entry was protected.

Renaming a Data Set in the VTOC

You can use the RENAME and CAMLST macro to rename a non-VSAM data set. Rename processing causes the data set name in all format-1 DSCBs to be replaced with the new name you supply. The new data set name must conform to standard data set naming conventions.

Specifying the Volumes Affected

When renaming a data set, build a volume list in virtual storage. The volume list consists of an entry for each volume on which the data set resides. If you are renaming an SMS-managed data set, specify at least one SMS-managed volume in the list. The first two bytes of the list indicate the number of entries in the list. Each 12-byte entry consists of a 4-byte device code (the UCBTYP field from the volume's UCB), a 6-byte volume serial number, and a 2-byte rename status code that should be initialized to zero.

Volumes are processed according to their order in the volume list. If a volume is not mounted, a message is issued to the operator requesting that the volume be mounted. This only occurs when you indicate the direct access device on which unmounted volumes are to be mounted by loading register 0 with the address of the UCB of the device. (The device must be allocated to your job.) **If you do not load register 0 with a UCB address, its contents must be zero**, and at least one of the volumes in the volume list must be mounted before the RENAME macro instruction is issued. Use the address of a UCB, not a UCB copy, in register 0 with this macro.

If the requested volume cannot be mounted, the operator replies by indicating that the request cannot be fulfilled. A status code is then set in the last byte of the volume list entry (the second byte of the rename status code) for the unavailable volume, and the next volume indicated in the volume list is processed.

System-Managed-Storage Considerations

SMS screens all data set RENAME requests. If the volumes specified in your volume list are SMS managed, SMS does a catalog LOCATE to determine the actual volume serial numbers, and coordinates the required changes to the VTOC, the VTOC index, and the catalog.

If DADSM encounters a processing error while SMS is active and all the volumes in your list are SMS managed, SMS determines the volume on which the failure occurred. The first entry in your list will be overlaid with the entry for the volume on which the request failed.

You might find that a volume indicated as being in error was not specified in the volume list your program provided. This occurs if the volumes in your list are different from the volumes in the data set's catalog entry.

If SMS is not active, you cannot rename SMS-managed data sets.

General Considerations and Restrictions

To rename a data set that is stored on more than one volume, either a device must be available for mounting the volumes, or at least one volume must be mounted. In addition, all other volumes of the data set must be serially mountable.

You cannot use the RENAME macro with either a SYSIN or SYSOUT data set.

RENAME does not support HFS files. You will receive unpredictable results if you use RENAME for HFS files.

Only a program with ALTER access authority to the data set in the DATASET class can rename an RACF-defined data set. For information on RACF-defined data sets, see *OS/390 Security Server (RACF) Introduction*.

If you attempt to rename a password-protected data set, the operating system issues message IEC301A asking the operator or TSO operator to verify the password. The data set will be renamed if the password supplied is associated with a WRITE protection mode indicator. The protection mode indicator is described in Chapter 6, “Using Password Protected Data Sets” on page 147.

You cannot rename VIO data sets.

RENAME and CAMLST Macro Specification

The format of the RENAME and CAMLST macros is:

```
►►──┬──────────┬──RENAME──listname_addrx──────────────────────────────►◄
    └─label────┘
```

```
►►──listname──CAMLST──RENAME──,dsname_relexp──,new name_relexp──────────►
►──,vol list_relexp──────────────────────────────────────────────────────►◄
```

listname_addrx
points to the parameter list (labeled listname) set up by the CAMLST macro instruction.

RENAME
code this operand as shown.

dsname_relexp
specifies the virtual storage location of a fully-qualified data set name to be replaced. The area containing the name must be 44 bytes long.

new name_relexp
specifies the virtual storage location of a fully-qualified data set name that is to be used as the new name. The area containing the name must be 44 bytes long.

vol list_relexp
specifies the virtual storage location of an area that contains a volume list. The area must begin on a halfword boundary.

Example

In the following example, data set A.B.C is renamed D.E.F. The data set resides on two volumes.

```
         SR      0,0             SET REG 0 TO ZERO
         RENAME  DSABC           CHANGE DATA SET
                                 NAME A.B.C TO D.E.F

DSABC    CAMLST  RENAME,OLDNAME,NEWNAME,VOLIST
OLDNAME  DC      CL44'A.B.C'     OLD DATA SET NAME
NEWNAME  DC      CL44'D.E.F'     NEW DATA SET NAME
VOLIST   DC      H'2'            TWO VOLUMES
         DC      X'3030200E'     3380 DISK DEVICE CODE
         DC      CL6'000017'     VOLUME SERIAL NO.
         DC      H'0'            RENAME STATUS CODE
         DC      X'3030200E'     3380 DISK DEVICE CODE
         DC      CL6'000018'     VOLUME SERIAL NO.
         DC      H'0'            RENAME STATUS CODE
```

Note: Check the return codes and RENAME status codes.

The RENAME macro instruction points to the CAMLST macro instruction. The RENAME operand specifies that a data set be renamed. OLDNAME specifies the virtual storage location of a 44-byte area where you have placed the fully-qualified name of the data set to be renamed. NEWNAME specifies the virtual storage location of a 44-byte area where you have placed the new name of the data set. VOLIST specifies the virtual storage location of the volume list you have built.

Return Codes from RENAME

Control returns to the instruction following the instructions generated by the RENAME macro. Register 15 contains the DADSM return code as shown in Figure 29 on page 67.

DADSM RENAME returns 4 bytes of diagnostic information in register 0. If an error occurs, DADSM issues message IEC614I, consisting of failure-related information including the return code and the 4 bytes of diagnostic information. See *DFSMS/MVS DFSMSdfp Diagnosis Reference* for a description of this information.

Each volume's volume list entry contains the rename status code as shown in Figure 30 on page 67. To determine whether the data set has been successfully renamed on each volume, check the rename status code, contained in the last byte of each entry in the volume list.

Figure 29 describes the conditions indicated by the DADSM return code.

Figure 29. DADSM RENAME Return Codes

Return Code	Description
000 (X'00')	If the 4 bytes of diagnostic information returned in register 0 are all zeros, the data set has been successfully renamed. If they are nonzero, use the DADSM RENAME diagnostic information tables in *DFSMS/MVS DFSMSdfp Diagnosis Reference* to determine the failure-related conditions.
004 (X'04')	No volume containing any part of the data set was mounted. The data set can be a VIO data set but cannot be renamed.
008 (X'08')	An unusual condition was encountered on one or more volumes. The diagnostic information is in register 0. Use the DADSM RENAME diagnostic information tables in *DFSMS/MVS DFSMSdfp Diagnosis Reference* to determine the failure-related conditions.
012 (X'0C')	One of the following conditions occurred: • The DADSM RENAME parameter list is not valid. • The volume list is not valid. • At entry to RENAME, register 0 was not zero and did not point to a valid UCB. The address must be that of a UCB, not a UCB copy. The RENAME status code will not have been set.

Status Codes from RENAME

After the RENAME macro instruction is executed (for RENAME return codes 0,4,and 8 only), the last byte of each 12-byte entry in the volume list indicates one of the conditions described in Figure 30.

Figure 30 (Page 1 of 2). RENAME Status Codes

Status Code	Meaning
0 (X'00')	The format-1 DSCB for the data set has been renamed in the VTOC on this volume.
1 (X'01')	The VTOC of this volume does not contain the format-1 DSCB of the data set to be renamed.
2 (X'02')	One of the following conditions occurred: • The data set could not be renamed because the data set was password protected and the password was not supplied in the two attempts allowed. • An attempt was made to rename a VSAM data space or an integrated catalog facility VSAM data set. • An attempt was made to rename a VTOC index data set. • An SMS-validation failure occurred.

Figure 30 (Page 2 of 2). RENAME Status Codes

Status Code	Meaning
3 (X'03')	A format-1 DSCB containing the new data set name already exists in the VTOC of this volume, or an attempt was made to rename a data set to a name starting with SYS1.VTOCIX.
4 (X'04')	One of the following conditions occurred: • A permanent I/O error occurred while trying to rename the data set on this volume. • An invalid format-1 DSCB was encountered while processing this volume. • No space is available in the index VIER for the new name, and no additional VIERs are available.
5 (X'05')	It could not be verified that this volume was mounted nor was a unit available for mounting the volume.
6 (X'06')	The operator was unable to mount this volume.
7 (X'07')	The data set was not renamed, because it was currently open for processing.
8 (X'08')	The data set is defined to RACF, but either you are not authorized to the data set or the data set is defined to RACF on multiple volumes.

Chapter 3. Using Catalog Management Macros

This chapter describes catalog management macro instructions only for compatibility purposes. IBM doesn't recommend using CVOLs because integrated catalog facility (ICF) catalogs are faster and more reliable than CVOLs. IBM strongly recommends that you convert to ICF catalogs. Catalog management macro instructions can be used to:

- Retrieve information from an integrated catalog facility catalog, a VSAM catalog, or a CVOL.
- Catalog, uncatalog, or recatalog non-VSAM data sets in an integrated catalog facility catalog, a VSAM catalog, or a CVOL. You cannot catalog, recatalog, or uncatalog SMS-managed non-VSAM data sets.
- Read a block from a CVOL.
- Build an index or generation index in a CVOL.
- Assign an alias to a high-level index in a CVOL.
- Delete an index or index alias from a CVOL.
- Connect or disconnect two CVOLs.

Application Program Considerations

A catalog management request can be satisfied in either a CVOL, VSAM catalog, or integrated catalog facility catalog. Restrictions and limitations to consider in relation to your application programs consist of:

- The data set name SYSCTLG is restricted and required for all CVOLs.
- CVOL volume serials are honored in CAMLSTs, and direct a catalog management request to the specified CVOL. In this case, a master catalog CVOL pointer of the form SYSCTLG.V*volser* must be defined as described in *DFSMS/MVS Managing Catalogs*.
- CVOLs to be accessed or allocated cannot be defined using JOBCAT or STEPCAT DD statements.
- A CVOL (SYSCTLG data set) cannot be scratched while a job or TSO session that opened the CVOL is active.
- A catalog management request is expressed in a parameter list pointed to by register 1. Generate the parameter list with a CAMLST macro. The CAMLST and its associated fields must not be located in read-only storage and must reside below the 16MB line.
- Register 15 contains the return code. If it is a CVOL request that was satisfied in an integrated catalog facility or VSAM catalog, register 0 contains the integrated catalog facility or VSAM catalog management return code. These return codes are explained in *OS/390 MVS System Messages, Vol 4 (IEC-IFD)* under IDC3009I. The return code received in register 15 is referred to as the CVOL return code.

© Copyright IBM Corp. 1979, 1999

Catalog Search Order

Catalogs are searched for entries using the following methods:

1. If a catalog is specified in a macro, only that catalog is searched.
2. User catalogs specified in the current job step with a STEPCAT DD statement are searched in order of concatenation. Once the entry is found, no other catalog is searched.
   - If a STEPCAT catalog is specified and the entry is not found, the JOBCAT catalog is not searched. The catalog search continues with step 3.
   - If a STEPCAT catalog is not specified for the job step, and user catalogs are specified for the current job with a JOBCAT DD statement, the JOBCAT catalogs are searched in order of concatenation. Once the entry is found, no other catalog is searched.
3. If the entry is identified with a qualified entry name and its first qualifier is the same as the name or alias of a user catalog, or the alias of a CVOL, the user catalog or CVOL is searched. Once the entry is found, no other catalog is searched.
4. The master catalog is searched.

See *DFSMS/MVS Access Method Services for ICF* for more detailed information.

Retrieving Information from a Catalog

To read an entry from a catalog, use the LOCATE and CAMLST macro instructions. You can specify the entry to be read into your output area using:

- The fully- or partially-qualified name of a data set
- The relative block address of the block within a CVOL containing the entry.

If you specify a fully-qualified data set name, a list of volumes on which the data set resides is read into your output area. This volume list always begins with a 2-byte entry indicating the number of volumes in the list. If the data set resides on more than 20 volumes and is cataloged in an CVOL, the address of a volume control block follows the volume list entries.

Note: When CAMLST is used to locate a data set that is over 20 volumes in length, only information from the first 20 volumes is returned.

If you specify a partially-qualified data set name, the first block in the CVOL pointed to by the lowest-level index specified will be read into your output area. This occurs when you specify two or more qualifiers, or the *cvol_relexp* parameter in the CAMLST macro. Register 15 will contain return code 12. If you specify a single qualifier and do not include the *cvol_relexp* parameter, the CVOL identifier `'SYSCTLG.Vyyyyyy'` is read into your output area (the area previously occupied by the data set name). You can then insert `'yyyyyy'` as the *cvol_relexp* parameter in the CAMLST and reissue the LOCATE.

If you specify a relative track address, by using a TTR (track, track, record) address, the block at that relative address in the CVOL catalog is read into your output area.

When specifying either an unqualified name or the highest level of a partially-qualified name to retrieve information from a CVOL, you receive the volume information for the CVOL found in the master catalog. The single qualifier name that you specified is replaced by the SYSCTLG.Vyyyyyy name. You can use that information to specify the CVOL volume serial number in CAMLST so that the search starts in the CVOL and gives you the desired information.

For the Catalog interface, a fully-qualified name is one which represents a single data set. A partially-qualified name is one which may contain multiple qualifiers, but does not specify a full data set name.

For example, if LEVEL1.LEVEL2.LEVEL3.LEVEL4 is a data set, then LEVEL1.LEVEL2.LEVEL3.LEVEL4 is a fully-qualified name. The following would be considered partially-qualified names:

LEVEL1.LEVEL2.LEVEL3
LEVEL1.LEVEL2
LEVEL1

Note: For the catalog interface, you cannot specify an asterisk (*) or an ampersand (&) to specify a partially-qualified data set name.

See *DFSMS/MVS DFSMSdfp Diagnosis Reference* for descriptions of the contents of the volume control block and the other catalog data areas.

See “Return Codes from LOCATE” on page 77 for a description of the LOCATE return codes.

Retrieving Information by Data Set Name (LOCATE and CAMLST NAME)

Specifying a data set name returns a volume list in your output area. A volume list consists of an entry for each volume on which part of the data set resides; it begins with a 2-byte field containing the number of volumes in the list. The count field is followed by a variable number of 12-byte entries. Each 12-byte entry consists of a 4-byte device code, a 6-byte volume serial number, and a 2-byte volume sequence number. As many as 20 of these 12-byte entries can be built in your output area. LOCATE can return bytes 252 - 254 of your area containing the relative track address of the DSCB on the first or only volume for that data set. Otherwise, these bytes are zero. Bytes 242 to 251 are reserved, byte 255 contains zeros, and bytes 256 to 264 are also reserved and not intended as a programming interface.

If the data set is cataloged in a CVOL and resides on more than five volumes, the volume list in your output area is a volume control block (VCB) that has been read into your output area. In a VCB, the count field contains the number of volume entries in this VCB and any following VCBs. A count of 41, for example, indicates two following VCBs with counts of 20 and 1, respectively. The relative track address (TTR) of the next VCB is in bytes 252 - 254 of your output area. The last VCB for a data set has binary zeros in bytes 252 - 254.

A CAMLST LOCATE on a VSAM cluster returns the volumes for all components. The format of this volume list is the same as described previously. If a CAMLST LOCATE request is satisfied in a VSAM or integrated catalog facility catalog, the output is translated into CVOL format before returning to the caller. The original VSAM or integrated catalog facility return code is saved in register 0.

The macro format for the LOCATE and CAMLST NAME is:

```
►►──┬──────────┬──LOCATE──listname_addrx──────────────────────────────►◄
    └─label────┘
```

```
►►──listname──CAMLST──NAME──,dsname_relexp──,┬──────────────┬──────────►
                                             └─cvol_relexp──┘
►──,area_relexp──────────────────────────────────────────────────────►◄
```

listname_addrx
points to the parameter list (labeled listname) set up by the CAMLST macro instruction.

NAME
to retrieve information from a catalog by name, code this operand as shown.

dsname_relexp
specifies the virtual storage location of a fully-qualified data set name. The area that contains the name must be 44 bytes long. The name can be defined by a C-type define constant (DC) instruction.

cvol_relexp
specifies the virtual storage location of the 6-byte volume serial number of the CVOL to which this catalog request is directed. For a discussion of the effect of specifying this operand, see “Catalog Search Order” on page 70.

area_relexp
specifies the virtual storage location of your 265-byte output area, that you must define. The output area must begin on a doubleword boundary.

Example

In the following example, the catalog entry containing a list of the volumes on which data set A.B resides, is read into virtual storage.

```
          LOCATE    INDAB              READ CATALOG ENTRY FOR DATA SET A.B
*                                      INTO VIRTUAL STORAGE AREA NAMED LOCAREA.
*                                      LOCAREA MAY ALSO CONTAIN A 3-BYTE
*                                      TTR OR THE 6-BYTE CVOL SERIAL NUMBER
         Check Return Codes

INDAB     CAMLST    NAME,AB,,LOCAREA
AB        DC        CL44'A.B'
LOCAREA   DS        0D
          DS        265C
```

The LOCATE macro instruction points to the CAMLST macro instruction. NAME, the first operand of CAMLST, specifies that the system is to search for a catalog entry using the name of a data set. AB, the second operand, specifies the virtual storage location of the fully-qualified data set name LOCAREA, the fourth operand, specifies a 265-byte area you have reserved in virtual storage.

After these macro instructions execute, the 265-byte area contains a volume list or a volume control block for the data set A.B. If the entry has been located and read successfully, register 15 contains zeros. Otherwise, register 15 contains a return code (see “Return Codes from LOCATE” on page 77).

Retrieving Information by Generation Data Set Name (LOCATE and CAMLST NAME)

Specify the name of a generation data set using the fully-qualified generation index name and the relative generation number of the data set. The value of a relative generation number reflects the position of a data set in a generation data group. The following values can be used:

- Zero—specifies the latest data set (highest generation number) cataloged in a generation data group.
- Negative number—specifies a data set cataloged before the latest data set.

  Note: If DISP (disposition) is DELETE to make room for other data sets and no generation data group exists, the job will complete indicating a deleted generation name (G0000V00). If a generation data group exists but is not in the range specified for deletion, the step will fail.
- Positive number—specifies a data set not yet cataloged in the generation data group.

Using zero or a negative number as the relative generation number places a volume list (or a volume control block) in your output area and replaces the relative generation number with the absolute generation name.

Using a positive number as the relative generation number creates an absolute generation name and replaces the relative generation number. Because there are no entries in the catalog, zeros are read into the first 256 bytes of your output area.

The format for the LOCATE and CAMLST NAME macros is:

```
►►──┬──────────┬──LOCATE──list_addrx──────────────────────────────────►◄
    └─label────┘
```

```
►►──listname──CAMLST──NAME──,dsname_relexp──,┬──────────────┬──────────►
                                             └─cvol_relexp──┘
►──,area_relexp──────────────────────────────────────────────────────►◄
```

list_addrx
points to the parameter list (labeled listname) set up by the CAMLST macro instruction.

NAME
to read a block from the catalog by generation data set name, code this operand as shown.

dsname_relexp
specifies the virtual storage location of the name of the generation index and the relative generation number. The area that contains these must be 44 bytes long.

cvol_relexp
specifies the virtual storage location of the 6-byte volume serial number of the CVOL to which this catalog request is directed. For a discussion of the effect of specifying this operand, see “Catalog Search Order” on page 70.

area_relexp
specifies the virtual storage location of your 265-byte output area, which you must define. The output area must begin on a doubleword boundary. The output area will contain a volume list that is built from the catalog. If the data set resides on one volume, bytes 252 - 254 can contain the relative track address of the DSCB. This address is relative to the beginning of the volume.

Example

In the following example, the list of volumes containing generation data set A.PAY(-3) is read into virtual storage.

```
          LOCATE    INDGX                 READ CATALOG ENTRY
*                                         FOR DATA SET A.PAY(-3)
*                                         INTO YOUR STORAGE
*                                         AREA NAMED LOCAREA
         Check Return Codes

INDGX     CAMLST    NAME,APAY,,LOCAREA
APAY      DC        CL44'A.PAY(-3)'
LOCAREA   DS        0D
          DS        265C
```

The LOCATE macro instruction points to the CAMLST macro instruction. NAME, the first operand of CAMLST, initiates a search for a catalog entry using the name of a data set. APAY, the second operand, specifies the virtual storage location of the name of the generation index and the relative generation number of a data set in the generation data group. LOCAREA, the fourth operand, specifies a 265-byte area you have reserved to receive the catalog information.

After executing this macro instruction, the system replaces the relative generation number that you specified with the data set's absolute generation name. Control is returned to your program at the next executable instruction following the LOCATE macro instruction. If the entry has been located and read successfully, register 15 contains zeros. Otherwise, register 15 contains a return code (see “Return Codes from LOCATE” on page 77). See “Retrieving Information by Data Set Name (LOCATE and CAMLST NAME)” on page 71 for a description of the contents of the output area.

Retrieving Information by Alias (LOCATE and CAMLST NAME)

For each of the preceding functions, you can specify an alias as the name of a data set. Functions proceed as previously described with one exception: the true name replaces the specified alias name.

Note: Aliases are not allowed for generation data sets cataloged in CVOLs.

The format for the LOCATE and CAMLST NAME macros is:

```
►►──┬──────────┬──LOCATE──list_addrx──────────────────────────────────►◄
    └──label───┘
```

```
►►──listname──CAMLST──NAME──,dsname_relexp──,┬──────────────┬──────────►
                                             └──cvol_relexp─┘
►──,area_relexp──────────────────────────────────────────────────────►◄
```

list_addrx
points to the parameter list (labeled listname) set up by the CAMLST macro instruction.

NAME
to retrieve information from a catalog, code this operand as shown.

dsname_relexp
specifies the virtual storage location of a fully-qualified data set name, the first or only name of which is the alias. The area containing the name must be 44 bytes long. The name can be defined by a C-type DC instruction.

cvol_relexp
specifies the virtual storage location of the 6-byte volume serial number of the CVOL to which this catalog request is directed. For a discussion of the effect of specifying this operand, see “Catalog Search Order” on page 70.

area_relexp
specifies the virtual storage location of your 265-byte output area, that you must define. The output area must begin on a doubleword boundary. The first 256 bytes of the output area will contain a volume list that is read from a catalog. If the data set resides on one volume, bytes 252 - 254 can contain the relative track address of the DSCB. This address is relative to the beginning of the volume.

Example

In the following example, the catalog entry containing a list of the volumes on which data set A.B.C resides is read into virtual storage (data set A.B.C, however, is addressed by an alias name, X.B.C).

```
          LOCATE    INDAB                READ CATALOG ENTRY
*                                        FOR DATA SET X.B.C
*                                        INTO VIRTUAL STORAGE
*                                        AREA NAMED LOCAREA.
          Check Return Codes

INDAB     CAMLST    NAME,ABC,,LOCAREA
ABC       DC        CL44'X.B.C'
LOCAREA   DS        0D
          DS        265C
```

The LOCATE macro instruction points to the CAMLST macro instruction. NAME, the first operand of CAMLST, initiates a search of the catalog for an entry using the name of a data set. ABC, the second operand, specifies the virtual storage location of the fully-qualified name of a data set (in this case, data set A.B.C is addressed by its alias X.B.C). LOCAREA, the fourth operand, specifies a 265-byte area you have reserved in virtual storage.

See “Return Codes from LOCATE” on page 77 for a description of the LOCATE return codes.

Reading a Block by Relative Block Address (LOCATE and CAMLST BLOCK)

You can read a block in a CVOL by specifying the identification of the block and its location relative to the beginning of the catalog.

The format for the LOCATE and CAMLST BLOCK macros is:

```
►►──┬──────────┬──LOCATE──list_addrx──────────────────────────────►◄
    └──label───┘
```

```
►►──listname──CAMLST──BLOCK──,ttr_relexp──,cvol_relexp──,area_relexp──►◄
```

list_addrx
: points to the parameter list (labeled listname) set up by the CAMLST macro instruction.

BLOCK
: code this operand as shown.

ttr_relexp
: specifies the virtual storage location of a 3-byte relative block address (TTR). This address indicates the position of the track containing the block (TT), and the block identification (R) on that track, relative to the beginning of the catalog data set.

cvol_relexp
: specifies the virtual storage location of a 6-byte volume serial number for the volume to be processed.

area_relexp
: specifies the virtual storage location of your 265-byte output area, that you must define. The output area must begin on a doubleword boundary. The output area will contain the block that is read from the catalog, and the last six bytes of the area will contain the serial number of the volume on which the block was found. If the data set resides on one volume, bytes 252 - 254 will contain the relative track address of the DSCB.

Example

In the following example, the block at the location indicated by TTR is read into virtual storage.

```
          LOCATE   BLK

         Check Return Codes

BLK       CAMLST   BLOCK,TTR,VOLSER,LOCAREA
*                                   READ A BLOCK INTO
*                                   VIRTUAL STORAGE AREA
TTR       DC       H'5'             RELATIVE TRACK 5
          DC       X'03'            BLOCK 3 ON TRACK
VOLSER    DC       C'111111'        VOLUME SERIAL OF CVOL
LOCAREA   DS       0D               NAMED LOCAREA
          DS       265C             LOCAREA ALSO CONTAINS
*                                   6-BYTE SERIAL NO.
```

The LOCATE macro instruction points to the CAMLST macro instruction. BLOCK, the first operand of CAMLST, initiates a search of the catalog for the block indicated by TTR, the second operand. VOLSER, the third operand, specifies the virtual storage location of a 6-byte volume serial number for the volume to be processed. LOCAREA, the fourth operand, specifies a 256-byte area you have reserved in virtual storage.

After these macro instructions execute, the 265-byte area contains the 256-byte block and the 6-byte serial number of the volume on which the block was found (in bytes 259 through 264).

Control is returned to your program at the next executable instruction following the LOCATE macro instruction. Register 15 contains one of the return codes described under “Return Codes from LOCATE.”

Return Codes from LOCATE

Control is returned to your program at the next executable instruction following the LOCATE macro instruction. If the information has been successfully read from the catalog, the register contains zeros. Otherwise, register 15 might contain one of the following return codes. See Figure 31 for the LOCATE return codes.

Note: Instead of one of these codes, you might get one of the codes listed in Figure 34 on page 94.

Figure 31 (Page 1 of 2). LOCATE Return Codes

Code	Meaning
0 (X'00')	Operation successful.
4 (X'04')	Either the required catalog does not exist or it cannot be opened or there is a closed chain of pointers in blocks in a CVOL.

Figure 31 (Page 2 of 2). LOCATE Return Codes

Code	Meaning
8 (X'08')	One of the following happened: • The entry was not found. If in a CVOL, register 0 contains the number of valid index levels. If in an integrated catalog facility or a VSAM catalog, register 0 contains the catalog return code. • The user is not authorized to perform this operation. Register 0 contains hexadecimal 38. • A generation data group (GDG) alias was found. Register 0 contains the number of valid index levels. The alias name was replaced by the true name. • A generation data group base entry was found in an integrated catalog facility or VSAM catalog.
12 (X'0C')	One of the following happened: • An index or generation data group base entry was found in a CVOL when the list of qualified names was exhausted. Register 0 contains the number of valid index levels. The output area contains the first block of the specified index. • An alias entry was found in a CVOL. The alias name was replaced in the user parameter list by the true name. • An invalid low-level GDG name was found.
16 (X'10')	A data set exists at other than the lowest index level specified. Register 0 contains the number of the index level where the data set was encountered.
20 (X'14')	Syntax error. A blank was found in the first byte of the name field.
24 (X'18')	One of the following happened: • Permanent I/O error occurred. Register 0 contains the VSAM or integrated catalog facility return code, or 0 if in a CVOL. • Nonzero ESTAE return code. • Error in parameter list.
28 (X'1C')	Relative track address supplied to LOCATE routine is outside of the SYSCTLG data set extents.
32 (X'20')	Reserved.
38 (X'26')	DFSMShsm LOCATE preprocessor has experienced an error.

Note: See *OS/390 MVS System Messages, Vol 4 (IEC-IFD)*, message IDC3009I, for documentation of integrated catalog facility catalog and VSAM catalog return codes.

Building and Deleting Indexes

You handle CVOL indexes (build them, delete them, etc.) by using combinations of the INDEX and CAMLST macro instructions.

Building an Index (INDEX and CAMLST BLDX)

To build a new CVOL index structure and add it to the catalog, you can create each level of the index separately. (You can also create index levels while you are cataloging a data set onto those index levels. To create each level of the index, use the INDEX and CAMLST macro instructions.) These two macro instructions can also be used to add index levels to existing index structures.

The format for the INDEX and CAMLST BLDX macros is:

```
►►──┬──────────┬──INDEX──list_addrx──────────────────────────────────►◄
    └──label───┘
```

```
►►──listname──CAMLST──BLDX──,namerelexp──┬──────────────┬──────────►◄
                                         └──,cvol_relexp──┘
```

list_addrx
: points to the parameter list (labeled listname) set up by the CAMLST macro instruction.

BLDX
: code this operand as shown.

namerelexp
: specifies the virtual storage location of the fully-qualified name of a data set or index level. The name cannot exceed 44 characters. If the name is less than 44 characters, it must be followed by at least one blank.

cvol_relexp
: specifies the virtual storage location of a 6-byte volume serial number of the CVOL to which this catalog request is directed. For a discussion of the effect of specifying this operand, see “Catalog Search Order” on page 70.

Example

In the following example, index structure A.B.C is built on the CVOL with the serial number 000045. Each INDEX macro instruction points to an associated CAMLST macro instruction. BLDX, the first operand of CAMLST, specifies that an index level be built. The second operand specifies the virtual storage location where you have placed the fully-qualified name of an index level. The third operand specifies the virtual storage location containing the 6-byte serial number of the volume on which the index level is to be built.

```
          INDEX    INDEXA          BUILD INDEX A

         Check Return Codes

          INDEX    INDEXB          BUILD INDEX STRUCTURE
*                                  A.B

         Check Return Codes

          INDEX    INDEXC          BUILD INDEX STRUCTURE
*                                  A.B.C

         Check Return Codes

INDEXA    CAMLST   BLDX,ALEVEL,VOLNUM
INDEXB    CAMLST   BLDX,BLEVEL,VOLNUM
INDEXC    CAMLST   BLDX,CLEVEL,VOLNUM
VOLNUM    DC       CL6'000045'     VOLUME SERIAL NUMBER
ALEVEL    DC       CL2'A'          INDEX STRUCTURE NAMES
BLEVEL    DC       CL4'A.B'        FOLLOWED BY A BLANK
CLEVEL    DC       CL6'A.B.C'      WHICH DELIMITS FIELDS
```

Register 15 contains zeros if the operation was successful, otherwise register 15 contains one of the return codes described under “Return Codes from INDEX” on page 87.

Building a Generation Index (INDEX and CAMLST BLDG)

Use the INDEX and CAMLST macro instructions to build a generation index in a CVOL. All higher levels of the index must exist. If the higher levels of the index are not in the catalog, you must build them. Procedures for building an index are explained in “Building an Index (INDEX and CAMLST BLDX)” on page 78.

The format for the INDEX and CAMLST BLDG is:

```
►►──┬──────────┬──INDEX──list_addrx──────────────────────────────────►◄
    └─label────┘
```

```
►►──listname──CAMLST──BLDG──,namerelexp──,──┬──────────────┬──────────►
                                            └─cvol_relexp──┘
►──,,──┬──────────┬──,──┬──────────┬──,number_absexp──────────────────►◄
       └─DELETE───┘     └─EMPTY────┘
```

list_addrx
: points to the parameter list (labeled listname) set up by the CAMLST macro instruction.

BLDG
: code this operand as shown.

namerelexp
specifies the virtual storage location of the fully-qualified name of a data set or index level. The name cannot exceed 44 characters. If the name is less than 44 characters, it must be followed by at least one blank.

cvol_relexp
specifies the virtual storage location of a 6-byte volume serial number of the CVOL to which this catalog request is directed. For a discussion of the effect of specifying this operand, see “Catalog Search Order” on page 70.

DELETE
specifies that all data sets on direct access volumes that are removed from a generation data group are to be deleted, that is, the space allocated to the data sets is to be made available for reallocation. A SCRATCH macro instruction will be issued by the catalog management routines, and the data set will be deleted from the volume if there are no conditions preventing deletion (for example, expiration date not passed, password not verified, volume not mounted, permanent I/O error encountered while trying to delete the data set).

EMPTY
specifies that references to all data sets in a generation data group cataloged in the generation index are to be removed from the index when the number of entries specified is exceeded.

number_absexp
specifies the number of data sets to be included in a generation data group. This number must be specified, and cannot exceed 255.

Example

In the following example, generation index D is built on the CVOL, serial number 000045. The higher-level indexes A.B.C already exist. When the number of generation data sets in the generation index D exceeds four, the oldest data set is uncataloged. When the DELETE operand has been specified and the data set has been successfully uncataloged, the catalog management routines issue a SCRATCH macro to delete the data set. If there are no conditions preventing it (for example, the expiration date was not passed, the password could not be verified, or a permanent I/O error was encountered when trying to delete the data set), the data set will be deleted.

```
          INDEX    GENINDX         BUILD GENERATION INDEX

         Check Return Codes

GENINDX    CAMLST   BLDG,DLEVEL,VOLNUM,,DELETE,,4
DLEVEL     DC       CL8'A.B.C.D '   ONE BLANK, DELIMITER
VOLNUM     DC       CL6'000045'
```

The INDEX macro instruction points to the CAMLST macro instruction. BLDG, the first operand of CAMLST, specifies that a generation index is to be built. DLEVEL, the second operand, specifies the virtual storage location of the fully-qualified name of a generation index. VOLNUM, the third operand, specifies the virtual storage location of the 6-byte serial number of the volume on which the generation index is to be built. DELETE, the fifth operand, specifies that all data sets dropped from the generation data group are to be deleted. The final operand, 4, specifies the number of data sets that are to be maintained in the generation data group. Control is

returned to your program at the next executable instruction following the INDEX macro instruction. If the generation index was built successfully, register 15 contains zeros. Otherwise, register 15 contains one of the return codes described under “Return Codes from INDEX” on page 87.

Deleting an Index (INDEX and CAMLST DLTX)

You can delete any number of index levels from an existing CVOL index structure, but each level of the index must be deleted separately. If an index level has an alias, other index levels, or data sets cataloged under it, it cannot be deleted. Use the INDEX and CAMLST macro instructions to delete each level of the index. Generation indexes are also removed the same way. (You can also delete index levels automatically when you uncatalog a data set.)

The format for the INDEX and CAMLST DLTX is:

```
►►──┬──────────┬──INDEX──list_addrx──────────────────────────────────►◄
    └─,label───┘
```

```
►►──listname──CAMLST──DLTX──,namerelexp──┬──────────────┬──────────────►◄
                                         └─,cvol_relexp─┘
```

list_addrx
points to the parameter list (labeled listname) set up by the CAMLST macro instruction.

DLTX
code this operand as shown.

namerelexp
specifies the virtual storage location of the fully-qualified name of a data set or index level. The name cannot exceed 44 characters. If the name is less than 44 characters, it must be followed by at least one blank.

cvol_relexp
specifies the virtual storage location of a 6-byte volume serial number of the CVOL to which this catalog request is directed. For a discussion of the effect of specifying this operand, see “Catalog Search Order” on page 70.

Example

In the following example, index level C is deleted from index structure A.B.C.

```
          INDEX    DELETE          DELETE INDEX LEVEL C FROM INDEX STRUCTURE
*                                  A.B.C

         Check Return Codes

DELETE    CAMLST   DLTX,LEVELC
LEVELC    DC       CL6'A.B.C'      ONE BLANK FOR
*                                  DELIMITER
```

The INDEX macro instruction points to the CAMLST macro instruction. DLTX, the first operand of CAMLST, specifies that an index level is to be deleted. LEVELC, the second operand, specifies the virtual storage location of the fully-qualified name of the index structure whose lowest level is to be deleted. Control is returned to your program at the next executable instruction following the INDEX macro instruction. Register 15 contains one of the return codes described under “Return Codes from INDEX” on page 87.

Assigning an Alias for an Index (INDEX and CAMLST BLDA)

For CVOLs, assign an alias to an index level using the INDEX and CAMLST macro instructions. An alias can be assigned only to a high-level index; for example, index A of index structure A.B.C can have an alias, but index B cannot. Assigning an alias to a high-level index effectively provides aliases for all data sets cataloged under that index. An alias cannot be assigned to a generation index.

The format for the INDEX and CAMLST BLDA is:

```
►►──┬──────────┬──INDEX──list_addrx──────────────────────────────────►◄
    └─label────┘
```

```
►►──listname──CAMLST──BLDA──,index_namerelexp──,┬──────────────────────►
                                                └─cvol_relexp──────────┘
►──,alias_namerelexp──────────────────────────────────────────────────►◄
```

list_addrx
: points to the parameter list (labeled listname) set up by the CAMLST macro instruction.

BLDA
: code this operand as shown.

index namerelexp
: specifies the virtual storage location of the name of a high-level index. The area that contains the name must be 8 bytes long.

cvol_relexp
: specifies the virtual storage location of a 6-byte volume serial number of the CVOL catalog to which this catalog request is directed. For a discussion of the effect of specifying this operand, see “Catalog Search Order” on page 70.

alias namerelexp
: specifies the virtual storage location of the name to be an alias for a high-level index. The area that contains the name must be 8 bytes long. The name can be defined by a C-type DC instruction.

Example

In the following example, high-level index A is assigned an alias of X.

```
          INDEX      ALIAS              BUILD AN ALIAS FOR
*                                       A HIGH LEVEL INDEX

         Check Return Codes

ALIAS     CAMLST     BLDA,DSNAME,,DSALIAS
DSNAME    DC         CL8'A'             MUST BE 8-BYTE FIELDS
DSALIAS   DC         CL8'X'
```

The INDEX macro instruction points to the CAMLST macro instruction. BLDA, the first operand of CAMLST, specifies that an alias is to be built. DSNAME, the second operand, specifies the virtual storage location of the name of the high-level index to be assigned an alias. DSALIAS, the fourth operand, specifies the virtual storage location of the alias to be assigned.

Control is returned to your program at the next executable instruction following the INDEX macro instruction. Register 15 contains one of the return codes described under “Return Codes from INDEX” on page 87.

Deleting an Alias for an Index (INDEX and CAMLST DLTA)

For CVOLs, you can delete an alias previously assigned to a high-level index using the INDEX and CAMLST macro instructions.

The format for the INDEX and CAMLST DLTA macros is:

```
►►──┬──────────┬──INDEX──list_addrx──────────────────────────────────►◄
    └─label────┘
```

```
►►──listname──CAMLST──DLTA──,alias_namerelexp──┬──────────────────┬──►◄
                                               └─,cvol_relexp─────┘
```

list_addrx
: points to the parameter list (labeled listname) set up by the CAMLST macro instruction.

DLTA
: code this operand as shown.

alias namerelexp
: specifies the virtual storage location of the name that is used as an alias for a high-level index. The area that contains the name must be 8 bytes long.

cvol_relexp
: specifies the virtual storage location of a 6-byte volume serial number of the CVOL catalog to which this catalog request is directed. For a discussion of the effect of specifying this operand, see “Catalog Search Order” on page 70.

Example

In the following example, alias X, previously assigned as an alias for index level A, is deleted.

```
          INDEX    DELALIAS            DELETE AN ALIAS FOR
*                                     A HIGH LEVEL INDEX

          Check Return Codes

DELALIAS   CAMLST   DLTA,ALIAS
ALIAS      DC       CL8'X'              MUST BE 8-BYTE FIELD
```

The INDEX macro instruction points to the CAMLST macro instruction. DLTA, the first operand of CAMLST, specifies that an alias is to be deleted. ALIAS, the second operand, specifies the virtual storage location of the alias to be deleted.

Connecting and Disconnecting CVOLs

Connect and disconnect CVOLs using combinations of the INDEX and CAMLST macro instructions.

Connecting CVOLs (INDEX and CAMLST LNKX)

Use the INDEX and CAMLST macro instructions to connect two CVOLs.

Supply the serial number of the volume to be connected and the high-level index name used to associate the two volumes. If the index name is an alias of a CVOL pointer entry in the master catalog, then you can omit the serial number of the “from” volume. Otherwise, supply the serial numbers of both volumes and the name of a high-level index associated with the volume to be connected.

As a result of connecting CVOLs, the volume serial number of the CVOL connected and the name of a high-level index are entered into the volume index of the volume to which it was connected. This entry is called a control-volume pointer.

The format for the INDEX and CAMLST LNKX is:

```
►►──┬──────────┬──INDEX──list_addrx──────────────────────────────────►◄
    └──label───┘
```

```
►►──listname──CAMLST──LNKX──,index_namerelexp──,┬──────────────┬──────►
                                                └──cvol_relexp─┘
►──,new_cvol_relexp──────────────────────────────────────────────────►◄
```

list_addrx
: points to the parameter list (labeled listname) set up by the CAMLST macro instruction.

LNKX
code this operand as shown.

index namerelexp
specifies the virtual storage location of the name of a high-level index. The area that contains the name must be 8 bytes long.

cvol_relexp
specifies the virtual storage location of a 6-byte volume serial number of the CVOL catalog to which this catalog request is directed. For a discussion of the effect of specifying this operand, see “Catalog Search Order” on page 70.

new cvol_relexp
specifies the virtual storage location of the 4-byte device code and 6-byte volume serial number of the control volume that is to be connected to another CVOL.

Example

In the following example, the CVOL with serial number 001555 is connected to the CVOL numbered 000155. The name of the high-level index is HIGHINDX.

```
          INDEX    CONNECT            CONNECT TWO CVOLS
*                                     WHOSE SERIAL NUMBERS ARE
*                                     000155 and 001555.
*                                     3380 DISK DEVICE CODE
          Check Return Codes

CONNECT   CAMLST   LNKX,INDXNAME,OLDCVOL,NEWCVOL
*
INDXNAME  DC       CL8'HIGHINDX'
OLDCVOL   DC       CL6'000155'
NEWCVOL   DC       X'3010200E'
          DC       CL6'001555'
```

The INDEX macro instruction points to the CAMLST macro instruction. LNKX, the first operand of CAMLST, specifies that control volumes are to be connected. INDXNAME, the second operand, specifies the virtual storage location of the name of the high-level index of the volume to be connected. OLDCVOL, the third operand, specifies the virtual storage location of the serial number of the volume to which you are connecting.

NEWCVOL, the fourth operand, specifies the virtual storage location of the volume to be connected and the volume serial number.

Control is returned to your program at the next executable instruction following the INDEX macro instruction. Register 15 contains one of the return codes described under “Return Codes from INDEX” on page 87.

Disconnecting CVOLs (INDEX and CAMLST DRPX)

Use the INDEX and CAMLST macro instructions to disconnect two CVOLs. As a result of disconnecting CVOLs the CVOL pointer is removed from the volume index of the volume from which you are disconnecting.

The format for the INDEX and CAMLST DRPX is:

```
►►──┬──────────┬──INDEX──list_addrx──────────────────────────────────►◄
    └──label───┘
```

```
►►──listname──CAMLST──DRPX──,index_namerelexp──┬──────────────────┬──►◄
                                               └──,cvol_relexp────┘
```

list_addrx
points to the parameter list (labeled listname) set up by the CAMLST macro instruction.

DRPX
code this operand as shown.

index namerelexp
specifies the virtual storage location of the name of a high-level index. The area that contains the name must be 8 bytes long.

cvol_relexp
specifies the virtual storage location of a 6-byte volume serial number of the CVOL catalog to which this catalog request is directed. For a discussion of the effect of specifying this operand, see “Catalog Search Order” on page 70.

Example

In the following example, the CVOL containing the high-level index HIGHINDX is disconnected from the CVOL pointed to by the entry 'HIGHINDX' in the master catalog.

```
           INDEX    DISCNECT               DISCONNECT TWO CVOLS

          Check Return Codes

DISCNECT   CAMLST   DRPX,INDXNAME
INDXNAME   DC       CL8'HIGHINDX'          MUST BE 8-BYTE FIELD
```

The INDEX macro instruction points to the CAMLST macro instruction. DRPX, the first operand of CAMLST, specifies that CVOLs are to be disconnected. INDXNAME, the second operand, specifies the virtual storage location of the name of the high-level index of the CVOL to be disconnected.

Control is returned to your program at the next executable instruction following the INDEX macro instruction. Register 15 contains one of the return codes described under “Return Codes from INDEX.”

Return Codes from INDEX

Control is returned to your program at the next executable instruction following the INDEX macro instruction. Register 15 might contain one of the following return codes.

Note: Instead of one of these codes, you might get one of the codes listed in Figure 34 on page 94. < <

Figure 32. INDEX Return Codes

Code	Meaning
0 (X'00')	Operation successful.
4 (X'04')	The CVOL does not exist or cannot be opened.
8 (X'08')	One of the following happened: • The existing catalog structure is inconsistent with the operation requested. If the error was detected while processing in a CVOL, register 0 has the number of valid index levels and register 1 has the return code that would have resulted if a LOCATE macro had been issued on the same entry name. If the error was detected during the master catalog search process, register 0 contains the catalog return code and register 1 contains zero. • The user is not authorized to perform the operation. Register 0 contains 56 (decimal); register 1 contains 0.
12 (X'0C')	An attempt was made to build an index or generation index that has an alias or has indexes or data sets cataloged under it. The index is unchanged.
16 (X'10')	The qualified name specified when building an index or generation index implies an index structure that does not exist; the high-level index, specified when connecting control volumes, does not exist.
20 (X'14')	Space is not available on the specified CVOL.
24 (X'18')	Reserved.
28 (X'1C')	A permanent I/O error was found when processing the catalog, or a nonzero return code from ESTAE or GETMAIN was encountered.

Using Non-VSAM Data Set Catalog Entries

You can catalog, uncatalog, and recatalog non-VSAM data sets in CVOLs, integrated catalog facility catalogs, and VSAM catalogs using the CATALOG and CAMLST macro instructions. CATALOG macro instructions are used to point to CAMLST macro instructions and to specify cataloging options.

For a description of the search algorithms used for cataloging, uncataloging, and recataloging non-VSAM data sets in an integrated catalog facility or VSAM catalogs, see the DEFINE and the DELETE commands in *DFSMS/MVS Access Method Services for ICF* .

Cataloging a Non-VSAM Data Set (CATALOG and CAMLST CAT)

The format of the CATALOG and CAMLST macros is:

```
►►──┬──────────┬──CATALOG──list_addrx──────────────────────────────────►◄
    └──label───┘
```

```
►►──listname──CAMLST──┬──CAT──┬──,name_relexp──,──┬──────────────────┬──►
                      └──CATBX┘                   └──cvol_relexp─────┘
►──,vol_list_relexp──┬──────────────────────────────────────────────────►◄
                     └──,DSCBTTR=dscb_ttr_relexp──┘
```

list_addrx
points to the parameter list (labeled listname) set up by the CAMLST macro instruction.

CAT or CATBX
code this operand as shown. Either CAT or CATBX can be coded; but, in either case, any missing indexes within a CVOL are created.

name_relexp
specifies the virtual storage location of the fully-qualified name of a data set. The name cannot exceed 44 characters. If the name is less than 44 characters, it must be followed by at least one blank. In a DFSMShsm environment, if the data set name is less than 44 characters, it must be padded with blanks until the 44-character length is reached.

cvol_relexp
specifies the virtual storage location of the 6-byte volume serial number of the CVOL catalog to which this catalog request is directed. For a discussion of the effect of specifying this operand, see “Catalog Search Order” on page 70.

vol list_relexp
specifies the virtual storage location of an area that contains a volume list. The list must begin on a halfword boundary and consist of an entry for each volume on which the data set is stored. The first two bytes of the list indicate the number of entries in the volume list; the number cannot be zero. Each 12-byte volume list entry consists of a 4-byte device code, a 6-byte volume serial number, and a 2-byte data set sequence number. The sequence number is always zero for direct access volumes.

DSCBTTR=*dscb ttr_relexp*
specifies the virtual storage location of the 3-byte relative track address (TTR) of the data set control block (DSCB). This DSCB is on the first or only volume of the data set. The address is relative to the beginning of the volume.

Programming Considerations for Multiple-Step Jobs

When executing multiple-step jobs, it is preferable to catalog or uncatalog data sets using JCL, instead of using IDCAMS, IEHPROGM, or a user program. Because step allocation and unallocation monitors data sets during job execution and is unaware of functions performed by user programs, conflicting functions can be performed or GDG orientation can be lost.

Unallocation can recatalog existing cataloged data sets at job termination. This action occurs when the data set is opened during the job and the DSCB TTR could not be found in the catalog entry. If you are using the CAMLST macro to uncatalog and then catalog data sets with new volume information, be sure to include the DSCB TTR.

Example

In the following example, the non-VSAM data set named A.B.C is cataloged. The data set is stored on two volumes.

```
          CATALOG ADDABC            CATALOG DATA SET A.B.C.

         Check Return Codes

ADDABC    CAMLST     CAT,DSNAME,,VOLUMES
DSNAME    DC         CL6'A.B.C'      ONE BLANK FOR DELIMITER
VOLUMES   DC         H'2'            DATA SET ON TWO VOLUMES
          DC         X'3010200E'     3380 DISK DEVICE CODE
          DC         CL6'000014'     VOLUME SERIAL NUMBER
          DC         H'0'            DATA SET SEQUENCE NUMBER
          DC         X'3010200E'     3380 DISK DEVICE CODE
          DC         CL6'000015'     VOLUME SERIAL NUMBER
          DC         H'0'            SEQUENCE NUMBER
```

The CATALOG macro instruction points to the CAMLST macro instruction. CAT, the first operand of CAMLST, specifies that a data set is to be cataloged. DSNAME, the second operand, specifies the virtual storage location of the data set name A.B.C. VOLUMES, the fourth operand, specifies the virtual storage location of the volume list.

Control is returned to your program at the instruction following the CATALOG macro instruction. Register 15 contains one of the return codes described under “Return Codes from CATALOG” on page 93.

Uncataloging a Non-VSAM Data Set (CATALOG and CAMLST UNCAT)

Use this macro to remove a data set reference and unneeded indexes.

The format of the CATALOG and CAMLST macros is:

```
►►──┬──────────┬──CATALOG──list_addrx──────────────────────────────────►◄
    └──label───┘
```

```
►►──listname──CAMLST──┬──UNCAT──┬──,name_relexp──┬──────────────────┬──►◄
                      └──UCATDX─┘                └──,cvol_relexp────┘
```

list_addrx
points to the parameter list (labeled listname) set up by the CAMLST macro instruction.

UNCAT *or* **UCATDX**
code this operand as shown. Either UNCAT or UCATDX can be coded but, in either case, unneeded indexes, with the exception of the highest-level index, are removed along with the data set reference.

name_relexp
specifies the virtual storage location of the fully-qualified name of a data set or index level. The name cannot exceed 44 characters. If the name is less than 44 characters, it must be followed by at least one blank. In a DFSMShsm envi-

ronment, if the data set name is less than 44 characters, it must be padded with blanks until the 44-character length is reached.

cvol_relexp
specifies the virtual storage location of the 6-byte volume serial number of the CVOL catalog to which this catalog request is directed. For a discussion of the effect of specifying this operand, see “Catalog Search Order” on page 70.

Example

In the following example, the catalog entry for data set A.B.C is removed from a catalog. In a CVOL, index B is removed unless it contains references to other data sets. Index A remains because it is the highest-level index.

```
         CATALOG REMOVE             REMOVE REFERENCES TO DATA SET A.B.C
*                                   FROM CATALOG

        Check Return Codes

REMOVE   CAMLST     UNCAT,DSNAME
DSNAME   DC         CL6'A.B.C'      ONE BLANK FOR DELIMITER
```

The CATALOG macro instruction points to the CAMLST macro instruction. UNCAT, the first operand of CAMLST, specifies that references to a data set are to be removed from the catalog. DSNAME, the second operand, specifies the virtual storage location of the fully-qualified name of the data set whose references are to be removed.

Control is returned to your program at the instruction following the CATALOG macro instruction. Register 15 contains one of the return codes described under “Return Codes from CATALOG” on page 93.

Note: The CAMLST UNCAT or UCATDX function is not supported for system-managed data sets. These are ignored. The function is not performed and the return code is 0.

Recataloging a Non-VSAM Data Set (CATALOG and CAMLST RECAT)

You can recatalog a non-VSAM data set using the CATALOG and CAMLST macro instructions. Recataloging is usually necessary if a data set is extended to a new volume.

Build a complete volume list in virtual storage consisting of an entry for each volume on which the data set resides. The first 2 bytes of the list indicate the number of entries in the list; the number might not be zero. Each 12-byte volume pointer consists of a 4-byte device code, a 6-byte volume serial number, and a 2-byte data set sequence number. The sequence number is always zero for direct access volumes.

The format of the CATALOG and CAMLST macros is:

```
►►──┬──────────┬──CATALOG──list_addrx──────────────────────────────────►◄
    └─label────┘
```

```
►►──listname──CAMLST──RECAT──,name_relexp──,┬──────────────┬──────────►
                                            └─cvol_relexp──┘
►──,vol_list_relexp──┬──────────────────────────────────┬──────────►◄
                     └─,DSCBTTR=dscb_ttr_relexp──────────┘
```

list_addrx
points to the parameter list (labeled listname) set up by the CAMLST macro instruction.

RECAT
code this operand as shown.

name_relexp
specifies the virtual storage location of the fully-qualified name of a data set. The name cannot exceed 44 characters. If the name is less then 44 characters, it must be followed by at least one blank. In a DFSMShsm environment, if the data set name is less than 44 characters, it must be padded with blanks until the 44-character length is reached. A C-type DC instruction can define the name.

cvol_relexp
specifies the virtual storage location of the 6-byte volume serial number of the CVOL catalog to which this catalog request is directed. For a discussion of the effect of specifying this operand, see “Catalog Search Order” on page 70.

vol list_relexp
specifies the virtual storage location of an area that contains a volume list. The area must begin on a halfword boundary.

DSCBTTR=*dscb ttr_relexp*
specifies the virtual storage location of the 3-byte relative track address (TTR) of the identifier DSCB. This DSCB is on the first or only volume of the data set. The address is relative to the beginning of the volume.

Example

In the following example, the two-volume data set named A.B.C is recataloged to add a third volume. An entry is added to the volume list, that previously contained only two entries.

```
          CATALOG RECATLG            RECATALOG DATA SET
*                                    A.B.C ADDING A NEW
*                                    VOLUME

         Check Return Codes

RECATLG   CAMLST    RECAT,DSNAME,,VOLUMES
DSNAME    DC        CL6'A.B.C '      FOR DELIMITER ONE BLANK
VOLUMES   DC        H'3'             THREE VOLUMES
          DC        X'3010200E'      3380 DISK DEVICE CODE
          DC        CL6'000014'      VOLUME SERIAL NUMBER
          DC        H'0'             SEQUENCE NUMBER
          DC        X'3010200E'      3380 DISK DEVICE CODE
          DC        CL6'000015'      VOLUME SERIAL NUMBER
          DC        H'0'             SEQUENCE NUMBER
          DC        X'3010200E'      3380 DISK DEVICE CODE
          DC        CL6'000016'      VOLUME SERIAL NUMBER
          DC        H'0'             SEQUENCE NUMBER
```

The CATALOG macro instruction points to the CAMLST macro instruction. RECAT, the first operand of CAMLST, specifies that a data set is to be recataloged. DSNAME, the second operand, specifies the virtual storage location of the fully-qualified name of the data set to be recataloged. VOLUMES, the fourth operand, specifies the virtual storage location of the volume list you have built.

Control is returned to your program at the instruction following the CATALOG macro instruction. If the data set has been successfully recataloged, register 15 contains zeros. Otherwise, register 15 contains one of the return codes described under “Return Codes from CATALOG.”

Return Codes from CATALOG

Control is returned at the instruction following the CATALOG macro instruction. Register 15 might contain one of the following return codes.

Note: Instead of one of these codes, you might get one of the codes listed in Figure 34 on page 94.

Figure 33 (Page 1 of 2). CATALOG Return Codes

Code	Meaning
0 (X'00')	Operation successful.
4 (X'04')	Either the required catalog does not exist, it is not open, or the do not allocate bit is on.
8 (X'08')	One of the following happened: • The existing catalog structure is inconsistent with the operation requested. If the error was detected while processing in a CVOL, register 0 has the number of valid index levels and register 1 has the return code that would have resulted if a LOCATE macro had been issued for the same entry name. If the error was detected in an integrated catalog facility or a VSAM catalog, register 0 contains the catalog return code and register 1 contains zero. • The user is not authorized to perform the operation. Register 0 contains decimal 56 ('38') and register 1 contains zero.
12 (X'0C')	Reserved.
16 (X'10')	The index structure necessary to catalog the data set does not exist.
20 (X'14')	There is insufficient space in the catalog data set.

Figure 33 (Page 2 of 2). CATALOG Return Codes

Code	Meaning
24 (X'18')	An attempt was made to catalog an improperly named generation data set, or the generation index is full and the named data set is older than any currently in the index. This return code only applies to CVOLs.
28 (X'1C')	One of the following happened: • A permanent I/O or unrecoverable error was encountered. • An error was found in a parameter list. • An I/O error occurred in a CVOL. • There was a nonzero return code from ESTAE or GETMAIN.

CVOL Processor Return Codes

Figure 34 describes return codes from the CVOL processor.

Figure 34. CVOL Processor Return Codes

Code	Meaning
0	Successful.
4	Unable to allocate a CVOL.
8	Either the data set was not found, there is an inconsistent structure of a CVOL, SYSCTLG does not exist on the specified volume, or the CVOL could not be opened.
24	There was an I/O or unrecoverable error on locate.
28	There was an I/O or unrecoverable error on nonlocate.
40	Insufficient space was supplied. The size of the calculated space is returned in the CTGFDBK area of the integrated catalog facility or VSAM catalog parameter list.
44	Insufficient space was supplied. Size required not known.
48	The function was not valid. It is not consistent with a CVOL.
56	Security or password verification failure.
84	DELETE failed because of unexpired purge date causing a SCRATCH failure.
164	ESTAE or GETMAIN return code was nonzero.
168	DELETE failed because of unmatched device type causing a SCRATCH failure.
184	DELETE failed because of data set being currently open, causing a SCRATCH failure.

Chapter 4. Executing Your Own Channel Programs

This chapter contains information about the execute-channel-program (EXCP) and execute-channel-program-virtual-real (EXCPVR) macro instructions and is provided for compatibility with other IBM operating systems. References to EXCP apply equally to EXCPVR unless otherwise stated. See "Executing Fixed Channel Programs in Real Storage (EXCPVR)" on page 108 for EXCPVR and EXCP differences. We recommend using an access method such as VSAM in place of EXCP.

The EXCP macro instruction allows you to control the data organization based on device characteristics. The exceptions to this capability are partitioned data sets extended (PDSEs), extended format data sets, spooled and dummy data sets, TSO terminals, and HFS (hierarchical file system) data sets. They are not supported for user-written applications using EXCP. This chapter describes the EXCP macro instruction application and function and includes descriptions of specific control blocks and macro instructions. Factors that affect the operation of EXCP, such as device variations and program modification, are also discussed.

Before reading this chapter, you should be familiar with the operational characteristics of the I/O devices required by your channel programs. Operational characteristics are described in IBM publications for each I/O device. Many of them are cited in "Referenced Publications" on page xvi. You also need to understand the information in the following publications:

- *High Level Assembler/MVS & VM & VSE Programmer's Guide* and *Assembler H V2 Programming Guide* contain the information necessary to code programs in the assembler language.
- *ESA/390 Principles of Operation* describes channel command words (CCWs) and channel programs.
- *DFSMS/MVS Using Data Sets* contains the standard procedures for I/O processing under the operating system.
- *DFSMS/MVS Macro Instructions for Data Sets* describes the system macro instructions that can be used in programs coded in the assembler language.

EXCP is primarily for I/O programming situations that cannot be dealt with using standard access methods. When writing your own access method, include EXCP for I/O operations. You must also use EXCP for processing nonstandard magnetic tape labels, including reading and writing labels and positioning volumes.

To issue EXCP, provide a channel program (a list of channel command words) and control blocks in your program area. The I/O process then schedules I/O requests for the specified device, executes commands, handles interruptions, directs error recovery procedures, and posts the results of I/O requests.

Using EXCP

This section explains the procedures required when issuing the EXCP macro instruction. To issue the EXCP macro instruction directly, perform the following tasks:

1. Construct and open a data control block (DCB) with the DCB and OPEN macro instructions. See "Data Control Block (DCB) Fields" on page 110.

© Copyright IBM Corp. 1979, 1999

2. Create a channel program containing channel commands for I/O operations on the appropriate device.
3. Construct an input/output block (IOB) containing information about the channel program. See “Input/Output Block Fields” on page 120.
4. Construct an event control block (ECB) to be posted with a completion code when the channel program terminates. See “Event Control Block Fields” on page 124.
5. Issue an EXCP macro instruction to pass the address of the IOB to the routines that initiate and supervise I/O operations.

The I/O process generally handles scheduling the channel program and posting the completion codes.

After issuing EXCP, issue a WAIT or EVENTS macro instruction specifying the address of the ECB, to determine whether the channel program has terminated. If volume switching is necessary (because of unit exception or end of DASD extent), issue an EOV macro instruction. When data set processing has been completed, issue a CLOSE macro instruction to restore the DCB.

Programming interfaces for EXCP are compatible between MVS/370 and MVS/ESA or OS/390, except for supporting virtual and central storage addresses above 16MB. EXCP is available to programs executing in either 24-bit or 31-bit addressing mode. The following restrictions apply:

- EXCP only supports a 24-bit virtual storage interface except for 31-bit IDAW, a DCBE, and an XTIOT entry. In addition, all areas related to I/O operations (for example, channel command words, IOBs, DEBs, appendages, and so forth), must remain 24-bit virtual addressable. EXCP (channel command word translator) allows 24-bit virtual I/O buffers to be fixed above 16MB real. When a channel command word (CCW) references a central storage address above 16MB, the CCW translator builds one or more indirect addressing words (IDAWs) for that CCW. All virtual addresses, except those in IDAWs, must be below 16MB. For V=R users, CCWs and IDAWs must be below 16MB real.
- Only format-0 CCWs are supported.
- All user-specified appendage routines are given control in 24-bit addressing mode and must return in the same mode.

31-Bit IDAW Requirements

A virtual channel program provided by the EXCP caller can have one or more CCWs with the IDA flag set and the address portion of these CCWs pointing to a single 4-byte IDAW. This EXCP function is referred to as virtual IDAWs.

The 4-byte IDAW can contain a valid virtual address up to the maximum 31-bit address. Virtual IDAWs are supported on all virtual CCWs except:

- Transfer in channel (TIC) commands
- All nondata-transfer type commands: for example, recalibrate, rewind, forward space, fold, block data check, no operation, control commands
- Read, read backward, and sense commands, with the skip flag set.

The same addressing restrictions apply to EXCPVR inputs except that 31-bit real data areas can be specified by the user-created CCWs through the use of IDAWs.

All CCWs and IDAWs must be below 16MB real. Only format-0 CCWs are supported.

All other areas related to the EXCP/EXCPVR I/O operation (for example, CCWs, IDAWs, IOBs, DEBs, DCBs, appendages, and so forth) must remain 24-bit addressable. The EXCP processor, however, allows both 24-bit and 31-bit virtual I/O buffers to be fixed above 16MB real.

EXCP Requirements

This section describes the channel program that is needed to issue EXCP and the control blocks to be constructed directly or by using macro instructions.

All areas related to the EXCP/EXCPVR I/O operation (for example, CCWs, IDAWs, IOBs, DEBs, DCBs, appendages, and so forth) must remain 24-bit addressable. The EXCP processor, however, allows both 24-bit and 31-bit virtual I/O buffers to be fixed above 16MB real.

Channel Program

The channel program you supply is composed of CCWs on doubleword boundaries. Each channel command word specifies a command to be executed and, for data transfer commands, the source or destination area. CCW operation codes are described in the IBM publications for each I/O device.

You can specify both data chaining and command chaining by setting applicable chaining bits in the channel command word and indicating the type of chaining in the IOB. If an I/O error occurs while your channel program executes, the corresponding chaining bits in the IOB need to be set. Otherwise, error recovery could be impossible. The integrity of your data could be compromised. (See “Input/Output Block Fields” on page 120 for additional information.) If you specify both data and command chaining in the same channel command word, data chaining takes precedence.

EXCP does not support channel programs that modify themselves, regardless of the method of modification, except as noted in “Virtual and Central Storage Addresses” on page 99. The intended modification in virtual storage has no effect on the running real-channel program (see “Modifying a Channel Program during Execution” on page 99).

Control Blocks Used with EXCP

Using EXCP requires familiarity with the function and structure of the IOB, ECB, DCB, DEB, and optionally IDAW, IOBE, DCBE, UCB and IEDB. DCB, IOB, IOBE, IEDB, and ECB fields are illustrated in “Control Block Fields” on page 110. The DEB fields used for EXCP and EXCPVR are illustrated in Appendix C, “Control Blocks” on page 419 (all the DEB fields are illustrated in *DFSMS/MVS DFSMSdfp Diagnosis Reference*). The handling of IDAWs is described in “SIO Appendage” on page 131. The UCB is documented in *OS/390 MVS Data Areas, Vol 5 (SSAG-XTLST)* .

All EXCP control blocks that you provide must be located in storage that your program's protection key allows the program to modify. Descriptions of these control blocks follow.

Input/Output Block

The IOB is used for communication between the problem program and the system. It provides the addresses of other control blocks, and maintains information about the channel program, such as the type of chaining and the progress of I/O operations. You must define the IOB and specify its address as the only parameter of the EXCP or EXCPVR macro instruction.

Event Control Block

The ECB provides you with a completion code that describes whether the channel program was completed with or without error. A WAIT or EVENTS macro instruction, which can be used to synchronize I/O operations with the problem program, must identify the ECB. You must define the ECB and specify its address in the IOB.

Data Control Block

The DCB provides the system with information about the characteristics and processing requirements of a data set to be read or written by the channel program. A DCB must be generated by a DCB macro instruction that includes parameters for EXCP. If you are not using appendages, a short DCB is constructed. Such a DCB does not support reduced error recovery. You specify the address of the DCB in the IOB.

Data Control Block Extension

The DCBE provides further processing options. The only option currently supported by EXCP is the EODAD parameter when you issue an EOV macro.

Data Extent Block

The DEB contains one or more extent entries for the associated data set and other control information. An extent defines all or part of the physical boundaries on an I/O device occupied by, or reserved for, a particular data set. Each extent entry contains the address of a unit control block (UCB) that provides information about the type and location of an I/O device. More than one extent entry can contain the same UCB address. For all I/O devices supported by the operating system, the DEB is produced during execution of the OPEN macro instruction for the DCB. The system places the address of the DEB into the DCB.

Input/Output Block Common Extension

The IOBE can control the level of ERP processing and provides the anchor to the IEDB. The IOBE is an extension to the IOB and, like the IOB, provides for communication between the user of EXCP and the system. This control block is optional.

Input/Output Error Data Block

The system uses the IEDB to provide extended error information. This control block is optional.

Executing the Channel Program

This section explains how the system uses your channel program and control blocks after you issue EXCP.

Initiating the Channel Program

Issuing EXCP requests execution of the channel program specified in the IOB. The I/O process validates the request by checking fields of the control blocks associated with this request. If the I/O process detects invalid information in a control block, it initiates abnormal termination procedures. The EXCP processor gets the address of the:

- DCB from the IOB
- DEB from the DCB
- UCB from the DEB.

If you have provided a start I/O (SIO) appendage, the EXCP processor passes control to it. The return address from the SIO appendage determines whether the EXCP processor executes or skips the I/O operation. For a description of the SIO appendage and its linkage to the EXCP processor, see “Start-I/O Appendage” on page 130.

Virtual and Central Storage Addresses

The channel program you construct contains virtual addresses.

Unless your program is running with APF-authorization with ADDRSPC=REAL (V=R) on the JCL EXEC statement, the EXCP processor performs the following:

- Copies your virtual channel program and translates the copy into one that uses only central storage addresses
- Fixes in real storage the pages used as I/O areas for the data transfer operations specified in your channel program.

The EXCP processor builds the translated (real) channel program in a portion of real storage.

User-constructed channel programs for I/O operations of an APF-authorized program in a V=R address space are not translated. Because the address space is V=R, any CCWs created by the user have correct real data addresses. (Translation would only re-create the user's channel program, so the CCWs are used directly.)

If an active channel program is modified by data read in or by processor instructions in a V=R address space, it takes effect in an APF-authorized address space, but not in a V=V address space.

Modifying a Channel Program during Execution

Except for APF-authorized programs in a V=R address space, the system builds a translated copy of your channel program in real storage. Channel program changes made by your page-fix or start-I/O appendage will affect the real channel program. Later changes to your copy of your channel program with processor instructions or with data read in by an I/O operation will not effect the real, translated channel program. Thus, in a V=V address space any attempt to modify an active channel program affects only the virtual image of the channel program, not the real channel program being executed by the channel subsystem.

If you wish your changed channel program to be executed, your program can issue another EXCP macro or your channel-end appendage can return at offset 8.

Device Type Considerations

DASD Considerations

For direct access devices, specify the seek address in the IOB. For a CKD device, the system constructs a CCW chain to issue the seek and the set file mask specified in the DEB, and to pass control to the real version of your channel program. You cannot issue the initial seek or set file mask CCWs. For an ECKD device, the system constructs a define extent command instead of the seek and set file mask command. The system uses the contents of DEBXDEF when building the define extent command. It is described in Appendix C, “Control Blocks” on page 419. You can issue a define extent command at the beginning of the channel program. The system copies the data area and replaces the beginning and ending extent addresses and the file mask byte.

For both CKD and ECKD, the file mask is set to prohibit seek-cylinder CCWs, or, if space is not allocated in full cylinders, seek-head commands. If the data set is open for INPUT, write CCWs are also prohibited. For an ECKD device, your channel program can contain locate-record CCWs.

Command Retry Considerations

Command retry is a function of the IBM 3990 storage control. When the channel subsystem receives a retry request, it repeats the execution of the CCW, without requiring any additional input/output interrupts. For example, a storage control might initiate a retry procedure to recover from a transient error.

A command retry during the execution of a channel program can cause the following conditions to be detected by the initiating program:

- Modifying CCWs: For an APF-authorized program running in a V=R address space, a CCW used in a channel program must not be modified before the CCW operation has been successfully completed. Without the command retry function, a command is fetched only once from storage by a channel. This allowed a program to determine through condition codes or program controlled interruptions (PCI) that a CCW had been fetched and accepted by the channel. The CCW could be modified before execution. With the command retry function, this procedure cannot be repeated because the channel fetches the CCW from storage again on a command retry sequence. In the case of data chaining, the channel retries commands starting with the first CCW in the data chain.
- Program Controlled Interrupts (PCI): A CCW containing a PCI flag can cause multiple program-controlled interrupts. This will happen if the PCI-flagged CCW was retried during a command retry procedure and a PCI could be generated each time the CCW is executed.
- Residual Count: If a channel program is prematurely terminated during the retry of a command, the residual count in the channel status word (CSW) will not necessarily indicate how much storage was used. For example, if the storage control detects a wrong-length record error condition, an erroneous residual count is stored in the CSW until the command retry is successful. When the retry is successful, the residual in the CSW reflects the correct length of the data transfer.

- Command Address: When data chaining with command retry, the CSW might not indicate how many CCWs have been executed at the time of a PCI. For example:

CCW#	Channel Program
1	Read, data chain
2	Read, data chain
3	Read, data chain, PCI
4	Read, command chain

  In this example, assume that the storage control signals command retry on Read #3 and the processor accepts the PCI after the channel resets the command address to Read #1 because of command retry. The CSW stored for the PCI will contain the command address of Read #1 when the channel has actually progressed to Read #3.

- Testing Buffer Contents on Data Read: Any program that tests a buffer to determine when a CCW has been executed and continues to execute based on this data can get incorrect results if an error is detected and the CCW is retried.

Magnetic Tape Considerations

For a magnetic tape device, the system constructs a CCW chain to set the mode specified in the DEB and pass control to the real version of your channel program. (You cannot set the mode yourself.) For cartridge tape devices, the mode byte also prohibits supervisor channel command words such as the mode set command. If your program opens a tape for input or read backwards and you do not have RACF authority to write on the volume, the system normally prevents writes. With a reel tape the system does this by requiring the operator to remove the write ring. With a cartridge tape, the system does this using the Mode Set command. See *DFSMS/MVS Using Magnetic Tapes* for more information on tape handling.

Lost Data Condition on IBM 3800

With the IBM 3800 Printing Subsystem, a cancel key or a system-restart-required paper jam causes both a lost data indicator to be set in DCBIFLGS and a lost page count and channel page identifier to be stored in the UCB extension. Reset the lost data indicator bit (DCBIFLDT) and the first two bits in the DCBIFLGS field to zero before reissuing requests to the printer. For additional information see *IBM 3800 Printing Subsystem Programmer's Guide* and *IBM 3800 Printing Subsystem Models 3 and 8 Programmer's Guide.*

Completing Execution

The system considers the channel program completed when it receives a channel-end condition in the subchannel status word (SCSW). Unless a channel-end (CHE) or abnormal-end (ABE) appendage directs otherwise, a completion code is placed in the ECB, IOB, and IEDB. The completion code refers to errors associated with channel end. If device and channel end occur simultaneously, errors associated with device end (that is, unit exception or unit check) are also accounted for.

If device end follows channel end and an error is associated with device end, the completion code in the ECB will not indicate the error. However, the status of the unit and channel is saved by the system for the device. The next I/O request directed to the I/O device from any address space is marked as intercepted. The

error is assumed to be permanent, and the completion code in the ECB for the intercepted request indicates interception. The DCBIFLGS field of the DCB will also indicate a permanent error. Note that, if a write-tape-mark or erase-long-gap CCW is the last or only CCW in your channel program, the I/O process does not attempt recovery procedures for device end errors. In these circumstances, command chaining a NOP CCW to your write-tape-mark or erase-long-gap CCW ensures initiation of device-end error recovery procedures.

To be prepared for device-end errors, you should be familiar with device characteristics that can cause such errors. After one of your channel programs has terminated, do not release buffer space until determining that your next request for the device has not been intercepted. You can reissue an intercepted request.

Interruption Handling and Error Recovery Procedures

An I/O interruption allows the processor to respond to signals from an I/O device that indicate either termination of a phase of I/O operations or external action on the device. A complete explanation of I/O interruptions is contained in the publication *ESA/390 Principles of Operation.* For descriptions of interruption by specific devices, see the IBM publications for each device.

If error conditions are associated with an interruption, the system schedules the appropriate device-dependent error routine. The operating system might then start another request that is not related to the channel program in error. If the error recovery procedures (ERPs) fail to correct the error, the system places an error code in the ECB, IOB, and IEDB.

A channel program might depend upon the successful completion of a previous channel program (as when one channel program retrieves data to be used in building another). The previous channel program is called a related request and must be identified to the system. For a description of this procedure, see IOBFLAGS in “Input/Output Block Fields” on page 120 and “Purging and Restoring I/O Requests” on page 194.

If a permanent error occurs in the channel program of a related request, the system removes the request queue elements for all dependent channel programs and returns them to the caller without executing the request. For all requests dependent on the channel program in error, the system places completion codes in the ECBs. The system also places ones in the first two bit positions of the DCBIFLGS field of the DCB. Any requests for a DCB with error flags are posted complete without execution. To reissue requests that depend on the channel program in error, reset the first two bits of the DCBIFLGS field of the DCB to zero. Then reissue EXCP for each desired channel program.

Reexecuting Channel Programs by Error Recovery Procedures: Under some circumstances the ERP might reexecute a channel program from the beginning. You will want to build channel programs that the ERP can restart after any CCW has failed.

If a CCW modifies a data area used by an earlier CCW, the channel program might not reexecute properly.

The following are some situations where a channel program might not give correct results when reexecuted by an APF-authorized program in a V=R address space (see “Modifying a Channel Program during Execution” on page 99):

- The channel program modifies itself.
- The application program or the PCI appendage modifies the channel program or a data area before receiving notification that the channel program has completed. Generally you can attempt to add CCWs to the end of the channel program.

Example: The following is an example of a DASD channel program that will not always execute correctly. Using this channel program is inadvisable.

```
         CCW   X'47',LRArea,X'60',L'LRArea   Locate Record
         CCW   X'86',Data,X'60',L'Data       Read Data
         CCW   X'92',Search,X'60',L'Search   Read Count
         CCW   X'22',Sector,X'20',1          Read Sector for this count
LRArea   DC    0XL16             Area for Locate Record command
         DC    X'06010002'       Operation, auxiliary, block count (2)
Seek     DC    X'xxxxxxxx'       BBCCHH for seek
Search   DC    X'xxxxxxxxxx'     Search argument
Sector   DC    X'FF0000'         Sector and transfer length factor
Data     DC    ----
```

The application program would have to store appropriate values at labels "Seek" and "Search" before issuing EXCP. (It will execute more efficiently if an appropriate value is stored at "Sector" also.)

If all or part of the third CCW executes (Read Count), then reexecution of the channel program by the ERP will give different results.

This programming example demonstrates the use of mixed case letters that IBM High Level Assembler handles.

Requesting Extended Error Information: You can request that EXCP and EXCPVR processing return extended error information. This extended error information consists of sense information and a completion code, which you can use to help more accurately diagnose I/O errors later encountered. Extended error information is available for all devices.

To request extended error information, do the following:

- Follow the EXCP/EXCPVR initialization procedures described in “Input/Output Block Fields” on page 120, including building and initializing the DCB, IOB and ECB.
- Define an input/output block common extension (IOBE) and an input/output error data block (IEDB) using the IOSDIOBE and IOSDIEDB mapping macros, respectively. Each must be defined on word boundaries. You must supply both an IOBE and an IEDB to receive extended error information. Initialize the fields as described in “Input/Output Block Common Extension Fields” on page 123 and “Input/Output Error Data Block Fields” on page 124.
- Set register 0 to the address of the IOBE.
- Set register 1 to the address of the IOB.
- Set flag IOBCEF on in the IOB to indicate that the IOB extension is present.
- Issue the EXCP or EXCPVR macro (see “Using Executable Macro Instructions with EXCP” on page 105 and “Executing Fixed Channel Programs in Real Storage (EXCPVR)” on page 108 respectively).

The IEDB contains the following status information:

- The completion code from I/O processing that gives the status. The completion codes and their meanings are shown in Figure 45 on page 125.

  The completion code is contained in the IEDBCOD, which is updated with the results of the I/O requests. The system sets the IEDBCOD field prior to calling the abnormal-end, normal-end, PCI, and end-of-extent appendages. The system also sets this field when the ECB is posted, whether or not you have set any appendages. The system can set the IEDBCOD multiple times as events occur.
- The sense information, set only after a unit check. Once set, the sense information remains until overlaid by the next unit check. Since the system never clears this area, you might want to clear it before issuing EXCP. See the appropriate device publication for an explanation of the sense information.

Requesting Different Levels of ERP Processing: You can request the system to limit error recovery procedure (ERP) processing to selected functions, typically when EXCP or EXCPVR processing has encountered an I/O error. The processing selections available depend upon the device type.

For all devices, except magnetic tape subsystems that use cartridges, the following processing levels are available:

- **No ERP processing**. ERP processing does not attempt error recovery or issue messages. However, ERP processing can perform recovery for non-error unit checks for logging, forced logging mode and buffered log overflow. To request no ERP processing, set at least one of the DCBIFIOE bits on in the DCBIFLGS field in the DCB.
- **Full ERP processing.** ERP processing performs such functions as logging the errors, logging data collected by a control unit for a device, retrying errors, issuing error messages and processing requests from the device. You do not have to request full ERP processing; it is the system default.

For magnetic tape subsystems that use cartridges, such as the 3490 or 3590-1, the following processing levels are available:

- **Basic ERP processing.** ERP processing logs the errors, logs data collected by a control unit for a device and processes requests from a device. In this case, the system does not issue messages or retry errors. To request basic ERP processing, set one of the DCBIFIOE bits on in the DCBIFLGS field in the DCB.
- **Intermediate ERP processing.** ERP processing performs the functions provided by basic processing and also issues any permanent error messages. To request intermediate ERP processing, define an IOBE and set IOBEPMSG on in IOBEERPM. In addition, set at least one of the DCBIFIOE bits on in the DCBIFLGS field in the DCB.
- **Full ERP processing.** ERP processing performs such functions as logging the errors, logging data collected by a control unit for a device, retrying errors, issuing error messages and processing requests from the device. You do not have to request full ERP processing; it is the system default. Bits in DCBIFIOE must be 0. The DCB macro assembles them as zeros.

See *DFSMS/MVS Planning for Installation* to find which magnetic tape subsystems use cartridges.

VIO Considerations: When you issue EXCP or EXCPVR against a VIO data set, the system simulates all the common channel commands. When working with VIO data sets, if you supply an IEDB the system verifies its validity, but will not set any fields in it in the current level of the operating system.

Using Executable Macro Instructions with EXCP

When using the EXCP macro instruction, also use OPEN, CLOSE, and, in some cases, the EOV macro instruction. The parameters of the EOV and EXCP macro instructions are described here. The parameters and different forms of the OPEN (without TYPE=J) and CLOSE macro instructions are described in *DFSMS/MVS Macro Instructions for Data Sets* . The OPEN macro with the TYPE=J parameter is described in this manual. See “Executing Fixed Channel Programs in Real Storage (EXCPVR)” on page 108 for a description of the EXCPVR macro.

Initializing Data Control Blocks

The OPEN macro instruction initializes one or more DCBs so that their associated data sets can be processed. Issue OPEN for all DCBs used by your channel programs. (A dummy data set cannot be opened for EXCP.) Some of the procedures performed by OPEN are:

- Checking data set access authorization
- Constructing the data extent block (DEB)
- Completing the fields in the DCB
- Verifying or creating standard labels
- Positioning tape
- Loading your appendage routines.

The parameters and different forms of the OPEN macro instruction are described in *DFSMS/MVS Macro Instructions for Data Sets* and “OPEN—Initialize Data Control Block for Processing the JFCB” on page 193.

Direct Data Set Considerations

To process a multivolume direct data set (BDAM) with EXCP, use the open routines to build a data extent block for each volume and to issue mount messages for them. This can be done by reading in the JFCB with a RDJFCB macro instruction and opening each volume of the data set with a separate DCB. See “ Using RDJFCB to Process a Multivolume Direct Data Set” on page 184 for an example of how to code a routine to do this, and “Reading and Modifying a Job File Control Block” on page 178 for further uses of the RDJFCB macro.

VSAM Data Set Considerations

With a DCB used to open a component of a VSAM data set you can:

1. Verify that an application has master password or RACF alter authority for the data set.
2. Read from or write to a data set to repair data set or catalog damage if normal VSAM processing cannot. Because of the potential for damaging a valid data set or catalog, exercise extreme caution when writing an application using this interface.

You can specify a DCB when opening a component (data or index) of a VSAM data set if the following conditions are met:

- The application has master password or RACF alter authority for the data set.
- The component must not reside on multiple volumes.
- The component must not be a member of a concatenation.
- The DCB must specify the EXCP access method.
- The data set disposition must be either (OLD,KEEP,KEEP) or (SHR,KEEP,KEEP).
- The DCB must specify either the INPUT or UPDAT option.
- Your program must be either APF authorized or in supervisor state.

Executing a Channel Program

The EXCP macro instruction initiates channel program I/O operations. Whenever you want to execute one of your channel programs, issue EXCP.

The format of the EXCP macro is:

```
►►──┬──────────┬──EXCP──iob_addr──────────────────────────────────────►◄
    └─label────┘
```

iob_addr—**RX-type address, (2-12),** or **(1)**
the address of the input/output block of the channel program to be executed.

If your program is also supplying an IOBE, then set register 0 as described in “Requesting Extended Error Information” on page 103.

End of Volume

The EOV macro instruction identifies end-of-volume and end-of-data-set conditions. For an end-of-volume condition, EOV causes switching volumes and verifying or creating standard labels. For an end-of-data-set condition (except when another data set is concatenated), EOV causes your end-of-data set routine to be entered. Before processing trailer labels on a tape input data set, decrement the DCBBLKCT field. Your program issues EOV for any of the following reasons:

- Switching magnetic tape or direct access volumes is necessary
- Performing a secondary allocation on the same or another volume for a direct access data set opened for output
- Switching to the next concatenated data set is necessary.

For magnetic tape, issue EOV when either a tapemark is read or a write command received a unit exception condition. You can also issue EOV to go to the next volume or data set even though neither a tapemark was read nor end-of-tape reached. Bit settings in the 1-byte DCBOFLGS field of the DCB determine the action taken when EOV is executed. Before issuing EOV for magnetic tape or DASD make sure that appropriate bits are set in DCBOFLGS. Bit positions 2, 3, 6, and 7 of DCBOFLGS are used only by the system; you are concerned with bit positions 0, 1, 4, and 5. The use of these DCBOFLGS bit positions is as follows:

Figure 35. DCBOFLGS Usage

Position	Meaning
0	set to 1 indicates that a write command was executed and that a tapemark or DASD file mark is to be written. OPEN sets this bit to 1 if the OPEN option is OUTPUT, OUTIN, OUTINX, or EXTEND. OPEN sets it to 0 for any other OPEN option. See page 117 for more information.
1	indicates that a backward read was the last I/O operation
4	indicates that concatenated data sets are to be treated as unlike. For further information, refer to *DFSMS/MVS Using Data Sets*.
5	indicates that a tapemark has been read (tape only).

If bits 0 and 5 of DCBOFLGS are both off when EOV is executed, EOV spaces the tape past a tapemark, and standard labels, if present, are verified on both the old and new volumes. The direction of spacing depends on bit 1. If bit 1 is off, the tape is spaced forward; if bit 1 is on, the tape is backspaced.

For tape, if bit 0 is on, but bit 5 is off, EOV writes a tapemark at the current position, which is assumed to be following the last data record of the data set on the current volume. EOV also writes labels if it is a labeled tape. See *DFSMS/MVS Using Magnetic Tapes* for information on label processing. For DASD, if bit 0 is on, EOV attempts to write a file mark. See the DCBFDAD description on page 117

When you issue EOV, the system might rebuild the DEB at another location. After each EOV, obtain the rebuilt DEB address from the DCB. If the data set was allocated with the nocapture option of dynamic allocation, then EOV might have uncaptured the UCB for the previous volume. If so, that UCB might become invalid. Note that the 24-bit address of the new UCB might be the same as the previous UCB.

After issuing EOV for sequentially organized output data sets on direct access volumes, you can determine whether additional space was obtained on the same or a different volume. Do this by examining the DEB and the UCB. If the volume serial number in the UCB has not changed, additional space was obtained on the same volume. Otherwise, space was obtained on a different volume.

The format of the EOV macro is:

```
►►──┬──────────┬──EOV──dcb_addr──┬──────────────────────────────────────►◄
    └─label────┘                 │                  ┌─24─┐
                                 └─,MODE=──┬────────┬─┘
                                           └─31─────┘
```

dcb_addr—**RX-type address, (2-12),** or **(1)**
: the address of the DCB that is opened for the data set. If this parameter is specified as (1), register 1 must contain this address.

MODE=24 or **31**
: indicates how the registers are set. With either value, your program can be in 24-bit or 31-bit mode. The modes are:

24 If you do not specify the MODE operand, this mode is assumed. The expansion of the EOV macro stores the DCB address into register 1. The high-order byte of register 1 is ignored during EOV processing. The DCB must be below 16MB, but the calling program can be above the line.

31 The EOV macro expansion code puts the DCB address into register 15 and sets register 1 to zeros. The DCB address must be below 16MB[7] .The high-order byte of the address specified in the EOV macro must be zero.

Note: To determine how the system disposes of a tape volume when an EOV macro is issued, see the description of the DISP parameter of the OPEN macro in *DFSMS/MVS Macro Instructions for Data Sets*.

Restoring Data Control Blocks

The CLOSE macro instruction restores one or more DCBs so that processing of their associated data sets can be terminated. Issue CLOSE for all DCBs that were used by your channel programs. Some of the procedures performed when CLOSE is executed are:

- Releasing data extent block
- Removing information transferred to DCB fields when OPEN was executed
- Verifying or creating standard labels
- Volume disposition
- Releasing programmer-written appendage routines
- Uncapture UCB if the actual UCB is above the line and the allocation was done with the nocapture option

When CLOSE is issued for a data set on magnetic tape volume, labels are processed according to bit settings in the DCBOFLGS field of the DCB. Before issuing CLOSE for magnetic tape, set the appropriate bits in DCBOFLGS. The significant DCBOFLGS bit positions are listed in the EOV macro instruction description.

When CLOSE is issued for a data set open for output on a direct access device, the system will try to write a file mark if fields are as described in “Device-Dependent Parameters” on page 116.

The parameters and different forms of the CLOSE macro instruction are described in *DFSMS/MVS Macro Instructions for Data Sets*.

Executing Fixed Channel Programs in Real Storage (EXCPVR)

The EXCPVR macro instruction provides you with the same functions as the EXCP macro instruction (that is, a device-dependent means of performing input/output operations). In addition, it allows your program to improve the efficiency of the I/O operations in a paging environment by translating its own virtual channel programs to real channel programs. In order to issue EXCPVR, your program must be executing in protection key zero to seven, executing in supervisor state, or be APF authorized.

[7] Providing a DCB above 16MB causes an ABEND50D.

The program issuing the EXCPVR must remain in authorized state until completion of the channel programs. A description of how to authorize a program is in *OS/390 MVS Authorized Assembler Services Guide*.

The format of the EXCPVR macro is:

```
►►──┬──────────┬──EXCPVR──iob_addr──────────────────────────────────►◄
    └─label────┘
```

iob_addr—**RX-type address, (2-12),** or **(1)**
the address of the input/output block of the channel program to be executed.

If your program is also supplying an IOBE, then set register 0 as described in "Requesting Extended Error Information" on page 103.

To use EXCPVR, follow the procedures needed to execute an EXCP request and also follow the procedures listed below. In a V=R address space, steps 1 and 2 are not necessary.

1. Code PGFX=YES in the DCB associated with the EXCPVR requests and provide a page-fix (PGFX) appendage by specifying SIOA=symbol in the DCB. In a V=R address space, this step is not necessary.
2. Fix the data area containing your channel program, the data areas referred to by your channel program, the PCI appendage (if your program can generate program-controlled interrupts), and any area referred to by the PCI appendage including the DEB, IOB, etc.. To fix these areas, build a list in your PGFX appendage containing the addresses of these virtual areas. Any area that you know already is in fixed storage for the duration of the I/O can be omitted from the page fix list.

The following two items must be done in your start I/O (SIO) appendage. The SIO appendage is described in "Start-I/O Appendage" on page 130.

3. Determine whether the data areas in virtual storage specified in the address fields of your CCWs cross page boundaries. If they do, build an indirect data address list (IDAL) and put the address of the IDAL in the affected CCW and turn on the IDA bit in the CCW. Refer to *ESA/390 Principles of Operation* for more information.
4. Translate the addresses in your CCWs from virtual to central storage addresses.

All other areas related to the EXCP/EXCPVR I/O operation (that is, CCWs, IDAWs, IOBs, DEBs, DCBs, appendages, and so forth) must remain 24-bit addressable. The system allows both 24-bit and 31-bit virtual I/O buffers to be fixed above 16MB real.

EXCPVR users can specify 31-bit real data areas by creating CCWs that point to IDAWs.

Control Block Fields

The fields of the DCB, input/output block, IOBE, IEDB, event control block, and data extent block are illustrated and explained in this section. See “DSORG Parameter of the DCBD Macro” on page 120 to use the DCBD macro to map the DCB.

Data Control Block (DCB) Fields

The EXCP form of the DCB macro instruction produces a DCB that can be used with the EXCP macro instruction. Code a DCB macro instruction for each data set to be processed by your channel programs. (Notation conventions and format illustrations of the DCB macro instruction are given in *DFSMS/MVS Macro Instructions for Data Sets* .) DCB parameters that apply to EXCP depend on the following elements of the DCB that are generated:

- Foundation block: This portion is required and is always 12 bytes in length. Specify two of the parameters in this category.
- EXCP interface: This portion is optional. If you specify any parameter in this category, 20 bytes are generated.
- Foundation block extension and common interface: This portion is optional and is always 20 bytes in length. If this portion is generated, the device-dependent portion is also generated.
- Device dependent: This portion is optional and is generated only if the foundation block extension and common interface portion is generated. Its size ranges from 4 to 20 bytes, depending on specifications in the DEVD parameter. If you do not specify the DEVD parameter and the foundation extension and common interface portions are generated, 20 bytes for this portion are also generated.

Some of the procedures performed by the system when the DCB is opened and closed (such as writing file marks for output data sets on direct access volumes) require information from optional DCB fields. Make sure that the DCB is large enough to provide information for the procedures you want the system to handle.

Figure 36 on page 112 shows the relative position of each portion of an opened DCB. The fields, corresponding to the DCB macro instruction parameters, are also identified, except for DDNAME. (DDNAME is not included in a DCB that has been opened.) The fields in parentheses represent system information that is not associated with parameters of the DCB macro instruction.

Sources of information for DCB fields other than the DCB macro instruction are data definition (DD) statements or dynamic allocation parameters, data set labels, and DCB modification routines. You can use any of these sources to specify DCB parameters. However, if a particular portion of the DCB is not generated by the DCB macro instruction, the system will not accept information intended for that portion from any alternative source.

You can provide symbolic names for the fields in one or more EXCP DCBs by coding a DCBD macro to generate a dummy control section (DSECT). For further information, see “DSORG Parameter of the DCBD Macro” on page 120.

DCB Fields that do not have Macro Parameters

DCBOFLGS: See “End of Volume” on page 106.

DCBIFLGS: See “Completing Execution” on page 101 and “Interruption Handling and Error Recovery Procedures” on page 102.

DCBTIOT: If the data set allocation was dynamic and had the nocapture option S99ACUCB, then this field contains zeroes. You can use DEBXTNP, DEBXDSAB and DSABTIOT to get the TIOT entry address. If the allocation was not dynamic or did not have the nocapture option, this field is unsigned (16-bit) offset in the TIOT to the entry.

0 The device-dependent portion of the data control block varies in length and format according to specifications in the DSORG and DEVD parameters. Illustrations of this portion for each device type are included in the description of the DEVD parameter.		Device Dependent
20 BUFNO	BUFCB	Common Interface
24 BUFL	DSORG	
28 IOBAD		
32 BFTEK, BFALN	EODAD	Foundation Block Extension
36 RECFM	EXLST	
40 (TIOT)	MACRF	Foundation Block
44 (IFLGS)	(DEB Address)	
48 (OFLGS)	Reserved	
52 OPTCD	Reserved	EXCP Interface
56 Reserved		
60 EOEA	PCIA	
64 SIOA	CENDA	
68 XENDA	Reserved	

Note: The fields in parentheses represent system information that is not associated with parameters of the DCB macro instruction.

Figure 36. Data Control Block Format for EXCP (After OPEN)

Foundation Block Parameters

DDNAME=*symbol*

The name of the data definition (DD) statement that describes the data set to be processed. This parameter is required and must be supplied before your program issues the OPEN macro.

MACRF=E
The EXCP macro instruction is to be used in processing the data set. This operand must be coded and must be supplied before your program issues the OPEN macro.

REPOS=Y or **N**
Magnetic tape volumes: This parameter indicates to the system whether the user is keeping an accurate block count. (Maintain the block count in the DCBBLKCT field.) The system ignores this parameter when the program is executing and the device is not magnetic tape.

The system uses this information for the following:

- IBM standard and ISO/ANSI standard labels.
- Dynamic device reconfiguration (DDR) routine, for open magnetic tape devices with reels, not cartridges.
- Taking a checkpoint for open magnetic tape devices with reels, not cartridges.

Y—The count is accurate and the DDR routine can attempt to swap the volume.

N—The block count is inaccurate, and the DDR routine cannot swap the volume unless the device is magnetic tape with a cartridge.

If the operand is omitted, N is assumed.

EXCP Interface Parameters

If you supply these parameters, they must be supplied before the end of the DCB OPEN exit routine. For more information on EXCP appendages, see “Making Appendages Available to the System” on page 128.

EOEA=*symbol*
Last two characters of an EOE appendage name that you have entered into SYS1.LPALIB or SYS1.SVCLIB.

PCIA=*symbol*
Last two characters of a PCI appendage name that you have entered into SYS1.LPALIB or SYS1.SVCLIB.

SIOA=*symbol*
Last two characters of a SIO appendage name that you have entered into SYS1.LPALIB or SYS1.SVCLIB.

PGFIX=YES
The SIO appendage includes a page-fix entry point. Refer to “Executing Fixed Channel Programs in Real Storage (EXCPVR)” on page 108 and “Page Fix and EXCPVR Start I/O Appendage” on page 130.

CENDA=*symbol*
Last two characters of a CHE appendage name that you have entered into SYS1.LPALIB or SYS1.SVCLIB.

XENDA=*symbol*
Last two characters of an ABE appendage name that you have entered into SYS1.LPALIB or SYS1.SVCLIB.

OPTCD=Z
indicates that, for devices that have a magnetic tape reel (input only), a reduced error recovery procedure (4 reads only) will occur when a data check is encountered. Only specify this operand when the tape is known to contain errors and the application does not require that all records be processed. Its proper use would include error frequency analysis in the SYNAD routine. Specification of this parameter will also cause generation of a foundation block extension. This parameter does not apply to magnetic tape subsystems that use cartridges. Your program can change this parameter value at any time.

IMSK=*value*
Specification indicates that the system will not use the system's error routines.

Foundation Block Extension and Common Interface Parameters

EXLST=*address*
the address of an exit list that you have written for exception conditions. The format and purpose of the exit list are provided in *DFSMS/MVS Using Data Sets* .

EODAD=*address*
the address of your end-of-data-set routine for input data sets. If this routine is not available when it is required, the task is abnormally terminated.

It is required when you issue an EOV macro for the last volume containing data and bit 0 of DCBOFLGS indicates that the last operation was a read. (See “End of Volume” on page 106 for more information concerning EOV.) An exception to this is when another data set is concatenated after the current data set.

The EODAD address used here is a 24-bit address in the DCB. If you use the DCB extension (DCBE), its 31-bit EODAD address is used.

DSORG=PS or **PO** or **DA** or **IS**
the data set organization (one of the following codes). Each code indicates that the format of the device-dependent portion of the DCB is to be similar to that generated for a particular access method:

Code	DCB Format:
PS	QSAM or BSAM
PO	BPAM
DA	BDAM
IS	QISAM or BISAM

When writing in a partitioned or sequential DASD data set, you can cause EOV or CLOSE to write a file mark. Refer to “Device-Dependent Parameters” on page 116 for further information.

Do not issue the STOW macro against an EXCP DCB; it can corrupt the directory contents.

IOBAD=*address*
the address of an input/output block. You can use this field for any purpose.

RECFM=*code*
the record format of the data set. (Record format codes are given in *DFSMS/MVS Macro Instructions for Data Sets*.) When writing a data set to be read later, RECFM, LRECL, and BLKSIZE should be specified to identify the

data set attributes. LRECL and BLKSIZE can only be specified in a DD statement, in an SVC 99 call, or in the JFCB, because these fields do not exist in a DCB used by EXCP.

IBM recommends that when you are creating a data set you issue a RDJFCB macro and set JFCLRECL and JFCBLKSI before issuing OPEN TYPE=J. This will allow LRECL and BLKSIZE to be in the data set label.

The RECFM parameter is not used by the EXCP routines. It provides information stored for subsequent use by access methods that read or update the data set.

The following parameters are optional. The system does not manage the buffers described.

BFALN=F or **D**
the word boundary alignment of each buffer, either word or doubleword.

BUFL=*length*
the length in bytes of each buffer; the maximum length is 32760.

BUFCB=*address*
the address of a buffer pool control block, that is, the 8-byte field preceding the buffers in a buffer pool.

BUFNO=*number*
the number of buffers assigned to the associated data set; the maximum number is 255. For an EXCP DCB, OPEN ignores this parameter and the BFALN, BUFL, and BUFCB parameters. They have an effect only if your program uses those fields or if your program issues a GETPOOL or GETBUF macro.

Buffer number and block size affect the data transfer rate and the operating system overhead per block. Using more buffers reduces (per block transferred) the system overhead and the time needed for the DASD to select a cylinder and rotate to the requested record (device latency time). If you allocate more buffers than your program can process effectively, the virtual pages containing those buffers might be paged out, adding to the system overhead for the job. A large number of buffers also causes a large amount of real storage to be allocated to the job while the data is being transferred.

A job in a low-performance group can get swapped out more frequently than a higher-priority job. The number of buffers allocated for the job effect the number of pages that have to be swapped out.

Programs that access data sets with small block size (for example, 80) can easily make effective use of 30 buffers, which fit in, at most, two 4096-byte pages. The use of 30 buffers in this case is more efficient than 5, resulting in only 1 channel program rather than 6 channel programs to transfer 30 blocks.

Using data sets with large blocking factors such as half-track blocking on a 3380 or 3390 can be effective if only three or four buffers are specified, rather than five or more. The slightly lower DASD performance and small increase in system instruction costs should be more than offset by a reduction in paging or swapping in a constrained environment.

The DCB OPEN installation exit can use installation criteria for a default buffer number for EXCP DCBs (for a description of the OPEN installation exit, see

DFSMS/MVS Using Data Sets). Your program can use the BUFNO value that the OPEN installation exit might set.

Device-Dependent Parameters

DCBE=

the DCBE= parameter is a device independent parameter but is included here because if DCBE= is specified the entire device-dependent section of the DCB is generated. DCBE= is specified when the DCB extension (DCBE) is required. The first word of the DCB (at offset +0) points to the DCBE when 2 bits at offset 32 (X'20') are on as follows:

Name	Bit
DCBH1	X'80'
DCBH0	X'04'

Coding DCBE= in the DCB macro sets these 2 bits on. For further DCBE information see “Data Control Block Extension” on page 120. See *DFSMS/MVS Macro Instructions for Data Sets* for more information on the DCBE= parameter of the DCB macro. See also “Data Control Block Extension” on page 120.

DEVD=*code*

the device in which the data set might reside. The codes are listed in order of descending space requirements for the DCB:

Code	Device
DA	Direct access
TA	Magnetic tape
PR	Printer
PC	Card punch
RD	Card reader

If you do not want to select a specific device until job setup time, specify the device type requiring the largest area; that is, DEVD=DA.

The following diagrams illustrate the device-dependent portion of the DCB for each combination of device type specified in the DEVD parameter and data set organization specified in the DSORG parameter. Fields that correspond to device-dependent parameters in addition to DEVD are indicated by the parameter name.

When processing concatenated data sets, the system changes the value in DCBDEVT as appropriate as it reaches each new data set. These values are described in *DFSMS/MVS Macro Instructions for Data Sets*.

<table>
<tr><td colspan="3">0
DCBDCBE</td></tr>
<tr><td>4
Reserved</td><td colspan="2">5
(DCBFDAD)</td></tr>
<tr><td colspan="3">8</td></tr>
<tr><td></td><td colspan="2">13
(DCBDVTBL)</td></tr>
<tr><td>16
DCBKEYLE</td><td>17
(DCBDEVT)</td><td>18
(DCBTRBAL)</td></tr>
</table>

Note: The fields in parentheses represent information not associated with parameters of the DCB macro instruction. They are set by OPEN and EOV. Your program can modify DCBFDAD, DCBKEYLE, or DCBTRBAL as described in the text.

Figure 37. Device-dependent portion of the DCB with DEVD=DA and DSORG=PS (or DSORG=PO)

When writing on DASD, maintain certain fields of the device-dependent portion of the DCB. The system uses the information:

- To write a file mark for output data sets
- When releasing unused space at the end of the allocated area

  You can request partial space release via the management class, RLSE on the DD SPACE parameter, TSO/E ALLOCATE command RELEASE parameter or PARTREL macro.
- When adding to the data set with DISP=MOD or the EXTEND or OUTINX options of OPEN
- When DFSMSdss copies the data set.

These are the fields to maintain:

- The track balance (DCBTRBAL) field that contains a 2-byte unsigned binary number that indicates the remaining number of bytes on the current track. It is recommended that your program does not directly calculate this number but uses the TRKCALC macro (see “Performing Track Calculations” on page 198). Both the calculation values and the algorithm differ among device types.
- The full disk address (DCBFDAD) field that indicates the location of the current record. The address is in the form MBBCCHHR. For the actual form of MBBCCHR, see Figure 48 on page 136.

If space is available for output data sets, the system uses the contents of the full disk address (DCBFDAD) field, plus one, to write a file mark when the DCB is closed or EOV is issued. If the track balance field (DCBTRBAL) is less than 8, the file mark is written on the next sequential track.

Note that the track containing your last data block, as identified by DCBFDAD, might not be the best place for a file mark. Consider the following:

- If the file mark is near the end of the track and a future user extends the data set using DISP=MOD on the DD statement or using the OPEN EXTEND or OUTINX option, then the first new block might be on the next track. This leaves a file mark inside the data.
- For compatibility with BSAM and QSAM when you are writing fixed-standard records, you should cause the file mark to be written wherever the next block would have been written, as if all blocks were full size. The file mark should not be “squeezed” on to the current track.

If the system is to write a file mark, you must maintain the contents of these two fields and set on bit 0 of DCBOFLGS. For further information on DCBOFLGS, see “End of Volume” on page 106. Use the OPEN macro instruction to initialize DCBDVTBL and DCBDEVT. You can use DCBDVTBL with the DEVTAB parameter of the TRKCALC macro (see “Performing Track Calculations” on page 198 for the TRKCALC description).

16 DCBKEYLE	17 Reserved

Figure 38. Device-dependent portion of the DCB with DEVD=DA and DSORG=DA

12 (DCBBLKCT)			
16 DCBTRTCH	17 DCBDEVT	18 DCBDEN	19 Reserved

Note: The fields in parentheses represent information not associated with parameters of the DCB macro instruction. They are set by OPEN and EOV. Your program can modify DCBBLKCT as described for the REPOS parameter above.

Figure 39. Device-dependent portion of the DCB with DEVD=TA and DSORG=PS

The system uses the contents of the block count (DCBBLKCT) field to write the block count in trailer labels when the DCB is closed or when the EOV macro instruction is issued. For tape cartridges, the system also compares this count with a count calculated from hardware information. OPEN and EOV set this DCB field to zero except when reading standard labeled tape backward. In that case OPEN or EOV set DCBBLKCT to the block count in the trailer label.

Note: The I/O process increments this field by the contents of the IOBINCAM field of the IOB upon completion of each I/O request.

When using EXCP to process a tape reel data set open at a checkpoint, it is necessary to maintain the correct count; otherwise, the data set could be incorrectly positioned when restart occurs. If you code REPOS=Y, you must maintain the count for repositioning during dynamic device reconfiguration. Do not modify the DCB.

16 DCBPRTSP	17 Reserved

Figure 40. Device-dependent portion of the DCB with DEVD=PR and DSORG=PS

16 DCBMODE DCBSTACK	17 Reserved

Figure 41. Device-dependent portion of the DCB with DEVD=PC or RD and DSORG=PS

The following DCB operands pertain to specific devices and can be specified only when the DEVD parameter is specified.

KEYLEN=*length*
for direct access devices, the length in bytes of the key of a physical record, with a maximum value of 255. When a block is read or written, the number of bytes transmitted is the key length plus the record length. This parameter does not directly affect EXCP processing, but is stored in the data set label.

DEN=*value*
for magnetic tape, the tape recording density in bits per inch is shown in the following table:

Value	Density
2	800 (NRZI)
3	1600 (PE)
4	6250 (GCR)

NRZI—Non return-to-zero change to ones recording PE—phase encoded recording GCR—group coded recording

If this parameter is omitted, the highest density available on the device is assumed. Refer to *DFSMS/MVS Macro Instructions for Data Sets* for further information on DEN.

TRTCH=*value*
for magnetic tape subsystems with Improved Data Recording Capability, the tape recording techniques consist of:

Value	Tape recording technique
COMP	Record data in compacted format.
NOCOMP	Record data in standard uncompacted format.

for 7-track magnetic tape, the tape recording technique:

Value	Tape recording technique
C	Data conversion feature is available.
E	Even parity is used. (If omitted, odd parity is assumed.)
T	BCDIC to EBCDIC translation is required.

MODE=*value*
for a card reader or punch, the mode of operation. Either C (column binary mode) or E (EBCDIC code) can be specified. This field and parameter do not

directly affect EXCP processing but your program can use the field. This is useful to allow you to specify the value on the DD statement.

STACK=*value*
for a card punch or card reader, the stacker bin to receive cards, either 1 or 2. This field and parameter do not directly affect EXCP processing but your program can use the field. This is useful to allow you to specify the value on the DD statement.

PRTSP=*value*
for a printer, the line spacing is 0 — 3. This field and parameter do not directly affect EXCP processing but your program can use the field. This is useful to allow you to specify the value on the DD statement.

DSORG Parameter of the DCBD Macro

In addition to the operands described in *DFSMS/MVS Macro Instructions for Data Sets* for the DSORG parameter of the DCBD macro, you can specify the following operand:

DSORG=XA or **XE**
Specify the section of the DCB to be mapped.

XA
Specifies a DCB with the foundation block and EXCP interface.

XE
Specifies a DCB with the common interface, foundation block extension, and foundation block.

Code DSORG=(XA,XE) to map all four sections of the DCB.

Data Control Block Extension

The DCBE provides further processing options. The only option currently supported by EXCP is the EODAD parameter. Note that the DCBE can be above the 16MB line even though your user program is running in 24-bit mode. See *DFSMS/MVS Macro Instructions for Data Sets* for more information concerning the DCBE.

Input/Output Block Fields

The input/output block (IOB) is not automatically constructed by a macro instruction; it must be defined as a series of constants and be on a word boundary. For unit-record and tape devices, the IOB is 32 bytes long. For direct access, teleprocessing, and graphic devices, 8 additional bytes must be provided. You can use the system mapping macro IEZIOB, which expands into a DSECT, to help in constructing an IOB. IEZIOB fields that are not described here are not part of the programming interface.

In Figure 42 on page 121 the shaded areas indicate fields in which you must specify information. The other fields are used by the system and must be defined as all zeros. You cannot place information into these fields, but you can examine them.

There are certain IOB fields that you do not have to set to any particular value before issuing EXCP because the system itself sets them. They include:

- IOBSENS0

- IOBSENS1
- IOBECBCC
- IOBCSW
- IOBSIOCC

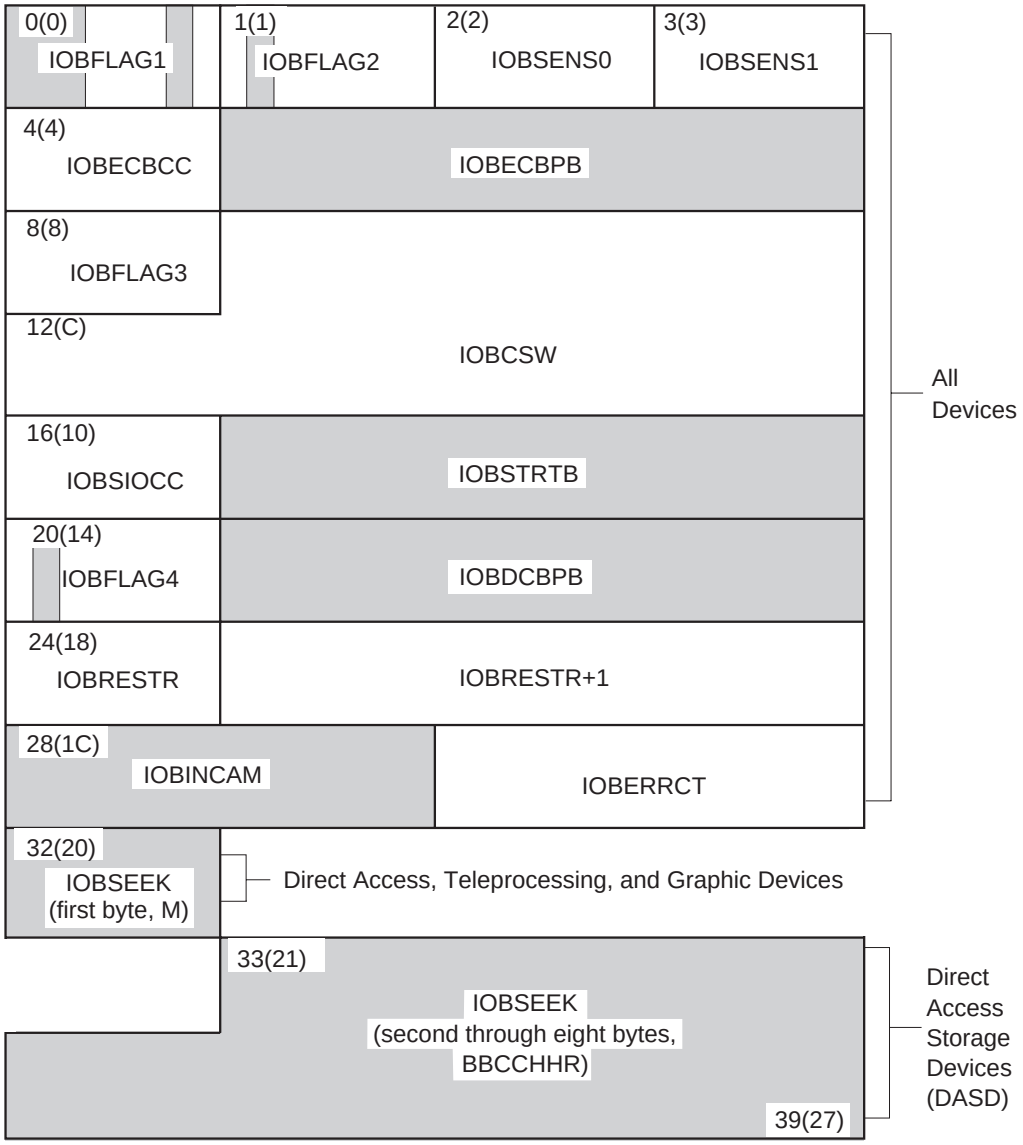


Figure 42. Input/Output Block (IOB) Format

IOBFLAG1 (1 byte)

Set bit positions 0, 1, and 6. One-bits in positions 0 and 1 indicate data chaining and command chaining, respectively. (If you specify both data chaining and command chaining, the system does not use error recovery routines except for the direct access devices.) If an I/O error occurs while your channel program executes, a failure to set the chaining bits in the IOB that correspond to those in the CCW might make successful error recovery impossible. The integrity of your data could be compromised.

A one-bit in position 6 indicates that the channel program is not a related request; that is, the channel program is not related to any other channel

program. See bits 2 and 3 of IOBFLAG2 below. If you intend to issue an EXCP macro with a BSAM, QSAM, or BPAM DCB, you should turn on bit 7 to prevent access-method appendages from processing the I/O request.

IOBFLAG2 (1 byte)
If you set bit 6 in the IOBFLAG1 field to zero, bits 2 and 3 in this field must then be set to:

- 00, if any channel program or appendage associated with a related request might modify this IOB or channel program.
- 01, if the conditions requiring a 00 setting do not apply, but the CHE or ABE appendage might retry this channel program if it completes normally or with the unit-exception or wrong-length-record bits on in the CSW.
- 10 in all other cases.

The combinations of bits 2 and 3 represent related requests, known as type 1 (00), type 2 (01), and type 3 (10). The type you use determines how much the EXCP processor can overlap the processing of related requests. Type 3 allows the greatest overlap, normally making it possible to quickly reuse a device after a channel-end interruption. (Related requests that were executed on a pre-MVS system are executed as type-1 requests if not modified.)

IOBSENS0 and IOBSENS1 (2 bytes)
are set by the EXCP processor when a unit check occurs. These are the first two sense bytes. Occasionally, the system is unable to obtain any sense bytes because of unit checks when sense commands are issued. In this case, the system simulates sense bytes by moving `X'10FE'` to IOBSENS0 and IOBSENS1.

If you wish to retrieve more than two sense bytes, supply an IOBE and IEDB as described in “Interruption Handling and Error Recovery Procedures” on page 102.

IOBECBCC (1 byte)
the first byte of the completion code for the channel program. The system places this code in the high-order byte of the event control block when the channel program is posted complete. The completion codes and their meanings are listed under “Event Control Block Fields” on page 124.

IOBECBPT (3 bytes)
the address of the 4-byte event control block (ECB) you have provided.

IOBFLAG3 (1 byte)
used by the system.

IOBCSW (7 bytes)
the low-order 7 bytes of the simulated channel status word that are placed into this field each time a channel-end or PCI interruption occurs. The first three bytes are the virtual address pointing after the last-executed CCW in your channel program. The next two bytes are status bytes. The last two bytes is the residual count, which is the number of bytes not transferred in the last CCW.

IOBSIOCC (1 byte)
in bits 0 and 1, the instruction-length code; in bits 2 and 3, the start subchannel (SSCH) condition code for the instruction the system issues to start the channel program; and, in bits 4 through 7, the program mask.

IOBSTART (3 bytes)
the starting address of the channel program to be executed.

IOBFLAG4 (1 byte)
Set bit 3 (IOBCEF) to indicate whether you are supplying an IOB common extension (IOBE). If this bit is 1, then register 0 contains the IOBE address when you issue EXCP. Refer to “Requesting Extended Error Information” on page 103 and “Requesting Different Levels of ERP Processing” on page 104.

IOBDCBPT (3 bytes)
the address of the DCB of the data set to be read or written by the channel program.

Reserved (1 byte)
used by the system.

IOBRESTR+1 (3 bytes)
if a related channel program is permanently in error, this field is used to chain together IOBs that represent dependent channel programs. To learn more about the conditions under which the chain is built, see “Purging and Restoring I/O Requests” on page 194.

IOBINCAM (2 bytes)
for magnetic tape, the amount by which the system increments the block count (DCBBLKCT) field in the device-dependent portion of the DCB. You can alter these bytes at any time. For forward operations, these bytes should contain a binary positive integer (usually +1); for backward operations, they should contain a binary negative integer. When these bytes are not used, all zeros must be specified. See Figure 39 on page 118.

IOBERRCT (2 bytes)
used by the system.

IOBSEEK (first byte, M)
for direct access devices, the extent entry in the data extent block that is associated with the channel program (0 indicates the first entry; 1 indicates the second, and so forth). For teleprocessing and graphic devices, it contains the UCB index.

IOBSEEK (last 7 bytes, BBCCHHR)
for direct access devices, the seek address for your channel program.

Input/Output Block Common Extension Fields

You can construct an IOB common extension (IOBE) block to receive extended error information or to control the level of error recovery procedure (ERP) processing. To provide an IOBE, set IOBCEF in IOBFLAG4 on and set the IOBE address in register 0 when you issue EXCP or EXCPVR. (The IOBE is mapped by IOSDIOBE macro).

Set bytes 0–3 to the constant ‘IOBE’ and byte 4 to `X'01'`. Set reserved bytes to `X'00'`. Set byte 7 as described in “Requesting Different Levels of ERP Processing” on page 104. If you want extended error information, set bytes 12 through 15 to the address of the IEDB. If you do not want extended error information, set bytes 12 through 15 to 0. Bytes 8 to 11 are available for you to use. Refer to “Requesting Extended Error Information” on page 103 for more information on using the IOBE.

The IOBE must reside below 16MB virtual. See Figure 43 on page 124 for a diagram of this control block.

0 IOBEID		
4 IOBEVERS	5-6 Reserved	7 IOBEERPM
8-11 IOBEUSER, IOBEUPTR		
12-15 IOBEIEDB		
16-47 Reserved		

Figure 43. Format of an IOBE

Input/Output Error Data Block Fields

The system uses the IEDB to provide extended error information. You provide an IEDB by setting its address in an IOBE. Set bytes 0–3 to the constant ‘IEDB’. Set byte 4 to X'01'. Set reserved fields to X'00'. The system moves the sense bytes to IEDBSNS. If fewer than 32 sense bytes are available, there might be residual data. The IEDB must reside below 16MB virtual. Refer to “Requesting Extended Error Information” on page 103 for more information on using the IEDB. See Figure 44 for a diagram of this control block.

0 IEDBID			
4 IEDBVERS	5 IEDBFLG1	6 IEDBCOD	7 Reserved
8-39 IEDBSNS			
40-47 Reserved			

Figure 44. Format of an IEDB

Event Control Block Fields

Define an event control block (ECB) as a 4-byte area on a word boundary. When the channel program has been completed, the system places a completion code containing status information into the ECB (Figure 45 on page 125). Before examining this information, test for the setting of the complete bit. If the complete bit is not on, and your problem program cannot perform other useful operations, issue a WAIT or EVENTS macro instruction that specifies the ECB. Do not construct a program loop to test for the complete bit.

You do not have to set any particular values in the ECB before issuing EXCP or EXCPVR because the system clears it.

WAIT bit=0	COMPLETE bit=1	Remainder of completion code

bit
0 1 2 31

Figure 45. Event Control Block after Posting of Completion Code

Wait bit
One in this position indicates that the WAIT or EVENTS macro instruction has been issued, but the channel program has not been completed.

Complete bit
A one in this position indicates that the channel program has been completed. If it has not been completed, a zero bit is in this position.

Completion code
This code, which includes the wait and complete bits, might be one of the following 4-byte hexadecimal expressions. If an appendage posts the ECB, it might use other 4-byte codes in which the first two bits are 01. Refer to "Start-I/O Appendage" on page 130 and "Channel-End Appendage" on page 136.

Code	Meaning
41	Permanent I/O error
42	Extent error (DASD only). Either IOBSEEK lies outside the extents described by the DEB or the channel program attempted execution outside the current extent.
44	An error occurred after the previous I/O request to the device was posted complete. The appendages and (if allowed to execute) the ERP have determined that this is a permanent error. The I/O request is terminated with the permanent error. The CSW contents and sense data in the IOB do not apply to the attempted operation. They apply to the previous operation attempted for any data set on the device. You can reissue the EXCP macro instruction to restart the channel program.
45	A program check or machine check occurred in IOS while processing the I/O request.
48	The channel program was purged.
4B	An error occurred during tape repositioning that was requested by the tape ERP.
4F	Error recovery routines were entered following a direct access error but are unable to read the home address or record 0.
51	Simulated error status. The appendages and, if allowed to execute, the ERP have determined that this is a permanent error. The I/O request is terminated with the permanent error. This code indicates that the device is in a permanent error state, boxed, or not connected. The code can indicate also

that a missing interrupt was detected and that the I/O operation was terminated as a result of recovery operations by the missing interrupt handler.

74 Simulated error status. This code is set as a result of attempting to start an I/O operation to a device that is in a permanent error state, boxed, or not connected. This code is also set if a missing interrupt was detected and the I/O operation was terminated as a result of recovery operations by the missing interrupt handler. The system has set this code temporarily in IOBECBCC and IEDB to invoke the ERP, if allowed to execute, and the appendages to determine if the error is permanent or correctable. If the error is determined to be permanent the system will change the value to 51. The system does not set the 74 code in the ECB.

7F Normal I/O completion, but does not appear in the ECB.

Data Extent Block Fields

The data extent block is constructed by the system when an OPEN macro instruction is issued for the DCB. You cannot modify the fields of the DEB, but you can examine them. The DEB is mapped by the IEZDEB macro. The DEB fields used for EXCP and EXCPVR are illustrated in Appendix C, “Control Blocks” on page 419 (all the DEB fields are illustrated in *DFSMS/MVS DFSMSdfp Diagnosis Reference*).

EXCP and EXCPVR Appendages

An appendage is a routine that provides additional control over I/O operations. Using appendages, you can examine the status of I/O operations and determine the actions to be taken for various conditions. An appendage can receive control as shown in Figure 46.

Figure 46. EXCP Appendages

Appendage	Description	When Called
ABE	Abnormal-end	Abnormal conditions
CHE	Channel-end	Channel-end, unit exception, wrong-length record
EOE	End-of-extent	Address in I/O block outside allocated extent limits
PCI	Program-controlled interruption	When one or more PCI bits are on in a channel program
PGFX	Page-fix	Prior to SIO for EXCPVR requests
SIO	Start-I/O	Just prior to translating channel program

Appendages get control in supervisor state, receiving the pointers from the EXCP processor described in the following table. The appendages receive control in 24-bit addressing mode and must return in the same mode.

Register	Contents
1	Points to the request queue element
2	Points to the input/output block

3 Points to the data extent block

4 Points to the data control block

6 Points to the seek address (MBBCCHHR) if control is given to an end-of-extent appendage

7 Points to the unit control block. On the current level of the system, this is always below the 16MB line. If you use the nocapture option of dynamic allocation and the actual UCB is above the line, then OPEN or EOV captures the UCB until EOV or CLOSE.

13 Points to a 16-word area you can use to save input registers or data

14 Points to the location in the system where control is to be returned following execution of an appendage. When returning control to the system, you can use displacements from the return address in register 14. Allowable displacements are summarized and described later for each appendage in Figure 47.

15 Points to the entry point of the appendage. When the PGFX appendage is entered, points to the SIO entry point.

The processing done by appendages is subject to these requirements and restrictions:

- Register 9, if used, must be set to binary zeros before control is returned to the system. All other registers, except those indicated in the descriptions of the appendage, must be saved and restored if you use them. Figure 47 summarizes register conventions.
- No SVC instructions or instructions that change the status of the system (for example, WTO, LPSW, or similar privileged instructions) can be issued.
- Loops testing for the completion of I/O operations cannot be used.

Appendage types are described in the following sections, with explanations of when they are created, how they return control to the system, and which registers they can use without saving and restoring their contents. If you do not supply a particular appendage, or supply no appendage, the system acts as though that appendage had returned at offset 0 from register 14.

Figure 47 (Page 1 of 2). Entry Points, Returns, and Available Work Registers for Appendages

Appendage	Entry Point	Returns		Available Work Registers
EOE	Reg 15	Reg 14 + 0 Reg 14 + 4 Reg 14 + 8	Call ABE Skip Try Again	Reg. 10, 11, 12, and 13
SIO	Reg 15	Reg 14 + 0 Reg 14 + 4	Normal Skip	Reg. 10, 11, and 13
PCI	Reg 15	Reg 14 + 0	Normal	Reg. 10, 11, 12, and 13
PGFIX	Reg 15+4	Reg 14 + 0	Normal	Reg. 10, 11, and 13

Figure 47 (Page 2 of 2). Entry Points, Returns, and Available Work Registers for Appendages

Appendage	Entry Point	Returns		Available Work Registers
CHE	Reg 15	Reg 14 + 0 Reg 14 + 4 Reg 14 + 8 Reg 14 + 12	Normal Skip Re-EXCP By-Pass	Reg. 10, 11, 12, and 13
ABE	Reg 15	Reg 14 + 0 Reg 14 + 4 Reg 14 + 8 Reg 14 + 12	Normal Skip Re-EXCP By-Pass	Reg. 10, 11, 12, and 13

Note: The register conventions for passing parameters from appendages to the system are described in the individual appendage descriptions.

Making Appendages Available to the System

This section provides guidance information on making appendages available to the system so that they can be executed.

Prior to execution, appendages must be members of either the SYS1.LPALIB or SYS1.SVCLIB data set. To put appendages into SYS1.LPALIB or SYS1.SVCLIB, link-edit them into these data sets after the system has been built. Each appendage must have an 8-character member name, the first six characters being IGG019 and the last two being anything in the range from WA to Z9. If your program runs in a V=R address space and uses a PCI appendage, the appendage and any routine that the PCI appendage refers to must be placed in either SYS1.SVCLIB or the fixed link pack area (LPA). For information on providing a list of programs to be fixed in storage, see *OS/390 MVS Initialization and Tuning Guide*.

The Authorized Appendage List (IEAAPP00)

If an unauthorized program opens a DCB to be used with an EXCP macro instruction, the names of any appendages associated with the DCB must be listed in the IEAAPP00 member of SYS1.PARMLIB. (An unauthorized program is one that runs in a protection key greater than 7 and has not been marked as authorized by the Authorized Program Facility.) Once you have added your appendages to SYS1.LPALIB or SYS1.SVCLIB after the system was built, you can add IEAAPP00 to SYS1.PARMLIB and put the names of the appendages in it in one job step with the IEBUPDTE utility or with another program that updates partitioned data sets.

The following example shows JCL statements and IEBUPDTE input that add IEAAPP00 to SYS1.PARMLIB and put the names of one EOE appendage, two SIO appendages, two CHE appendages, and one ABE appendage in IEAAPP00:

```
//              JOB      ...
//              EXEC     PGM=IEBUPDTE,PARM='NEW'
//SYSPRINT      DD       SYSOUT=A
//SYSUT2        DD       DSN=SYS1.PARMLIB,DISP=SHR
//SYSIN         DD       *
./ ADD NAME=IEAAPP00
EOEAPP WA,
SIOAPP X1,X2,
CHEAPP Z3,Z4,
ABEAPP Z2
./ ENDUP
/*
```

Note the following about the IEBUPDTE input:

- The type of appendage is identified by six characters that begin in column 1. EOEAPP identifies an EOE appendage, SIOAPP an SIO appendage, CHEAPP a CHE appendage, and ABEAPP an ABE appendage. (The PCI appendage identifier, PCIAPP, is not shown, because the example does not add a PCI appendage name to IEAAPP00.)
- Only the last two characters in an appendage's name are specified, beginning in column 8.
- Each statement that identifies one or more appendage names ends in a comma, except the last statement.

You can also use IEBUPDTE to add appendage names later or to delete appendage names. The following example shows JCL statements and IEBUPDTE input that adds the names of a PCI and an ABE appendage to the IEAAPP00 appendage list created in the preceding example, and deletes the name of an SIO appendage from that list:

```
//              JOB. . .
//              EXEC     PGM=IEBUPDTE,PARM='NEW'
//SYSPRINT      DD       SYSOUT=A
//SYSUT2        DD       DSN=SYS1.PARMLIB,DISP=SHR
//SYSIN         DD       *
./ ADD NAME=IEAPP00
PCIAPP Y1,
EOEAPP WA,
SIOAPP X1,
CHEAPP Z3,Z4,
ABEAPP Z2,Z4
./ ENDUP
/*
```

Note the following about the IEBUPDTE input:

- The command to IEBUPDTE is ADD but a replace occurs because PARM='NEW' is specified.
- All the appendage names that are to remain in IEAAPP00 are repeated.
- IGG019Z4 is both a CHE and an ABE appendage.

Start-I/O Appendage

Unless an ERP is in control, the system passes control to the SIO appendage just before the system translates and starts your channel program. It is called even if the channel program is not later translated. Your SIO appendages can build the channel program. The system does not test IOBSTART until after the SIO appendage returns.

Optional return vectors give the I/O requester the following choices:

Reg. 14 + 0 — Normal return. Normal channel program translation and initiation of I/O.
Reg. 14 + 4 — Skip the I/O operation. The channel program is not initiated. The channel program is not posted complete. You can post the channel program complete by using the POST macro. For more information on the POST macro, see *OS/390 MVS Authorized Assembler Services Guide*and *OS/390 MVS Authorized Assembler Services Reference LLA-SDU* .

The following steps cause the POST macro to post the channel program complete.

1. Save necessary registers.
2. Set the completion code in register 10. These are the four bytes of an ECB.
3. From the IOB, set the ECB address in register 11.
4. Issue the POST macro, `POST (11),(10),LINKAGE=BRANCH`
5. On return, reestablish necessary registers.

Page Fix and EXCPVR Start I/O Appendage

This appendage is a combination of two independent appendages. The complete appendage is a reenterable subroutine with two entry points, one for the SIO appendage and one for the PGFX appendage.

The SIO entry point is located at offset +0 in the SIO subroutine; from this entry point you might have an instruction to branch to any other location in the appendage. The entry point of the PGFX appendage is at offset +4 in the SIO subroutine. The address of offset +0, the SIO appendage entry address, is set in register 15 as the entry point of the PGFX appendage, allowing you to use the same entry linkage code for both entry points.

PGFX Appendage

This appendage lists the areas that must be fixed to prevent paging exceptions during the execution of the current I/O request. While this appendage can be entered more than once for one I/O request, each time it is entered it must create the same list of areas to be fixed. The appendage can use the 16-word save area pointed to by register 13. Registers 10, 11, and 13 can be used as work registers.

Each page-fix entry placed in the list by the appendage must have the following doubleword format:

0	1 … 31	32	33 … 63
0	Starting virtual address of area to be fixed	0	Ending virtual address of area to be fixed + 1

On return from the PGFX appendage to the system (via the return address provided in register 14), register 10 must point to the first page-fix entry and register 11 must contain the number of page-fix entries in the work area. The system then fixes the pages corresponding to the areas listed by the PGFX appendage. The pages remain fixed until the associated I/O request terminates.

If either the channel end appendage or the abnormal end appendage returns via the return address in register 14 plus 8, the PGFX appendage is not normally reentered. Instead, the SIO appendage is entered, and the page-fix list built by the PGFX appendage is still active. When a PURGE macro has been issued (for instance, when a storage swap has occurred), the PGFX appendage is entered after either the channel end appendage or the abnormal end appendage returns via the return address in register 14 plus 8. When I/O is restored, the PGFX appendage is entered.

Note: The page-fix list must be in page-fixed storage.

SIO Appendage

If you are using EXCPVR to execute a channel program, translate the virtual addresses in the operands of your channel program to central storage addresses. This should be done in the SIO appendage. If indirect data addressing is required, use the SIO appendage to build the indirect data address lists (IDALs) and turn on the IDA indicators in the associated CCWs.

EXCPVR may be used for VIO data sets with the following considerations for the SIO appendage:

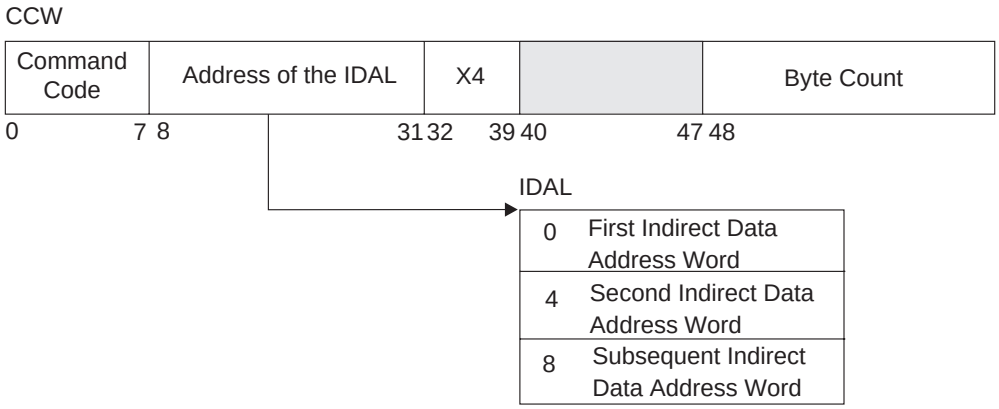
- The use of IDAWs and an IDAL is not required.
- Addresses in the CSWs, IDAWs, or an IDAL must be virtual. They must not be converted to central storage addresses.

The UCBVRDEB bit in the UCBJBNR byte can be checked to determine if the data set is being processed with VIO.

Translating Virtual Addresses and Building the IDAL: Convert the virtual addresses in the channel program to central storage addresses. Also check the areas whose addresses appear in bits 8 through 31 of your CCWs to determine whether the data areas cross 2K-byte boundaries. If they do, provide an entry in the IDAL for each crossed boundary. The channel subsystem uses the IDAL to identify the address where it will continue reading or writing when a 2K-byte boundary is crossed during a read or write operation. The IDAL must contain central storage addresses when it is processed by the channel.

The LRA instruction returns a 31-bit central storage addressregardless of the addressing mode. EXCP supports IDAW's containing addresses that exceed 16MB. Your program does not need to take any special action. For most efficient use of system resources you will want to code LOC=(ANY,ANY) or LOC=(BELOW,ANY) when obtaining storage with the GETMAIN or STORAGE macro. See *OS/390 MVS*

Assembler Services Reference. Do your page fixing before issuing the LRA. The following illustration shows the relationship between the CCW and the IDAWs.



Notes:

1. After the first entry, put one entry in the IDAL for each 2K-byte page boundary your data area crosses.
2. If the CCW has an IDAL address rather than a data address, bit 37 must be set on to signal this to the channel.
3. The maximum number of entries needed in the IDAL is determined from the count in the CCW as follows:

   Number of IDAL entries=((CCW byte-count – 1)/2048) + 1. (Round up to the next highest integer if the remainder is not zero.)

The number of IDAL entries required depends on the number of 2K-byte boundaries crossed by the data. For example, if your data is 800 bytes long and does not cross a 2K-byte page boundary, no IDAL entries are required. If your data crosses a 4K-byte page boundary, then two IDAL entries are required. If your data is 5000 bytes long, at least two IDAL entries are required. If your data crosses two 4K-byte page boundaries, four IDAL entries are required.

The first indirect address is the central storage address of the first byte of the data area. The second and subsequent indirect addresses are the central storage addresses of the second and subsequent 2K-byte boundaries of the data area. For example, if the data area central storage address is `X'707FF'` and the byte count is `X'1802'`, the IDAL would contain the following central storage addresses (assuming the central storage addresses are contiguous):

```
707FF
70800
71000
```

If the data area central storage address is `X'707FF'` and the byte count is `X'800'`, the IDAL would contain the following addresses:

```
707FF
70800
```

Program-Controlled Interruption Appendage

This appendage is entered if the channel finds one or more program-controlled-interruption (PCI) bits on in a channel program. It can be entered as many times as the channel finds PCI bits on, or more often. Before the appendage is entered, the contents of the subchannel status word are placed in the channel status word field of the input/output block.

A PCI appendage is reentered if an ERP is retrying a channel program in which a PCI bit is on. The IOB error flag is set when the ERP is in control (IOBFLAG1 = X'20'). (For special PCI conditions encountered with command retry, see “Command Retry Considerations” on page 100.)

To post the channel program from a PCI appendage after an EXCP (not EXCPVR), if the step is running ADDRSPC=VIRT (V=V), or an authorized program is running V=R, the procedure described for the SIO appendage is used.

If the step is running ADDRSPC=REAL (V=R) and an authorized program issued the EXCP request or if an EXCPVR request was issued, the PCI appendage uses central storage addresses. Use the following procedure to post the channel program from the PCI appendage. For more information on the POST macro, see *OS/390 MVS Authorized Assembler Services Guide* and *OS/390 MVS Authorized Assembler Services Reference LLA-SDU*.

The POST macro is coded as follows:

```
POST  ecbaddr,compcode,ASCB=addr,ERRET=addr,ECBKEY=key,
      LINKAGE=BRANCH,MEMREL=NO
```

The ERRET routine address must point to a BR 14 instruction. This instruction must be in storage addressable from any address space (for example, CVTBRET) and addressable by 24 bits.

Note: Using this keyword, only registers 9 and 14 are restored when returning from the POST macro.

The following procedure posts the channel program from the PCI appendage.

1. Save necessary registers as registers 9 and 14 are only restored on the return from the POST macro.
2. Set the ECB key in register 0.
3. Set the 4-byte completion code in register 10.
4. Set the ECB address in register 11.
5. Set the error routine address in register 12, by setting it to address of CVTBRET and turn on the high-order bit (X'80') of the high-order byte.
6. Set the ASCB address in register 13.
7. If the ASCB address has not already been saved, the following procedure can be used to obtain the ASCB address:
   a. Issue the EPAR instruction to obtain the ASID. For information on the EPAR instruction, see the publication *ESA/390 Principles of Operation.*
   b. Issue the LOCASCB macro to obtain the ASCB address. The LOCASCB macro is documented in the publication *OS/390 MVS Authorized Assembler Services Reference LLA-SDU*.

8. Issue the POST macro,

   ```
   POST (11),(10),ASCB=(13),ERRET=(12),ECBKEY=(0),LINKAGE=BRANCH
   ```

9. On return, reestablish necessary registers.

To return control to the system for normal operation, use the return address in register 14.

End-of-Extent Appendage

If an end-of-cylinder or file-protect condition occurs, the system updates the seek address to the next higher cylinder or track address and retries the request. If the new seek address is within the data set's extent, the request is executed; if the new seek address is not within the data set's extent, the EOE appendage is entered. To try the request in the next extent, move the new seek address to the location pointed to by register 6.

If a file protect is caused by a full seek (command code=07) embedded within a channel program, the request is flagged as a permanent error, and the ABE appendage is entered.

The end-of-extent (EOE) appendage is entered when the seek address specified in the input/output block is outside the allocated extent limits indicated in the data extent block.

Using the return address in register 14 to return control to the system, causes the ABE appendage to be entered. An end-of-extent error code (`X'42'`) is placed in the ECB code field of the input/output block for subsequent posting in the ECB.

You can use the following optional return addresses:

- Contents of register 14 plus 4: The channel program is posted complete with `X'42'`. The ABE appendage is not reentered for this request.
- Contents of register 14 plus 8: The request is tried again.

Registers 10 through 13 in an EOE appendage can be used without saving and restoring their contents.

Abnormal-End Appendage

To determine the method the system uses to handle an abnormal condition use the abnormal-end (ABE) appendage. The following information explains how to use the ABE appendage.

This appendage can be entered on abnormal conditions, such as unit exception, wrong-length indication, out-of-extent error, intercept condition (that is, device end error), unit check, program check, protection check, channel data check, channel control check, interface control check, and chaining check. It can also be entered when an EXCP is issued for a DCB that has already been purged.

1. If IOBECBCC is set to `X'41'`, this appendage was entered because of a unit exception or wrong-length record indication or both. The system previously called the channel-end appendage, if present. For further information on these conditions, see “Channel-End Appendage” on page 136.

2. If the IOBECBCC is set to X'42', this appendage was entered because of an out-of-extent error. The system previously called the end-of-extent appendage, if present.
3. If this appendage is entered with IOBECBCC set to X'4B', the tape error recovery procedure (ERP) either encountered an unexpected load point, or found zeros in the command address field of the CSW.
4. If the IOBECBCC is set to X'7E', the appendage was first entered because of an intercept condition. If it is then determined that the error condition is permanent, the appendage will be reentered with the IOBECBCC set to X'44'. The intercept condition signals that an error was detected at device end after channel end on the previous request.
5. If the IOBECBCC was set to X'48', the appendage was entered because of an EXCP being issued to an already purged DCB. This applies only to related requests.
6. If the appendage is entered with IOBECBCC set to X'7F', it might be because of a unit check, program check, protection check, channel data check, channel control check, interface control check, or chaining check. If the IOBECBCC is X'7F', it is the first detection of an error in the associated channel program. If the IOBEX flag (bit 5 of the IOBFLAG1) is on, the IOBECBCC field will contain X'41', X'42', X'48', X'4B', or X'4F', indicating a permanent I/O error.

To determine if an error is permanent, check the IOBECBCC field of the IOB. To determine the type of error, check the channel status word field and the sense information in the IOB. However, when the IOBECBCC is X'42', X'48', or X'4F', these fields are not applicable. For X'44', the CSW is applicable, but the sense is valid only if the unit check bit is set.

By using the return address in register 14 to return control to the system, the channel program is posted complete, and its request element is made available. You can use the following optional return addresses:

- Contents of register 14 plus 4: The channel program is not posted complete, but its request element is made available. You can post the channel program by using the calling sequence described under the SIO appendage.
- Contents of register 14 plus 8: The channel program is not posted complete, and its request element is placed back on the request queue to be retried. Reinitialize the IOBFLAG1, IOBFLAG2, and IOBFLAG3 fields of the input/output block and set the IOBERRCT field to zero. As an added precaution, clear the IOBSENS0, IOBSENS1, and IOBCSW fields.
- Contents of register 14 plus 12: The channel program is not posted complete, and its request element is not made available. (Use this return only if the appendage has passed the request queue element to the exit effector for use in scheduling an asynchronous routine.)

Registers 10 through 13 in an ABE appendage can be used without saving and restoring their contents.

Channel-End Appendage

This appendage is entered when a channel end (CHE), unit exception (UE) with or without channel end or when channel end with wrong-length record (WLR) occurs without any other abnormal-end conditions.

By using the return address in register 14 to return control to the EXCP processor, the channel program is posted complete, and its request element is made available. In the case of unit exception or wrong-length record, the ERP is performed before the channel program is posted complete, and the IOBEX flag (X'04') in IOBFLAG1 is set on. The CSW status can be obtained from the IOBCSW field.

If the appendage takes care of the wrong-length record or unit exception or both, it can turn off the IOBEX (X'04') flag in IOBFLAG1 and return normally. The event will then be posted complete (completion code X'7F' under normal conditions, taken from the high-order byte of the IOBECBCC field). If the appendage returns normally without resetting the IOBEX flag to zero, the request will be routed to the associated device ERP. If the ERP could not correct the error, the ABE appendage will be entered with the completion code in IOBECBCC set to X'41'. (See Step 1 of “Abnormal-End Appendage” on page 134.)

You can use the following optional return addresses:

- Contents of register 14 plus 4: The channel program is not posted complete, but its request element is made available. You can post the channel program by using the calling sequence described under the SIO appendage. This is especially useful to post an ECB other than the ECB in the input/output block.
- Contents of register 14 plus 8: The channel program is not posted complete, and its request element is placed back on the request queue so that a channel program at the same or different address can be executed. For correct execution of the channel program, reinitialize the IOBFLAG1, IOBFLAG2, and IOBFLAG3 fields of the input/output block and set the error counts field to zero. As an added precaution, clear the IOBSENS0, IOBSENS1, and IOBCSW fields. You can change IOBSTART before returning. The system will call appendages as for a new EXCP or EXCPVR.
- Contents of register 14 plus 12: The channel program is not posted complete, and its request element is not made available. (Use this return only if the appendage has passed the request queue element to the exit effector for use in scheduling an asynchronous routine. For information on asynchronous exit routines, see *OS/390 MVS Authorized Assembler Services Guide*.)

Registers 10 through 13 in a CHE appendage can be used without saving and restoring their contents.

Converting a Relative Track Address to an Actual Track Address

Convert a relative track address to the actual address by using a resident system conversion routine that can be called in 24 or 31 bit mode. The address of the conversion routine's entry point is in the CVTPCNVT field of the communication vector table (CVT). The address of the CVT is in storage location 16 (field FLCCVT of the PSA data area, mapped by macro IHAPSA).

The conversion routine does all its work in general registers. Load registers 0, 1, 2, 14, and 15 with input to the routine. Register usage is as follows:

Figure 48. Form for MBBCCHR

Register	Use
0	Must be loaded with a 4-byte value of the form TTR*n*, where: TT The track number relative to the beginning of the data set R The block identification on that track *n* The concatenation number of a partitioned data set. (0 indicates the first data set in the concatenation, an unconcatenated partitioned data set, or a non-partitioned data set.)
1	Must be loaded with the address of the data extent block of the data set. Note that the UCB address field is 3 bytes.
2	Must be loaded with the address of an 8-byte area that is to receive the actual address of the block to be processed. The converted address is of the form MBBCCHHR, where: M Indicates which extent entry in the data extent block is associated with the direct access program. (0 indicates the first extent, 1 indicates the second, and so forth) BB Two bytes of zeros CC The cylinder address HH The actual track address R The block number.
3-8	Not used by the conversion routine.
9-13	Used by the conversion routine and not restored.
14	Must be loaded with the address to which control is to be returned after execution of the conversion routine.
15	Used by the conversion routine as a base register and must be loaded with the address where the conversion routine is to receive control (from field CVTPCNVT of the CVT).

Return Codes from the Conversion Routine

When control is returned to your program, register 15 contains one of the following return codes:

Figure 49. Convert Routine Return Codes

Return Code	Description
0 (X'00')	Successful conversion.
4 (X'04')	The relative track address converts to an actual track address outside the extents defined in the DEB.

Converting an Actual Track Address to a Relative Track Address

Convert an actual track address to a relative track address by using a resident system conversion routine that can be called in 24 or 31 bit mode. The address of the conversion routine's entry point is in the CVTPRLTV field of the CVT. The address of the CVT is in storage location 16 (field FLCCVT of the PSA data area, mapped by macro IHAPSA).

The conversion routine does all its work in general registers. Load registers 1, 2, 14, and 15 with input to the routine. Register usage is as follows:

Figure 50. Registers and Their Use for Converting Actual to Relative

Register	Use
0	Loaded with the resulting TTR0 to be passed back to the caller.
1	Must be loaded with the address of the data extent block of the data set.
2	Must be loaded with the address of an 8-byte area containing the actual address to be converted to a TTR. The actual address is of the form MBBCCHHR.
3-8	Not used by the conversion routine.
9-13	Used by the conversion routine and not restored.
14	Must be loaded with the address to which control is to be returned after execution of the conversion routine.
15	Used by the conversion routine as a base register and must be loaded with the address where the conversion routine is to receive control (from field CVTPRLTV of the CVT).

Obtaining the Sector Number of a Block on an RPS Device

For programs that can be used with both RPS and non-RPS devices, test the UCBRPS bit (bit 3 at offset 17 of the UCB) to determine whether the device has rotational position sensing. If the UCBRPS bit is off, do not issue a channel program with a set sector command to the device. The address of the sector conversion routine's entry point is in the CVT0SCR1 field of the CVT.

Your program can call the conversion routine by issuing a BASR 14,15 or BAS 14,16(,15) instruction. If you are passing the track balance to the routine, invoke the routine using a BAS 14,8(15) or BAS 14,20(,15). If you are computing the sector value on modulo devices with variable length records, pass the track balance to the sector convert routine.

The sector convert routine can be called in 24-bit or 31-bit mode at any of its entry points. When calling in 24-bit mode all addresses must point below the 16MB line.

Notes:

1. The sector convert routine does not support PDSEs or extended format data sets. Using the sector convert routine against a PDSE or extended format data set returns results that are inconsistent with the physical record format.
2. The sector convert routine does not support HFS files. You will receive unpredictable results if you use the conversion routine with HFS files.

For RPS devices, the conversion routine does all its work in general registers. Load registers 0, 2, 14, and 15 with input to the routine. Register usage is as follows:

Figure 51. Registers and Their Use for A Sector Convert Routine

Register	Use
0	For fixed, standard blocks or fixed, unblocked records not in a partitioned data set: Register 0 must contain a 4-byte value in the form XXKR, where: XX A 2-byte field containing the physical block size K A 1-byte field containing the key length R A 1-byte field containing the number of the record for which a sector value is desired. To indicate fixed-length records, turn off (set to 0) the high-order bit of register 0. Passing the track balance: Register 0 must contain a 4-byte value of the track balance of the record preceding the required record. For all other cases: Register 0 must contain a 4-byte value in the form BBIR, where: BB The total number of key and data bytes on the track up to, but not including, the target record I A 1-byte key indicator (1 for keyed records, 0 for records without keys) R A 1-byte field containing the number of the record for which a sector value is desired. To indicate variable-length records, turn on (set to 1) the high-order bit of register 0.
1	Not used by the sector-convert routine.
2	When called at offset 0 or 8, must contain a 4-byte field where: The first byte is the UCB device type code for the device (obtainable from UCB+19) The remaining 3 bytes are the address of a 1-byte area that is to receive the sector value. If called at offset 16 or 20, must contain the address of a 1-byte area that is to receive the sector value.
3-8	Not used.
9-10	Used by the convert routine and not saved or restored.
11	If called at offset 0 or 8, not used. If called at offset 16 or 20, contains 1-byte UCBTBYT4 code in the low-order byte. The remaining three bytes must contain zero.
12,13	Not used.
14	Must be loaded with the address in which control is to be returned after execution of the sector conversion routine.
15	Used by the conversion routine as a base register and must be loaded with the address of the entry point to the conversion routine (from field CVT0SCR1 of the CVT).

Chapter 5. Using XDAP to Read and Write to Direct Access Devices

This chapter contains information about the XDAP macro instruction and is provided for compatibility with other IBM operating systems. We recommend using an access method such as VSAM in place of XDAP.

Execute direct access program (XDAP) is a macro instruction used to read, verify, or update a block on a direct access volume. This chapter explains what the XDAP macro instruction does and how it is used. It also discusses the macro instructions used with XDAP and the control blocks that are generated.

Because most of the specifications for XDAP are similar to those for the execute channel program (EXCP) macro instruction, you should be familiar with the information in Chapter 4, “Executing Your Own Channel Programs” on page 95 of this publication, and with the information in the publication *DFSMS/MVS Using Data Sets* that provides how-to information for using the access method routines of the system control program.

If you are not using the standard IBM data access methods, by issuing XDAP you can generate the control information and channel program necessary for reading or updating the records of a data set.

Note: XDAP does not support partitioned data sets extended (PDSEs), sequential extended-format data sets, or SYSIN and SYSOUT data sets.

While XDAP cannot be used to add blocks to a data set, it can be used to change the keys of existing blocks. Any block configuration and any data set organization can be read or updated.

Although the use of XDAP requires less storage than do the standard access methods, it does not provide many of the control program services that are included in the access methods. For example, when XDAP is issued, the system does not block or unblock records and does not verify block length.

Using XDAP

To issue XDAP, provide the actual track address of the track containing the block to be processed. Also provide either the block identification or the key of the block, and specify which of these is to be used to locate the block. If a block is located by identification, both the key and data portions of the block can be read or updated. If a block is located by key, only the data portion can be processed.

For additional control over I/O operations, you can write appendages that must be entered into the LPA library. Descriptions of these routines and their coding specifications are included under Chapter 4, “Executing Your Own Channel Programs” on page 95.

When using the XDAP macro instruction, code a DCB macro instruction in your program to generate a data control block (DCB) for the data set to be read or updated. Also code an OPEN macro instruction that initializes the data control block and produces a data extent block (DEB). The OPEN macro instruction must be executed before any XDAP macro instructions are executed.

© Copyright IBM Corp. 1979, 1999

When the XDAP macro instruction is assembled, a control block and executable code are generated. This control block can be logically divided into three sections:

- An event control block (ECB) that is supplied with a completion code each time the direct access channel program is terminated.
- An input/output block (IOB) that contains information about the direct access channel program.
- A direct access channel program consisting of three or four channel command words (CCWs). The type of channel program generated depends on the parameters of the XDAP macro instruction. When executed, it locates a block by either its actual address or its key and reads, updates, or verifies the block.

When the channel program has terminated, a completion code is placed into the event control block. After issuing XDAP and the direct access program has terminated, regain control by using a WAIT macro instruction to specify the address of the event control block. If volume switching is necessary, issue an EOV macro instruction. Once processing of the data set is completed, issue a CLOSE macro instruction to restore the data control block.

Macro Instructions Used with XDAP

When using the XDAP macro instruction, you must also code DCB, OPEN, CLOSE, WAIT, and, in some cases, the EOV macro instructions. The parameters of the XDAP macro instruction are described in “Executing Direct Access Programs” on page 143. Special requirements or options for the other required macro instructions are explained in the following sections. See “Using Executable Macro Instructions with EXCP” on page 105 for listings of their parameters.

Defining a Data Control Block

Issue a DCB macro instruction for each data set to be read, updated, or verified by the direct access channel program. To learn which macro instruction parameters to code, see “Data Control Block (DCB) Fields” on page 110.

Initializing a Data Control Block

The OPEN macro instruction initializes one or more data control blocks so that their associated data sets can be processed. Issue OPEN for all data control blocks used by the direct access program. Some of the procedures performed when OPEN is executed are:

- Constructing a data extent block
- Transfering information from DD statements and data set labels to the DCB
- Verifying or creating standard labels
- Loading programmer-written appendage routines.

The two parameters of the OPEN macro instruction are the address or addresses of the data control blocks to be initialized and the method of I/O processing of the data set. The processing method can be specified as INPUT, OUTPUT, UPDAT, or EXTEND; however, if nothing is specified, INPUT is assumed. The parameters and different forms of the OPEN macro instruction are described in *DFSMS/MVS Macro Instructions for Data Sets*.

End of Volume

The EOV macro instruction identifies end-of-volume (EOV) and end-of-data-set conditions. For an end-of-volume condition, EOV causes switching of volumes and verification or creation of standard labels. For an end-of-data-set condition (except when another data set is concatenated), EOV causes your end-of-data-set routine to be entered. When using XDAP, issue EOV if switching of direct access volumes is necessary or if secondary allocation is to be performed for a direct access data set opened for output. For details about the parameters of the EOV macro instruction, see “End of Volume” on page 106.

Restoring a Data Control Block

The CLOSE macro instruction restores one or more data control blocks so that processing of their associated data sets can be terminated. Issue CLOSE for all data sets used by the direct access channel program. Some of the procedures performed when CLOSE is executed are:

- Releasing the data extent block
- Removing information transferred to data control block fields when OPEN was executed
- Verifying or creating standard labels
- Releasing programmer-written appendage routines.

The CLOSE macro instruction must identify the address of at least one data control block to be restored, and can specify other parameters. The parameters and different forms of the CLOSE macro instruction are described in *DFSMS/MVS Macro Instructions for Data Sets* .

Executing Direct Access Programs

The XDAP macro instruction produces the XDAP control block (that is, the ECB, IOB, and channel program) and executes the direct access channel program.

The format of the XDAP macro instruction is:

```
>>──┬──────────┬──XDAP──ecb_symbol──,type──,dcb_addr──,area_addr──────────>
    └─label────┘
>──,length_value──,──┬──────────────────────────────────┬──,blkref_addr──>
                     └─(key_addr,keylength_value)───────┘
>──┬──────────────┬──┬──────────────────────────┬──────────────────────><
   └─,sector_addr─┘  └─,MF=──┬─E─┬──┬─────────┬──┘
                             └─L─┘  └─────────┘
```

ecb_symbol—symbol or (2-12)
the symbolic name to be assigned to the XDAP event control block. Registers can be used only with MF=E.

type— **RI** or **RK** or **WI** or **WK** or **VI** or **VK**
the type of I/O operation intended for the data set and the method by which blocks of the data set are to be located. One of the combinations shown must be coded in this field. The codes and their meanings are:

Code	Meaning
R	Read a block.
W	Update a block.
V	Verify that the device is able to read the contents of a block, but do not transfer data.
I	Locate a block by identification. (The key portion, if present, and the data portion of the block are read, updated, or verified.)
K	Locate a block by key. (Only the data portion of the block is read, updated, or verified.) If you code this value, code the *key_addr,keylength-value* operands.

dcb_addr—A-type address or (2-12)
the address of the data control block for the data set.

area_addr—A-type address or (2-12)
the address of an input or output area for a block of the data set.

length_value—absexp or (2-12)
the number of bytes to be transferred to or from the input or output area. If blocks are to be located by identification and the data set contains keys, the value must include the length of the key. The maximum number of bytes transferred is 32 767.

key_addr—RX-type address or (2-12)
when blocks are to be located by key, the address of a virtual storage field that contains the key of the block to be read, updated, or verified.

keylength_value—absexp or (2-12)
when blocks are to be located by key, the length of the key. The maximum length is 255 bytes.

blkref_addr—RX-type address or (2-12)
the address of a field in virtual storage containing the actual track address of the track containing the block to be located. The actual address of a block is in the form MBBCCHHR, where:

M	Indicates which extent entry in the data extent block is associated with the direct access program
BB	Must be zero
CC	The cylinder address
HH	The actual track address
R	The block identification. R is not used if blocks are to be located by key.

(For more detailed information, see “Converting a Relative Track Address to an Actual Track Address” on page 136.)

sector_addr—RX-type address or (2-12)
the address of a 1-byte field containing a sector value. The *sector_addr* parameter is used for rotational position sensing (RPS) devices. The parameter is optional, but its use will improve channel performance. When the parameter is coded, a set-sector CCW (using the sector value indicated by the data address field) precedes the search-ID-equal command in the channel program. The

sector_addr parameter is ignored if the *type* parameter is coded as RK, WK, or VK. If a sector address is specified in the execute form of the macro, then a sector address, not necessarily the same, must be specified in the list form. The sector address in the executable form will be used.

Note: No validity check is made on either the address or the sector value when the XDAP macro is issued. However, a unit check/command reject interruption will occur during channel-program execution if the sector value is not valid for the device or if the *sector_addr* operand is used when accessing a device without RPS. (For more detailed information, see "Obtaining the Sector Number of a Block on an RPS Device" on page 138.)

MF=
You can use the L-form of the XDAP macro instruction for a macro expansion consisting of a parameter list, or the E-form for a macro expansion consisting of executable instructions.

MF=E
The first operand (*ecb_symbol*) is required and can be coded as a symbol or supplied in registers 2 through 12. The *type*, *dcb_addr*, *area_addr,* and *length_value* operands can be supplied in either the L- or E-form. The *blkref_addr* operand can be supplied in the E-form or moved into the IOBSEEK field of the IOB by your program. The *sector_addr* is optional; it can be coded either in both the L- and E-form or in neither.

MF=L
The first two operands (*ecb_symbol* and *type*) are required and must be coded as symbols. If you choose to code *length_value* or *keylength_value*, they must be absolute expressions. Other operands, if coded, must be A-type addresses. (*blkref_addr* is ignored if coded.)

Note: The XDAP macro builds a channel program containing A-type addresses. These addresses refer to storage within the L-form of the macro. If you copy the L-form of the macro to a work area so that the program can be reentrant, the E-form of the XDAP macro does not update the A-type addresses. This results in an invalid channel program.

The *dcb_addr*, *area_addr*, *blkref_addr*, and *sector_value* operands can be coded as RX-type addresses or supplied in registers 2 through 12. The *length_value* and *keylength_value* operands can be specified as absolute expressions or decimal integers or supplied in registers 2 through 12.

Control Blocks Used with XDAP

The control blocks generated during execution of the XDAP macro instruction consist of the following:

Input/Output Block

The input/output block is 40 bytes in length and immediately follows the event control block. "Control Block Fields" on page 110 contains a diagram of the IOB (Figure 42 on page 121). You might want to examine the IOBSENS0, IOBSENS1, and IOBCSW fields if the ECB is posted with `X'41'`.

Event Control Block

Refer to “Event Control Block Fields” on page 124 for information on event control block fields.

Direct Access Channel Program

The direct access channel program is 24 bytes in length (except when set sector is used for RPS devices) and immediately follows the input/output block. Depending on the type of I/O operation specified in the XDAP macro instruction, one of four channel programs can be generated. The three channel command words for each of the four possible channel programs are shown in Figure 52.

Figure 52. Channel Program Command Words Used by XDAP

Type of I/O Operation	CCW	Command Code
Read by identification	1	Search ID Equal
	2	Transfer in Channel
Verify by identification	3	Read Key and Data
Read by key	1	Search Key Equal
	2	Transfer in Channel
Verify by key	3	Read Data
Write by identification	1	Search ID Equal
	2	Transfer in Channel
	3	Write Key and Data
Write by key	1	Search Key Equal
	2	Transfer in Channel
	3	Write Data

Note: For verifying operations, the third CCW is flagged to suppress the transfer of information to virtual storage.

When a sector address is specified with an RI, VI, or WI operation, the channel program is 32 bytes in length. Each of the channel programs in Figure 52 would be preceded by a set sector command.

RPS Device Sector Numbers

To obtain the performance improvement given by rotational position sensing, specify the *sector_addr* parameter in the XDAP macro. For programs that can be used with both RPS and non-RPS devices, test the UCBRPS bit to determine whether the device has rotational position sensing. If the UCBRPS bit is off, do not issue a channel program with a set sector command to the device.

The *sector_addr* parameter on the XDAP macro specifies the address of a 1-byte field in virtual storage. Store the sector number of the block to be located in this field. You can obtain the sector number of the block by using a resident system conversion routine. The address of the conversion routine's entry point is in the CVT0SCR1 field of the CVT.

For additional information about sector conversion see “Obtaining the Sector Number of a Block on an RPS Device” on page 138.

Chapter 6. Using Password Protected Data Sets

This chapter contains information about password protection for data sets. The use of password protection is not recommended but is provided for compatibility with other IBM operating systems. You should use RACF protection (using SAF) instead.

The password protection described in this chapter does not apply to data sets and catalogs managed by the Storage Management Subsystem (SMS) or to VSAM data sets.SMS ignores passwords. In addition, the PROTECT macro and SVC does not support a volume on a unit defined as dynamic.

If SAF is active, password protection is bypassed for all data sets. The system performs password validation only if SAF is inactive and the data set being accessed is not SMS-managed. The system provides SMS-managed data set and catalog protection through the system authorization facility (SAF) interface. For more SAF information, see *MVS/ESA SML: Managing Data* and the section “System Authorization Facility” in *OS/390 MVS Assembler Services Guide* and *OS/390 MVS Assembler Services Reference* .

For information about VSAM data set protection, see *DFSMS/MVS Using Data Sets* and *DFSMS/MVS Access Method Services for VSAM*. For information about RACF and its relationship to password protection, see *OS/390 Security Server (RACF) Introduction*.

The following are some reasons to use SAF instead of password protection:

- If you give a password to someone, you have no control over who they choose to give it to.
- Data sets tend to have various passwords, making you write them down. This is less secure than if you can memorize one SAF password.
- Batch job access or interactive non-TSO access requires that a system operator supply a password. Your communication to the operator is likely to be insecure. That operator might not be present when your job runs. The operator might have to give each data set's password to other operators.
- The program is halted while each password is supplied. This is contrary to the increased automation of modern systems.
- There is no way to know who has used a particular password.
- It is human nature not to change passwords, especially if there are many. As time passes, there is a greater danger of them being exposed.
- If more than a small number of data sets have passwords, then the time for the system to find the PASSWORD data set entry increases greatly. With RACF, the increase is much less. With a RACF generic profile there is no increase in search time when a new data set uses the same profile.
- With DASD shared between systems, the password definitions on each system are independent. They can get out of synchronization.
- The PASSWORD data set entry contains the data set name but not the volume serial number. If you create a data set before defining a password, you could find that someone has already defined a password for that data set name. Your data set will require the existing password just to scratch or rename it.

© Copyright IBM Corp. 1979, 1999

- Password protection is not supported on system-managed volumes or on dynamic devices.

To use the data set protection feature of the operating system, create and maintain a PASSWORD data set consisting of records that associate the names of the protected data sets with the passwords assigned to each data set. The ways to maintain the PASSWORD data set consist of:

- Writing your own routines
- Using the PROTECT macro instruction
- Using the utility control statements of the IEHPROGM utility program
- If you have TSO, using the TSO PROTECT command.

This chapter discusses the first two methods. The last two methods are discussed in the publications shown in the following list.

Before using the information in this chapter, you should be familiar with the contents of the following publications:

- *DFSMS/MVS Using Data Sets* describes the data set protection feature.
- The following publications describe the operator messages and replies associated with the data set protection feature:
  - *OS/390 MVS System Messages, Vol 1 (ABA-ASA)*
  - *OS/390 MVS System Messages, Vol 2 (ASB-EWX)*
  - *OS/390 MVS System Messages, Vol 3 (GDE-IEB)*
  - *OS/390 MVS System Messages, Vol 4 (IEC-IFD)*
  - *OS/390 MVS System Messages, Vol 5 (IGD-IZP)*
- *OS/390 MVS JCL Reference* describes the data definition (DD) statement parameter used to indicate that a data set is to be password protected. It is a subparameter of the LABEL parameter.
- *DFSMS/MVS Utilities* describes how to maintain the PASSWORD data set using the utility control statements of the IEHPROGM utility program.
- *OS/390 TSO/E Command Reference* describes how to use the TSO PROTECT command.

Providing Data Set Security

In addition to the usual label protection that prevents the opening of a data set without the correct data set name, the operating system provides data set security options that prevent unauthorized access to confidential data. Password protection prevents access to data sets until a correct password is entered by the system operator, or, for TSO, by a remote terminal operator.

The following types of access are allowed to password-protected data sets:

- PWREAD/PWWRITE—A password is required for read or write access.
- PWREAD/NOWRITE—A password is required for read access. Writing is not allowed.
- NOPWREAD/PWWRITE—Reading is allowed without a password. A password is required to write.

To prepare for use of the data set protection feature, place a sequential data set named PASSWORD on the system residence volume. This data set must contain at least one record for each data set placed under protection. Each record consists of a data set name, a password for that data set, a counter field, a protection-mode indicator, and a field for recording any information you wish to log. On the system residence volume, these records are formatted as a key area (data set name and password) and a data area (counter field, protection-mode indicator, and logging field). The data set is searched on the key area.

Notes:

1. The area allocated to the data set should not have been previously used for a PASSWORD data set, as this might cause unpredictable results when adding records to the data set.
2. If the system residence volume does not contain a PASSWORD data set, the system allows no access to password protected data sets. Do not rely on this for protection because anyone who creates a data set named PASSWORD on the system residence volume can define a password for any data set.
3. Data sets on magnetic tape are protected only when standard labels are used.

You can write routines to create and maintain the PASSWORD data set. For information on using the PROTECT macro instruction to maintain the PASSWORD data set, see “Maintaining the PASSWORD Data Set Using PROTECT” on page 152. Using the IEHPROGM utility program to maintain the PASSWORD data set is described in *DFSMS/MVS Utilities*. These routines can be placed in your own library or in the system's library (SYS1.LINKLIB). You can use a data management access method to read from and write to the PASSWORD data set.

Password-protected data sets can only be accessed by programs supplying the correct password. Upon receiving a request to open a protected data set, the operating system checks whether the data set has already been opened for this job step and if the access mode is compatible with the previously used protection mode. If neither condition is satisfied, a message requesting the password is sent to the operator console. If the program attempting to open the data is running under TSO in the foreground, the message is sent to the TSO terminal operator.

PASSWORD Data Set Characteristics

The PASSWORD data set and your operating system must reside on the same volume. That volume is the IPL volume. It is called the system residence volume. The space allocated to the PASSWORD data set must be contiguous. The amount of space allocated depends on the number of data sets you want to protect. Each entry in the PASSWORD data set requires 132 bytes of space. The organization of the PASSWORD data set is physical-sequential; the record format is unblocked, fixed-length records (RECFM=F). Each record that forms the data area is 80 bytes long (LRECL=80,BLKSIZE=80) and is preceded by a 52-byte key (KEYLEN=52). The key area contains the fully-qualified data set name (up to 44 bytes long) and a password, one to eight alphanumeric characters in length, left justified with blanks added to fill the areas.

Note: Data sets on magnetic tape complying with the specifications of the International Organization for Standardization (ISO) 1001-1979 or the American National Standards Institute (ANSI) X3.27-1978 do not include generation and version numbers as part of generation data set names. If included in

the PASSWORD data set, generation and version numbers for these data sets are ignored.

You should protect the PASSWORD data set by creating a password record for it when your program initially builds the data set or you should use RACF to control access. Thereafter, the PASSWORD data set cannot be opened (except by the operating system routines that scan the data set) without entering the password.

Note: If a problem occurs on a password-protected system data set, maintenance personnel require the password to access the data set and resolve the problem.

Creating Protected Data Sets

The data definition (DD) statement parameter LABEL= can be used to indicate that a data set is to be password protected. For data sets on DASD, an alternative method for a previously allocated data set is to use the PROTECT macro instruction, the IEHPROGM utility, or the TSO PROTECT command. A data set can be created and the protection indicator set in its label without entering a password record for it in the PASSWORD data set. In this case the system allows no access to the data set.

Operating procedures at your installation must ensure that password records for all data sets currently password-protected are entered in the PASSWORD data set. For installations where independent computing systems share common DASD resources, PASSWORD data sets on all systems must contain the appropriate password records for any protected data set on shared DASD.

Under certain circumstances, the order in which data sets are allocated and deallocated from multiple systems on shared DASD could result in loss of protection or corruption of data. For example, if a set is allocated and opened by a user on system A and then scratched by a different user on system B, the first user has a window to the unallocated (free) area. If any data set, protected or unprotected, is allocated in that space by a user on either system while the window is open, the new data set has no protection from the user with the window. The most common solution to this problem is to use GRS (see *OS/390 MVS Planning: Global Resource Serialization*).

While the allocation disposition is still NEW, a password-protected data set can be used without supplying a password. Once the data set is deallocated, a subsequent attempt to open it results in termination of the program unless the password record is available and the correct password is supplied. If the protection mode is NOPWREAD and the request is to open the data set for input or read backward, no password is required.

Tape Volumes Containing Multiple Password-protected Data Sets

To password protect a data set on a tape volume containing other data sets, password protect all the data sets on the volume. (Standard labels—SL, SUL, AL, or AUL—are required. For definitions of these label types and the protection-mode indicators that can be used, see *DFSMS/MVS Using Magnetic Tapes* .)

If you issue an OPEN macro instruction to create a data set following an existing password-protected data set, the password of the existing data set will be verified during open processing for the new data set. The password supplied must be associated with a PWWRITE protection-mode indicator.

Protection Feature Operating Characteristics

The topics that follow discuss the protection feature operating characteristics.

Terminating the Process

Processing is terminated if:

- The operator cannot supply the correct password for the protected data set within two attempts.
- A password record does not exist in the PASSWORD data set for the protected data set being opened.
- The protection-mode indicator in the password record and the method of I/O processing specified in the open routine do not agree; for example, OUTPUT specified against a read-only protection-mode indicator.
- There is a mismatch in data set names for a data set involved in a volume switching operation. This is discussed in the next paragraph.

Switching Volumes

Password protection is retained when volumes of a multivolume data set are switched. If the following conditions are met, the system accepts a newly-mounted tape volume to be used for input or a newly-mounted direct access volume:

- The data set names are the same in the password record for the data set and the job file control block (JFCB). (This ensures that the problem program has not changed the data set name in the JFCB since the data set was opened.)
- The protection-mode indicator in the password record is compatible with the processing mode, and a valid password has been supplied.

The system accepts a newly-mounted tape volume for output under any of the following conditions:

- The security indicator in the HDR1 label indicates password protection; the data set name in the password record is the same as the data set name in the JFCB; and the protection-mode indicator is compatible with the processing mode. (If the data set name in the JFCB has been changed, a new password is requested from the operator.)
- The security indicator in the HDR1 label does not indicate password protection. (A new label is written with the security indicator indicating password protection.)
- Only a volume label exists. (A HDR1 label is written with the security indicator indicating password protection.)

Concatenating Data Sets

A password is requested for every protected data set that is involved in a concatenation of data sets.

SCRATCH and RENAME Functions

To delete or rename a protected data set, the job step making the request must be able to supply the password. The system checks to see if the job step is currently authorized to write to the data set. If the job step is not, a password must be provided that is associated with a WRITE protection-mode indicator.

Maintaining the Counter

The operating system increments the counter in the password record on each usage, but no overflow indication will be given (overflow after 65535 openings). Provide a counter maintenance routine to check and, if necessary, reset this counter.

Maintaining the PASSWORD Data Set Using PROTECT

To use the PROTECT macro instruction, your PASSWORD data set must be on the system residence volume. The PROTECT macro can be used to:

- Add an entry to the PASSWORD data set
- Replace an entry in the PASSWORD data set
- Delete an entry from the PASSWORD data set
- Obtain a list of information about an entry in the PASSWORD data set; this list contains the security counter, access type, and the 77 bytes of security information in the data area of the entry.

The PROTECT macro also updates the DSCB of a protected direct access data set to reflect its protection status; this feature eliminates the need for job control language when you protect a data set.

PROTECT does not support data sets on dynamic devices.

Record Format

When using the PROTECT macro, the PASSWORD data set must contain at least one record for each protected data set. The password (the last eight bytes of the key area), that you assign when you protect the data set for the first time, is called the control password.

You can create as many secondary records for the same protected data set as needed. Passwords assigned to these additional records are called secondary passwords. This feature allows several users to access the same protected data set while you control the way it is used. For example, one user could be given read/write authorization while another user is assigned a password that allows only read access.

Protection-Mode Indicator

You can set the protection-mode indicator (third data byte) in the password record to one of the following values:

- `X'00'` to indicate that the password is a secondary password and the protected data set is to be read only (PWREAD).
- `X'80'` to indicate that the password is the control password and the protected data set is to be read only (PWREAD).
- `X'01'` to indicate that the password is a secondary password and the protected data set is to be read and written (PWREAD/PWWRITE).
- `X'81'` to indicate that the password is the control password and the protected data set is to be read and written (PWREAD/PWWRITE).

The checking sequence for the protection status of a data set produces the following defaults:

If control password is:	Secondary password must be:
1. PWREAD/PWWRITE or PWREAD/NOWRITE	PWREAD/PWWRITE or PWREAD/NOWRITE
2. NOPWREAD/PWWRITE	NOPWREAD/PWWRITE

If the control password is set to either of the settings in item 1 and you try to set the secondary password to NOPWREAD/PWWRITE, the secondary password reverts to PWREAD/PWWRITE.

If the control password is changed from either of the settings in item 1 to the setting in item 2, the secondary password is reset to NOPWREAD/PWWRITE.

If the control password is changed from the setting in item 2 to a setting in item 1, the secondary password is set to PWREAD/PWWRITE.

Because the DSCB of the protected data set is updated only when the control password is changed, protection attributes for secondary passwords might conflict with the protection attributes of the control password.

PROTECT Macro Specification

The format of the PROTECT macro is:

```
►►──┬──────────┬──PROTECT──parameter_list_address──────────────────►◄
    └─label────┘
```

parameter list address—A-type address, (2-12), or (1)
indicates the location of the parameter list. The parameter list must be created before the PROTECT macro is issued. The address of the parameter list can be passed in register 1, in any of registers 2 through 12, or as an A-type address. The first byte of the parameter list must be used to identify the function (add, replace, delete, or list). See Figure 53 on page 154 through Figure 56 on page 157 for the parameter lists and codes used to identify the functions.

Note: The parameter lists and the areas addressed by the list must reside below 16MB virtual. PROTECT will fail the request if the actual UCB of the system residence volume is above 16MB or if PROTECT tries to update the data set's DSCB and one of its UCBs is above the 16MB line.

PROTECT Macro Parameter Lists

ADD Function

0	X'01'	13	Control password pointer
1	00 00 00	16	Number of volumes
4	Data set name length	17	Volume list pointer
5	Data set name pointer	20	Protection code
8	00	21	New password pointer
9	00 00 00	24	String length
10	00	25	String pointer

Figure 53. Parameter List for ADD Function

0 X'01'
Entry code indicating ADD function.

4 Data set name length.

5 Data set name pointer.

13 Control password pointer.
The control password is assigned when the data set is placed under protection for the first time. The pointer can be 3 bytes of binary zeros if the new password is the control password.

16 Number of volumes.
If the data set is not cataloged, to have it flagged as protected, specify the number of volumes in this field. A zero requests that the catalog information be used.

17 Volume list pointer.
If the data set is not cataloged, to have it flagged as protected, provide the address of a list of volume serial numbers in this field. Zeros request that the catalog information be used.

20 Protection code.
A 1-byte number indicating the type of protection: `X'00'` indicates default protection (for the ADD function; the default protection is the type of protection specified in the control password record of the data set); `X'01'` indicates that the data set is to be read and written; `X'02'` indicates that the data set is read only; and `X'03'` indicates that the data set can be read without a password, but a password is needed for write access. The PROTECT macro uses the protection code value, specified in the parameter list, to set the protection-mode indicator in the password record.

21 New password pointer.
If the data set is being placed under protection for the first time, the new password becomes the control password. If you are adding a secondary entry, the new password is different from the control password.

24 String length.
The length of the character string (maximum 77 bytes) to be placed in the optional information field of the password record. If you do not want to add information, set this field to zero.

25 String pointer.
The address of the character string placed in the optional information field. If you do not want to add information, set this field to zero.

REPLACE Function

0	X'02'	13	Control password pointer
1	00 00 00	16	Number of volumes
4	Data set name length	17	Volume list pointer
5	Data set name pointer	20	Protection code
8	00	21	New password pointer
9	Current password pointer	24	String length
12	00	25	String pointer

Figure 54. Parameter List for REPLACE Function

0 X'02'.
Entry code indicating REPLACE function.

4 Data set name length.

5 Data set name pointer.

9 Pointer to current password.
The address of the password that is going to be replaced.

13 Control password pointer.
The address of the password assigned to the data set when it was first placed under protection. The pointer can be set to 3 bytes of binary zeros if the current password is the control password.

16 Number of volumes.
If the data set is not cataloged, to have it flagged as protected, specify the number of volumes in this field. A zero requests that the catalog information be used.

17 Volume list pointer.
If the data set is not cataloged, to have it flagged as protected, provide the address of a list of volume serial numbers in this field. If this field is zero, the catalog information is used.

20 Protection code.
A 1-byte number indicating the type of protection: X'00' indicates default protection (for the REPLACE function the default is the type of protection specified

in the control password record of the data set); `X'01'` indicates that the data set is to be read and written; `X'02'` indicates that the data set is to be read only; and `X'03'` indicates that the data set can be read without a password, but a password is needed for write access.

21 New password pointer.
The address of the password to be used to replace the current password.

24 String length.
The length of the character string (maximum 77 bytes) to be placed in the optional information field of the password record. If you do not want to add information, set this field to zero.

25 String pointer.
The address of the character string to be placed in the optional information field. If you do not want to add information, set this field to zero.

DELETE Function

0	X'03'	9	Current password pointer
1	00 00 00	12	00
4	Data set name length	13	Control password pointer
5	Data set name pointer	16	Number of volumes
8	00	17	Volume list pointer

Figure 55. Parameter List for DELETE Function

0 X'03'.
Entry code indicating DELETE function.

4 Data set name length.

5 Data set name pointer.

9 Current password pointer.
The address of the password to be deleted. You can delete either a control or secondary entry.

13 Control password pointer.
The address of the password assigned to the data set when it was placed under protection for the first time. The pointer can be 2 bytes of binary zeros if the current password is also the control password.

16 Number of volumes.
If the data set is not cataloged, to have it flagged as protected, specify the number of volumes in this field. A zero requests that the catalog information be used.

17 Volume list pointer.
If the data set is not cataloged, to have it flagged as protected, provide the address of a list of volume serial numbers in this field. Zeros request that the catalog information will be used.

LIST Function

0	X'04'	5	Data set name pointer
1	80–byte buffer pointer	8	00
4	Data set name length	9	Current password pointer

Figure 56. Parameter List for LIST Function

0 X'04'.
Entry code indicating LIST function.

1 80-byte buffer pointer.
The address of a buffer to receive the information returned to your program by the macro instruction.

4 Data set name length.

5 Data set name pointer.

9 Current password pointer.
The address of the password of the record to be listed.

Return Codes from the PROTECT Macro

When the PROTECT macro finishes processing, register 15 contains one of the following return codes:

Figure 57 (Page 1 of 2). PROTECT Return Codes

Return Code	Description
0 (X'00')	Updating of the PASSWORD data set completed successfully.
4 (X'04')	The password of the data set name was already in the PASSWORD data set.
8 (X'08')	The password of the data set name was not in the PASSWORD data set.
12 (X'0C')	A control password is required or the one supplied is incorrect.
16 (X'10')	The supplied parameter list was incomplete or incorrect.
20 (X'14')	There was an I/O error in the PASSWORD data set or the SYSRES volume (which contains the PASSWORD data set), has an actual UCB that resides above the 16MB line.
24 (X'18') [1]	The PASSWORD data set was full.
28 (X'1C')	The validity check of the buffer address failed.
32 (X'20') [2]	The LOCATE macro failed. LOCATE's return code is in register 1, and the number of indexes searched is in register 0.

Figure 57 (Page 2 of 2). PROTECT Return Codes

Return Code	Description
36 (X'24') [2]	The OBTAIN macro for the user data set failed. OBTAIN's return code is in register 1. The user dataset resides on a volume that has an actual UCB that resides above the 16MB line. Register 1 contains a 4, which simulates an OBTAIN return code.
40 (X'28') [2]	The DSCB could not be updated.
44 (X'2C')	The PASSWORD data set does not exist.
48 (X'30') [2]	Tape data set cannot be protected.
52 (X'32') [2]	Data set in use.
56 (X'38') [2]	The data set uses the virtual storage access method (VSAM).

Notes:

1. A message is written to the console indicating that the PASSWORD data set is full.
2. The PASSWORD data set has been updated, but the DSCB has not been flagged to indicate the protected status of the data set.

Chapter 7. Using System Macro Instructions

This chapter contains information about system macro instructions. The macros are grouped functionally where appropriate and perform the stated functions.

- Ensure data security (DEBCHK macro)
- Obtain device characteristics from control blocks and system tables (DEVTYPE macro)
- Modify the JFCB (RDJFCB and OPEN TYPE=J macros)
- Manipulate I/O activity queues (PURGE and RESTORE macros)
- Perform track capacity calculations (TRKCALC macro).

Some functions of these macros tend to depend on the internal logic of the system.

Before reading this chapter, you should be familiar with the publications *High Level Assembler/MVS & VM & VSE Language Reference* or *Assembler H V2 Language Reference.* They contain the information necessary to code programs in the assembler language.

Ensuring Data Security by Validating the Data Extent Block

Protecting one user's data from inadvertent or malicious access by an unauthorized user depends on protection of the data extent block (DEB). The DEB is a critical control block because it contains information about the device a data set is mounted on, and describes the location of data sets on direct access device storage volumes.

To ensure that only a valid system-provided DEB (normally built by open) is passed to data management functions, the DEBCHK verify function is used. OPEN places the address of DEBs it creates to a DEB table, which is used by the verify function. If you code a routine that builds a DEB, add the address of the DEB you built to the DEB table. If you code a routine that depends on the validity of a DEB that is passed to your routine, verify that the DEB passed to your routine has a valid entry in the DEB table and points to your DCB or access method control block (ACB). Use the TYPE=ADD and the TYPE=VERIFY operands of the macro, respectively.

To prevent an asynchronous routine from changing or deleting, or assigning a new DEB to a DCB, hold the local lock. In this case, use the branch entry to the DEBCHK verify routine and use the DEB address returned in register 1, not the DEB address in the DCB. The DCB will remain valid as long as your program holds the local lock or prevents untrustworthy programs from running.

Your program must be executing in 24-bit addressing mode when you call the DEBCHK macro.

The DEBCHK macro instruction provides four functions:

- Adds the address of a DEB to the DEB table.
- Verifies that the DEB table associated with a specific job step contains the address of a valid DEB and that the DEB points to the DCB (or ACB). A problem program can use this function to verify that a DEB is valid.

© Copyright IBM Corp. 1979, 1999

- Deletes the address of a DEB from the DEB table. Any program that deletes a user DEB must, before it deletes the DEB, issue a DEBCHK macro with a TYPE=DELETE operand to delete the address of the DEB from the DEB table. If the DEB validity-checking routine encounters an error while deleting the address from the DEB table, the job step is abnormally terminated.
- Deletes the address of a DEB from the DEB table in the same way as the preceding function, except that, instead of terminating the job step, this function merely returns an error code in register 15. This function is provided to prevent recurring abnormal termination.

The DEB fields used for EXCP and EXCPVR are illustrated in Appendix C, “Control Blocks” on page 419 (all the DEB fields are illustrated in *DFSMS/MVS DFSMSdfp Diagnosis Reference*).

DEBCHK Macro Specification

The format of the DEBCHK macro is:

```
>>──┬──────────┬──DEBCHK──cbaddr──┬──────────────────────────────────┬──────────────────────────>
    └─label────┘                  └─,TYPE=─┬─VERIFY─┬─┬──────────────┘
                                           ├─ADD────┤ │
                                           ├─DELETE─┤ │
                                           └─PURGE──┘ │
>──┬──────────────────────────┬──┬──────────────────────┬──────────────────────────────────────────>
  └─,AM=─┬─amtype──────────┬─┘  └─,BRANCH=─┬─NO──┬─┘
         ├─(amaddr)────────┤               └─YES─┘
         └─((amreg))───────┘
>──┬──────────────────────┬──┬──────────────────────┬──┬──────────────────────┬──────────────────>
  └─,TCBADDR=address──────┘  ├─,KEYADDR=address─────┤  └─,SAVREG=reg──────────┘
                             └─,KEYREG=reg──────────┘
>──┬──────────┬──────────────────────────────────────────────────────────────────────────────────>◄
  └─,MF=L────┘
```

cbaddr
Control block address.

for **BRANCH=NO**
RX-type address, (2-12), or (1)

A control block address passed to the DEBCHK routine. This operand is ignored if MF=L is coded. For verify, add, and delete requests, *cbaddr* is the address of a DCB or ACB that points to the DEB whose address is either verified to be in the DEB table, added to the DEB table, or deleted from the DEB table. For the purge function, *cbaddr* is the address of the DEB whose pointer is to be purged from the table: No reference is made to the DCB or ACB.

Note: A spooled DCB's DEB does not point back to the DCB, but to the spooled ACB; in this case, the DEBCHK should be issued against the ACB.

for **BRANCH=YES**
The A-type address of a 4-byte field, or a register (3-9) or (12), that points to the DCB or ACB containing the DEB to be verified.

TYPE=VERIFY or **ADD** or **DELETE** or **PURGE**
indicates the function to be performed. If MF=L is coded, TYPE is ignored. The functions are:

VERIFY
This function is assumed if the TYPE operand is not coded. The control program checks the DEB table to determine whether the DEB pointer is in the table at the location indicated by the DEBTBLOF field of the DEB. The DEB is also checked to verify that DEBDCBAD points to the DCB (or ACB) passed to DEBCHK. The DEBAMTYP field in the DEB is compared to the AM operand value, if given. The two must be equal. TYPE=VERIFY can be issued in either supervisor or problem state.

ADD
The DEB and the DCB (or ACB) must point to each other before the DEB address can be added to the DEB table. Before the DEB pointer can be added to the table, the DEB itself must be queued on the current TCB DEB chain (the TCBDEB field contains the address of the first DEB in the chain). DEBCHK adds the DEB address to the DEB table at some offset into the table. DEBCHK places a value in the DEBTBLOF field of the DEB and inserts the access method type into the DEBAMTYP field of the DEB. DEBCHK places a zero in the DEBAMTYP field if the AM operand is not coded. TYPE=ADD can be issued only in supervisor state.

DELETE
The DEB and the DCB (or ACB) must point to each other before the DEB address can be deleted from the DEB table. TYPE=DELETE can be issued only in supervisor state.

PURGE
DEBCHK removes the DEB pointer from the DEB table without checking the DCB (or ACB). TYPE=PURGE can be issued only in supervisor state.

AM
specifies an access method value. Each value corresponds to a particular access method type (note that BPAM and SAM have the same values):

Value	Type
(X'00')	NONE
(X'01')	VSAM
(X'02')	EXCP
(X'04')	TCAM
(X'08')	GAM
(X'10')	TAM
(X'20')	BPAM
(X'20')	SAM
(X'40')	BDAM
(X'80')	ISAM
(X'81')	SUBSYS
(X'84')	TCAMAP

The operand can be coded in one of the following three ways, only the first of which is valid for the list form (MF=L) of the instruction.

amtype
refers to the access method: ISAM, BDAM, SAM, BPAM, TAM (which refers to BTAM only), GAM, TCAM, EXCP, or VSAM. TCAMAP identifies a

TCAM application-program DEB. SUBSYS identifies a subsystem of the operating system, such as a job entry subsystem. NONE indicates that no access method or subsystem is specified.

(*amaddr***)**
is the RS-type address of the access method value. This format cannot be coded when MF=L is used.

((*amreg***))**
is one of the general registers 1 through 14 that contains the access method value in its low-order byte (bit positions 24 through 31). The high-order bytes are not inspected. This form cannot be used when MF=L is coded.

The use of *amaddr* and *amreg* should be restricted to those cases where the access method value has been generated previously by the MF=L form of DEBCHK. If MF=L is not coded, the significance of the AM operand depends upon the TYPE.

If TYPE is ADD and AM is specified, the access method value is inserted in the DEBAMTYP field of the DEB, and all subsequent DEBCHK macros referring to this DEB must either specify the same AM or omit the operand. When the AM operand is omitted for TYPE=ADD, a null value (0) is placed in the DEB and all subsequent DEBCHK macros must omit the AM operand.

If AM is specified when the TYPE is PURGE, DELETE, or VERIFY, the access method value is compared to the value in the DEBAMTYP field of the DEB. If AM is omitted, no comparison is made.

BRANCH=NO or **YES**
specifies whether you want to use the branch entry to the DEBCHK verify routines.

NO
specifies branch entry is not to be used. The operands SAVREG, TCBADDR, and KEYADDR/KEYREG are ignored.

YES
specifies the branch entry is to be used. TYPE=VERIFY must be implicitly or explicitly specified. The operands TCBADDR and KEYADDR/KEYREG are required. AM and MF are ignored. Notes for BRANCH=YES:

- Registers 1, 2, 10, 11, 14, and 15 must not be used for SAVREG=.
- Registers 1, 2, 10, 11, 14, 15, and the register specified for SAVREG= must not be used for *cbaddr*, TCBADDR=, or KEYADDR=/KEYREG=.
- The contents of registers 10, 11, and 14 are unpredictable on completion. Also, if you do not specify SAVREG=, the contents of register 2 are unpredictable.
- At completion, register 1 contains the address of the DEB, and register 15 contains either 0, 4, or 16 (see “Return Codes from DEBCHK” on page 163 for codes and their meanings).
- Can be specified when operating in 24-bit or 31-bit addressing mode.

TCBADDR=*address*—RX-type address, (3-9), or (12)
specifies the location or register containing the address of the TCB to be used by the DEBCHK verify routine. Use this operand only with BRANCH=YES.

KEYADDR=*address*—RX-type address
specifies the location, or a register pointing to the location of a field containing the key to be used when accessing the DCB (or ACB). Use this operand only with BRANCH=YES.

KEYREG=*reg*
specifies the register containing the key value in bit positions 24-27 to be used when accessing the DCB(or ACB). Use this operand only with BRANCH=YES.

SAVREG=*reg*
specifies the register in which register 2 is to be saved. Use this operand only with BRANCH=YES.

MF=L
indicates the list form of the DEBCHK macro instruction. When MF=L is coded, a parameter list is built, consisting of the access method value that corresponds to the AM keyword. This value can be referred to by name in another DEBCHK macro by coding AM=(*amaddr*), or it can be inserted into the low-order byte of a register before issuing another DEBCHK macro by coding AM=((*amreg*)).

Return Codes from DEBCHK

Register 15 contains one of the following codes:

Return Code	Meaning
0 (X'00')	The requested function completed successfully and register 1 contains the address of the DEB.
4 (X'04')	Either (a) the DEB table associated with the job step does not exist; or (b) the DEB field that is an index into the DEB table contains an invalid value; or (c) the control block address is not the same as the content of the DEB table entry.
8 (X'08')	An invalid TYPE was specified. (The DEBCHK routine was entered by a branch, not by the macro.)
12 (X'0C')	Your program was not authorized and TYPE was not VERIFY.
16 (X'10')	DEBDCBAD did not contain the address of the DCB (or ACB) that was passed to the DEBCHK routine.
20 (X'14')	The AM value does not equal the value in the DEBAMTYP field.
24 (X'18')	The DEB is not on the DEB chain and TYPE=ADD was specified.
28 (X'1C')	TYPE=ADD was specified for a DEB that was already entered in the DEB table.
32 (X'20')	The DEB table exceeded the maximum size and TYPE=ADD.

Obtaining I/O Device Characteristics

Use the DEVTYPE macro instruction to request information relating to the characteristics of an I/O device and to cause this information to be placed into a specified area. (The results of a DEVTYPE macro instruction executed before a checkpoint is taken should not be considered valid after a checkpoint/restart occurs.) The IHADVA macro maps the data returned by the DEVTYPE macro.

The topics that follow discuss the DEVTYPE macro, device characteristics, and the output for specific devices.

Your program can issue the DEVTYPE macro while executing in 24- or 31-bit mode. If your program is executing in 31-bit mode, the parameter list, information list, and the UCB address list can reside above the 16MB line.

Figure 62 on page 175 shows the output for each device type that results from issuing the DEVTYPE macro without the INFOLIST parameter.

Note: For all currently supported devices, DEVTYPE does not return enough information to perform space calculations. TRKCALC should be used to perform space calculations. For information on using the TRKCALC macro, see "Performing Track Calculations" on page 198.

DEVTYPE Macro Specification

There are four forms of DEVTYPE macro invocation. They are the standard form, execute form, list form, and INFO form. These forms are described in the following sections.

Parameter list compatibility note:

If you do not code the BELOW or ANY value for the UCBLIST parameter, then you can assemble and run DEVTYPE as described in this book on any release of DFSMS/MVS.

If you code the BELOW or ANY value for the UCBLIST parameter, then the program cannot be assembled on MVS/DFP Version 3 but the system will ignore the BELOW or ANY value if you assemble it on DFSMS/MVS.

If you code the INFOLIST or INFO parameter, then the macro cannot be assembled or run on MVS/DFP Version 3.

If you omit INFOLIST, INFO, BELOW and ANY and the PLISTVER parameter, then you can assemble the program on DFSMS/MVS but it will not run on MVS/DFP Version 3. If you code PLISTVER=0, then you can assemble the program on DFSMS/MVS but not on MVS/DFP and you can run the program on either level of the system. During assembly your program can choose which parameters to code by using the technique described in "Determining DFARELS During Assembler Macro Phase" on page 207.

There are two types of call to the standard form of the DEVTYPE macro.

- The minimum type call refers to the DD statement for the device. It has no list or execute forms.
- The UCBLIST/INFOLIST type call requires you to specify either the UCBLIST or INFOLIST parameter or both.

The format of the different types of call to the standard form of DEVTYPE macro follow and the parameter descriptions follow the formats.

Minimum Type Call

The minimum type of call refers to the DD statement for the device. For this type call, as the parameters are passed in general registers no list or execute form exists. It returns the device information to the area you specify.

The format of the minimum type call of the DEVTYPE macro is:

```
►►──┬──────────┬──DEVTYPE──ddloc,area_addrx──┬──────────────────────┬──►◄
    └─label────┘                             └─,DEVTAB──┬─────────┬─┘
                                                        └─,RPS────┘
```

UCBLIST or INFOLIST Type of Call

The UCBLIST or INFOLIST type call requires you to specify either the UCBLIST or INFOLIST parameter or both.

The format of the UCBLIST or INFOLIST type call is:

```
►►──┬──────────┬──DEVTYPE──┬──ddloc──────────────────────────────────────────┬──►
    └─label────┘           └──UCBLIST=(ucbl_addr,ucbl_num──┬─────────┬──)──┘
                                                            ├─,BELOW──┤
                                                            └─,ANY────┘
►──,(area_addr,area_size)──┬──────────────────────────────┬──────────────────►
                           └──INFOLIST=codel_addr──────────┘
►──┬──────────────────────────────────────────────────────────────────────►◄
   └──,PLISTVER=──┬─────┬──┘
                  ├─MAX─┤
                  ├─0───┤
                  └─1───┘
```

ddloc—A-type address or (1-12)
: the name of an 8-byte field that contains the symbolic name of the DD statement to which the device is assigned. The name must be left justified in the 8-byte field, and must be followed by blanks if the name is fewer than eight characters. Each DEVTYPE macro executable invocation must have either *ddloc* to identify an allocated data set or UCBLIST= to identify one or more devices.

You can specify *ddloc* as (1) only if you omit all keywords.

area_addrx—Rx-type address or (0, 2-12)
area_addr—A-type address or (2-12)
: the name of an area into which the device information is to be placed. If your program does not specify the UCBLIST or INFOLIST function, the area is two, five, or six words long, depending on whether you specify the DEVTAB and RPS operands. If your program specifies the UCBLIST parameter without INFOLIST the area must be 6 words long for each UCB. The area must be on a word boundary.

If your program specifies the INFOLIST parameter, then the length of the area depends on what you coded for INFO on the referenced DEVTYPE macro. The INFO description states how many bytes are returned for each value you specify in the INFO list. To calculate the area size needed, multiply the sum of those values by *ucbl_num* in UCBLIST. If you omit UCBLIST (and specify *ddloc*), do not multiply.

Note that if you specify UCBLIST (and not *ddloc*) then (*area_addr,area_size*) must still be the second positional parameter. Specify it after a comma.

area_size—absolute expression or (2-12)
the size (in bytes) of the area into which the device information is to be placed. See Figure 58 on page 166.

Figure 58. Minimum size of area

ddloc specified	UCBLIST specified	INFOLIST specified	Minimum size of area
Yes	No	Omitted or 0	8, 20, or 24 bytes depending on whether DEVTAB and RPS are coded.
Yes	No	Yes	Sum of the number of bytes returned for each code specified with INFO. See also the INFO keyword.
No	Yes	Omitted or 0	24 bytes per UCB
No	Yes	Yes	The product of the number of UCBs specified with UCBLIST and the sum of the number of bytes returned for each code specified with INFO. See also the INFO keyword.

DEVTAB[,RPS]
DEVTAB is only meaningful for direct access devices. If DEVTAB is specified, the following number of words of information is placed in your area:

- For direct access devices: 5 words
- For non-direct access devices: 2 words.

If you do not specify DEVTAB, INFOLIST, or UCBLIST, one word of information is placed in your area if the reference is to a graphics or teleprocessing device; for any other type of device, two words of information are placed in your area.

RPS
If RPS is specified, DEVTAB must also be specified. The RPS parameter causes one additional word of rotational position sensing information to be included with the DEVTAB information.

UCBLIST=*(ucbl_addr,ucbl_num* **[,BELOW** or **ANY])**
UCBLIST provides a list service in which the caller passes a list of 4-byte UCB addresses and specifies the number of UCB address entries that are in the list. You must specify the UCBLIST parameter or *ddloc*. The BELOW or ANY are optional keywords that indicate whether the address passed by the UCB parameter contains a 3-byte or 4-byte UCB address. This keyword only applies to callers running in AMODE 31. If the caller is running in AMODE 24, the keyword ANY is ignored, i.e. the high-order byte is treated as `X'00'`.

If you do not specify INFOLIST, then the information returned is always returned in 6-word entries (one entry per UCB address) regardless of the

device type. The words that would contain information not applicable to the device for that entry are not altered.

Note: The DEVTYPE macro will accept a captured UCB or an actual UCB address above or below the 16MB line, via the UCBLIST = parameter. It will also accept 24- or 31-bit addresses of UCB copies. The UCB copy must be on a word boundary but can be above or below the 16MB line. Unauthorized programs can get a copy of the UCB by using the UCBSCAN macro and specifying the COPY and UCBAREA keywords. Refer to *OS/390 HCD Planning* for details.

ucbl_addr—A-type address or (2-12)
the name of an area containing a list of 4-byte UCB addresses.

ucbl_num—absolute expression or (2-12)
the number of 4-byte UCB address entries in the list.

BELOW
The UCB parameter contains addresses of UCBs that reside in storage below 16MB, or a captured UCB. This is the default.

Note: If BELOW is specified, the high-order byte of the UCB address is treated as `X'00'`.

ANY
The UCB parameter contains a 4-byte UCB address.

Note: If ANY is specified when invoking in 31-bit mode, DEVTYPE treats each word in the UCB address list as a 31-bit address.

INFOLIST=*codel_addr*
the name of an area that specifies the types of information DEVTYPE is to return. Coding INFOLIST=0 has the same effect as omitting it. If you specify INFOLIST, then you must also specify *ddloc* or UCBLIST and you must specify *(area_addr,area_size)*. The format of the returned words is described under DEVTYPE, DASD, and SUFFIX (see page 171).

codel_addr—A-type address or (2-12)
Specifies an area which contains an instance of the DEVTYPE macro where you coded only the INFO keyword.

PLISTVER=0 or **1** or **MAX**
specifies the version of the parameter list for the macro to generate.

0 The program cannot be assembled on a level of the system earlier than DFSMS/MVS Release 3 but it can run on any system level that supports the coded parameters. You cannot code this with the INFOLIST or INFO parameter. You can code it with the third value for UCBLIST but MVS/DFP will ignore that third value. The parameter list will be 20 bytes long.

1 The program can be assembled and run only on DFSMS/MVS Version 1.1 or later. The parameter list will not be acceptable prior to DFSMS/MVS. This specification generates a 24-byte parameter list.

MAX
In the current release this has the same effect as coding 1. In a future release this value might allow a parameter list that is incompatible with an earlier release. This is the default and generates a 24-byte parameter list.

DEVTYPE—Execute Form

The execute form of the DEVTYPE macro is:

```
►►──┬──────────┬──DEVTYPE──────────────────────────────────────────►
    └─label────┘

►──┬──────────────────────────────────────────────────────────────┬──►
   │  ┌──────────────────────────────────────────────────────────┐
   │  │                                          ┌─,BELOW─┐      │
   │  └─UCBLIST=(ucbl_addr,ucbl_num──────────────┤        ├──)───┘
   │                                             └─,ANY───┘
   │
►──┬──────────────────────────────────────────────────────────────┬──►
   │  ┌──────────────────────────────────────────────────────────┐
   │  │                                                          │
   │  └─,(area_addrx,area_size)──┘   └─INFOLIST=codel_addrx──┘
   │
►──┬──────────────────────────────────────────────────────────────┬──►
   │                                                              │
   │                                                              │
   │                                                              │
```

ANY
has the same meaning as the standard form of the macro. If you write two or three values for UCBLIST on the execute form, it replaces and overrides all three values in the list form. If you omit UCBLIST on the execute form, DEVTYPE will use the values in the list form.

INFOLIST=*codel_addrx*
has the same meaning as the standard form of the macro. You can code INFOLIST=0 to remove a previous INFOLIST value that is in the list form.

codel_addrx—RX-type address or (2-12)
has the same meaning as the standard form of the macro.

PLISTVER=0 or **1** or **MAX**
has the same meaning as on the standard form of the macro. If you code a non-default value for PLISTVER on the list form, then code the same value on the execute form.

MF=(E,*name***)**
specifies the execute form of DEVTYPE. The execute form of the DEVTYPE macro is valid only with the UCBLIST or INFOLIST function.

E specifies the execute form of the macro. Use the execute form to modify a parameter list and call the DEVTYPE function.

name—RX-type address or (1-12)
label of the parameter list constructed by the corresponding MF=L form.

DEVTYPE—List Form

The list form of the DEVTYPE macro follows:

```
►►──┬──────────┬──DEVTYPE──────────────────────────────────────────►
    └─label────┘
►──┬──────────────────────────────────────────────────────────────►
   │  ┌─ddloc──────────────────────────────────────────────────┐
   │  └─UCBLIST=──┬─YES──────────────────────────────────────────┤
   │              │                  ┌─,BELOW─┐                 │
   │              └─(ucbl_addr,ucbl_num──┼──────┼──)──────────┘
   │                                  └─,ANY───┘
►──┬──────────────────────────────────────────────────────────────►
   │  ┌─,(area_addr,area_size)─┐  ┌─INFOLIST=codel_addr─┐
   │  │                        │  │                     │
►──┼──┼────────────────────────┼──┼─────────────────────┼──MF=L──►◄
   │  │                        │  │                     │
   │  ┌─,PLISTVER=──┬─MAX─┐────┘  │                     │
   │  │             ├─0───┤       │                     │
   │  │             └─1───┘       │                     │
```

label
label of the parameter list to be used. When you specify *label* with MF=L, *label* must be the same as *name* on MF=(E,*name*).

ddloc—A-type address
has the same meaning as the standard form of the macro.

UCBLIST=YES or (*ucbl_addr,ucbl_num*,**BELOW** or **ANY**)
has the same meaning as the standard form of the macro. This is the default.

YES
DEVTYPE allows UCBLIST=YES as a place holder when MF=L is coded. It has no effect on the macro expansion.

ucbl_addr—A-type address
has the same meaning as the standard form of the macro.

ucbl_num—absolute expression.
has the same meaning as the standard form of the macro.

BELOW
has the same meaning as the standard form of the macro. This is the default.

ANY
has the same meaning as the standard form of the macro.

Coding UCBLIST=YES has no effect on the macro expansion. Do not code UCBLIST=YES with a DD name parameter on the same macro.

area_addr—A-type address
has the same meaning as the standard form of the macro. This parameter must be coded on the list and/or the execute form.

area_size—absolute expression
has the same meaning as the standard form of the macro. This parameter must be coded on the list and/or the execute form.

INFOLIST=*codel_addr*
has the same meaning as the standard form of the macro.

codel_addr—A-type address or (2-12)
has the same meaning as the standard form of the macro.

PLISTVER=0 or **1** or **MAX**
has the same meaning as on the standard form of the macro. If you code a non-default value for PLISTVER on the list form, then code the same value on the execute form.

MF=L
specifies the list form of DEVTYPE. The list form of the DEVTYPE macro is valid only with the UCBLIST or INFOLIST function. By specifying MF=L you construct a parameter list, and you can subsequently supply the values by specifying the execute form of the macro.

DEVTYPE—Info Form

The INFO form of the DEVTYPE macro is used to generate a code list for the INFOLIST parameter. The INFO form is not executable. It has no list or execute form. When you specify the INFOLIST parameter of DEVTYPE, it refers to an expansion of DEVTYPE with only the INFO parameter.

The INFO form of the DEVTYPE macro is:

```
►►──┬──────────┬──DEVTYPE──INFO=(──┬──DEVTYPE──┬──)──────────────────────►◄
    └─label────┘                   │           ,│
                                   ├──DASD─────┤
                                   └──SUFFIX───┘
```

INFO=(DEVTYPE or **,DASD** or **,SUFFIX)**
specifies the types of information that you wish to retrieve. Specify any combination of the indicated values in any order. You can specify the same value more than once in the list. The parentheses around the INFO value can be omitted if there is only one value.

Specifying the INFO keyword causes the macro expansion to be a list and it will not be executable. If you specify the INFO keyword, then omit the MF keyword or specify MF=L. Note that the E-form does not provide for updating any of these values. This means that you can specify INFOLIST to refer to a DEVTYPE expansion that is assembled in a reentrant CSECT.

DEVTYPE will return information in the area that you supply in the *area_addrx* parameter. Each type of information that you request is of a fixed length, as stated below. At execution time, the DEVTYPE macro will check whether the area is long enough. The information will be returned in the order in which you specify the values with the INFO= keyword. If the requested information has no meaning for the device or data set, then DEVTYPE clears the appropriate storage. This means that each piece of information will be returned at a predictable offset in your area.

Compatibility note: IBM can add support to DEVTYPE for new INFO= codes in a future level of the system. DEVTYPE in the current level is expected to tolerate object code assembled on such a future level. DEVTYPE in the current level is expected to return the appropriate number of bytes. They should be binary zeroes or partial information padded on the right with binary zeroes. In either case DEVTYPE will also set a return code=0 with a reason code=4 to indicate this condition. If the information returned by the future code cannot validly be zero, then you can test for zeroes to determine which field or fields are not supported. You can also test the DFA to determine the level of the system.

DEVTYPE is not designed to support downward compatibility of source code; that is, you will not be able to assemble source code containing future INFO values on the current level.

DEVTYPE
returns the 4-byte UCBTYP field of the UCB.

Simulated UCBTYP: Any reference to one of the following types in Figure 59 causes the information shown to be placed in the output area.

Figure 59 (Page 1 of 2). Simulated Device UCBTYP Information

Data Set Type	Output in Hexadecimal
DUMMY or TCAM application process queue	0000 0000
TSO terminal	0000 0101

Figure 59 (Page 2 of 2). Simulated Device UCBTYP Information

Data Set Type	Output in Hexadecimal
SYSIN, SYSOUT, or subsystem (SUBSYS=)	0000 0102
HFS file	0000 0103

DASD
returns 16 bytes as follows:

Bytes 0-3 Number of cylinders on the device, excluding alternates. For a VIO data set, this number is the number of simulated cylinders needed to contain the data set.

Bytes 4-7 Number of tracks per cylinder.

Bytes 8-9 Flags (two bytes)

1... .... ECKD supported. The Define Extent, Locate Record and related data transfer commands are supported. For VIO data sets, this bit is on because those commands are always supported.

.1.. .... Locate Record Extended CCW is supported. For VIO data sets, this bit is on because those commands are always supported.

..1. .... Controller cache supported.

...x xxxx Reserved

Byte 9 Reserved

Bytes 10-11 Reserved.

Byte 12 Track set size. Zero if device does not support read-any or write-any. See *3990 Reference* for more information.

Bytes 13-15 Reserved. DEVTYPE currently returns zeroes but could return something different in a future release.

SUFFIX
returns in two bytes the length of the suffix that the system adds to each block in an extended format data set that can be stored on the device. Use this information for space calculations such as with the TRKCALC macro and for determining optimal block size. For device types that support extended format data sets, DEVTYPE returns 32. A non-zero value does not mean that the data set actually is extended format or that the device supports extended format data sets. Not all storage controllers can support extended format data sets. In a future release this value might change.

Device Characteristics Information

The following information is placed into your area as a result of issuing a DEVTYPE macro if you do not code the INFOLIST parameter.

Simulated Device Characteristics: Any reference to one of the following types in Figure 60 causes the information shown to be placed in word 0 and word 1 of the output area.

Figure 60. Simulated Device Characteristics Information

Data Set Type	Word 0 in Hexadecimal	Word 1 in Hexadecimal
DUMMY or TCAM application process queue	0000 0000	0000 0000
TSO terminal	0000 0101	0000 7FF8
SYSIN, SYSOUT, or subsystem (SUBSYS=)	0000 0102	0000 7FF8
HFS file	0000 0103	0000 7FF8

Word 0

Describes the device as defined in the UCBTYP field of the UCB.

Word 1

Maximum block size. For direct access devices, this value is the smaller of either the maximum size of a nonkeyed block or the maximum block size allowed by the operating system; for magnetic tape devices, this value is the maximum block size allowed by the access methods. For these and other device types, see Figure 61 on page 175.

If your program specifies either DEVTAB or UCBLIST without INFOLIST, the next three words contain the following information about direct access devices:

Word 2

Bytes 0-1 The number of physical cylinders on the device, including alternates.

Note: Before you use bytes 0 and 1, read the description of word 4, byte 1, bit 0. For a VIO data set, that bit is zero, and the number of cylinders is as many as are needed to contain the simulated data set. This can differ from the number for the real device being simulated.

Bytes 2-3 The number of tracks per cylinder.

Word 3

Bytes 0-1 Maximum track length. Note that this value is not equal to the value in word 1 (maximum block size).

Byte 2 Block overhead, keyed block—the number of bytes required for gaps and check bits for each keyed block other than the last block on a track.

Note: Before using bytes 2 and 3, read the description of word 4.

Byte 3 Block overhead—the number of bytes required for gaps and check bits for a keyed block that is the last block on a track.

Bytes 2-3 Block overhead—the number of bytes required for gaps and check bits for any keyed block on a track including the last block. Use of this form is indicated by a 1 in bit 4, byte 1 of word 4.

Basic overhead—the number of bytes required for the count field. Use of this form is indicated by a 1 in bit 3, byte 1 of word 4.

Word 4

Byte 0 Block overhead, block without key—the number of bytes to be subtracted from word 3, bytes 2 or 3 or bytes 2 and 3, if a block is not keyed.

If bit 3, byte 1 of word 4 is 1, this byte contains the modulo factor for a modulo device.

Byte 1

Bit 0 If on, the number of cylinders, as indicated in word 2, bytes 0 and 1 is not valid.

Bit 1 If on, Define Extent, Locate Record, and related data transfer commands are implemented for this device. For VIO data sets, this bit is on because those commands are always supported.

Bits 2-3 If both on, indicates the drive is attached to a cache storage control.

Bit 3 If on, indicates a modulo device (such as 3380, 3390).

Bit 4 If on, bytes 2 and 3 of word 3 contain a halfword giving the block overhead for any block on a track, including the last block.

Bit 5 If on, the device supports paging CCWs.

Bit 6 If on, the device has no alternate cylinders.

Bit 7 If on, a tolerance factor must be applied to all blocks except the last block on the track.

Bytes 2-3 Tolerance factor—this factor is used to calculate the effective length of a block. The calculation should be performed as follows:

Step 1 Add the block's key length to the block's data length.

Step 2 Test bit 7 of byte 1 of word 4. If bit 7 is 0, perform step 3. If bit 7 is 1, multiply the sum computed in step 1 by the tolerance factor. Shift the result of the multiplication 9 bits to the right.

Step 3 Add the appropriate block overhead to the value obtained above.

If bit 3, byte 1 of word 4 is 1, bytes 2 and 3 contain the overhead for the data or key field.

If your program specifies DEVTAB and RPS, or specifies UCBLIST without INFOLIST, the next word contains the following information:

Word 5

Bytes 0-1 R0 overhead for sector calculations

Byte 2 Number of sectors for each track

Byte 3 Number of data sectors for each track

Figure 61 on page 175 and Figure 62 on page 175 show the output for each device type that results from issuing the DEVTYPE macro.

Note: If your program specifies UCBLIST and not INFOLIST, the output consists of one 6-word entry for every UCB address contained in the UCB list.

Figure 61. Output from DEVTYPE Macro

IBM Device[1]	Maximum Record Size
2540 Reader	80
2540 Punch	80
2501 Reader	80
3890 Document Processor	80
3505 Reader	80
3525 Punch	80
1403 Printer	120 [1]
3203 Model 5 Printer	132
3211 Printer	132 [1]
3262 Model 5 Printer	132
4245 Printer	132
4248 Printer	132 [2]
3800 or 3900 Printing Subsystem	136 [3]
3410, 3420, 3422, 3424 [4] 3430, 3480, 3490, 3590 Tape Units	32760

Note:

1. Although certain models can have a larger line size, the minimum line size is assumed.
2. The IBM 4248 Printer returns 132 characters even if the 168 Print Position Feature is installed on the device.
3. The IBM 3800 Printing Subsystem can print 136 characters per line at 10-pitch, 163 characters per line at 12-pitch, and 204 characters per line at 15-pitch. The machine default is 136 characters per line at 10-pitch.
4. The 3424 Magnetic Tape Unit is available only in Brazil, S.A.

Figure 62 (Page 1 of 2). Output from DEVTYPE Macro — DASD Devices

IBM Device	Maximum Record Size (Word 1, Decimal)	DEVTAB (Words 2, 3, and 4, in Hexadecimal)	RPS (Word 5, in Hexadecimal)
3380 Models AD4, AJ4, BD4, BJ4, and CJ2 Disk Storage	32 760	0376 000F BB60 0100 2010 010B	04E0 DED6
3380 Models AD4, AJ4, BD4, BJ4, Disk Storage (attached to a cache storage control)	32 760	0376 000F BB60 0100 2030 010B	04E0 DED6
3380 Models AE4 and BE4 Disk Storage	32 760	06EB 000F BB60 0100 2010 010B	04E0 DED6
3380 Models AE4 and BE4 Disk Storage (attached to a cache storage control)	32 760	06EB 000F BB60 0100 2030 010B	04E0 DED6
3380 Models AK4 and BK4 Disk Storage	32 760	0A60 000F BB60 0100 2010 010B	04E0 DED6
3380 Models AK4 and BK4 Disk Storage (attached to a cache storage control)	32 760	0A60 000F BB60 0100 2030 010B	04E0 DED6
3390 Model 1 (attached to a 3990 Model 2)	32 760	0459 000F E5A2 0000 0052 0000	0594 E000
3390 Model 1 (attached to a 3990 Model 3)	32 760	0459 000F E5A2 0000 0072 0000	0594 E000
3390 Model 2 (attached to a 3990 Model 2)	32 760	08B2 000F E5A2 0000 0052 0000	0594 E000
3390 Model 2 (attached to 3990 Model 3)	32 760	08B2 000F E5A2 0000 0072 0000	0594 E000
3390 Model 3 (attached to a 3990 Model 2)	32 760	0D0B 000F E5A2 0000 0052 0000	0594 E000
3390 Model 3 (attached to a 3990 Model 3)	32 760	0D0B 000F E5A2 0000 0072 0000	0594 E000
3390 Model 3 (attached to a 3990 Model 6)	32 760	0D0B 000F E5A2 0000 0072 0000	0594 E000

Figure 62 (Page 2 of 2). Output from DEVTYPE Macro — DASD Devices

IBM Device	Maximum Record Size (Word 1, Decimal)	DEVTAB (Words 2, 3, and 4, in Hexadecimal)	RPS (Word 5, in Hexadecimal)
3390 Model 9 (attached to a 3990 Model 2)	32 760	2721 000F BC98 0000 0052 0000	0594 E000
3390 Model 9 (attached to a 3990 Model 3)	32 760	2721 000F BC98 0000 0520 0000	0594 E000
3390 Model 9 (attached to a 3990 Model 6)	32 760	2721 000F BC98 0000 0520 0000	0594 E000
9345 Model 1	32 760	05A0 000F BC98 0000 0052 0000	04A0 D500

Note: For all currently supported devices, DEVTYPE does not return enough information to perform space calculations. TRKCALC should be used to perform space calculations. For information on using the TRKCALC macro, see "Performing Track Calculations" on page 198.

DEVTYPE—Return Codes and Reason Codes

Control is returned to your program at the next executable instruction following the DEVTYPE macro instruction. Register 15 contains a return code from the DEVTYPE macro, and register 0 contains the reason code. The return codes and their meanings are as follows:

Return Code	Meaning	
0 (X'00')	Information has been successfully stored in your work area.	
	Reason Code	**Meaning**
	0 (X'00')	All the information is available.
	4 (X'04')	DEVTYPE did not recognize one or more INFO parameter codes. DEVTYPE cleared the appropriate amount of the return area and processed the rest of the INFO codes.
4 (X'04')	Invocation error.	
	Reason Code	**Meaning**
	4 (X'04')	DD name not defined.
	8 (X'08')	Parameter list not valid. The error might be version code, length, zero field, or return area is not large enough. DEVTYPE does not do a complete UCB check in the current release. The UCB address might not be valid because the correct value was not written for the third value of the UCBLIST parameter.
8 (X'08')	Unsupported device class.	
	Reason Code	**Meaning**
	12 (X'0C')	DEVTYPE does not support the device class. It must be DASD, tape, subsystem (including spooled), unit record, TSO terminal, dummy, communications, graphics or channel-to-channel adapter. If UCBLIST was coded, then DEVTYPE has ignored the rest of the list.

DEVTYPE—Example 1—Referring to a DD Statement

```
         DEVTYPE MYDD,DEVINFO,DEVTAB
         •
         •
         •
MYDD     DC      CL8'DATATAB'
DEVINFO  DC      5F'0'
```

Example 1 of DEVTYPE returns 20 bytes of device information if DATATAB is a DASD data set or 8 bytes otherwise.

DEVTYPE—Example 2—Includes Building a Parameter List

```
         MVC   DTLIST,KDTLIST     BUILD PARAMETER LIST IN DYNAMIC STORAGE
***********************************************************************
* RETRIEVE FOUR BYTE UCBTYP FOR SYSUT1 DEVICE
***********************************************************************
         DEVTYPE MF=(E,DTLIST),,(AREA,L'AREA)
         •
         •
         •
************************************************************************
* RETRIEVE 20 BYTES (DASD INFO AND UCBTYP) FOR THE UNIT DESCRIBED BY THE
* UCB THAT UCBAD POINTS TO.  THE AREA ADDRESS AND LENGTH ARE STILL
* IN THE PARAMETER LIST FROM THE DEVTYPE EXECUTION PERFORMED ABOVE
************************************************************************
         DEVTYPE UCBLIST=(UCBAD,1),INFOLIST=ILIST2,MF=(E,DTLIST)
         •
         •
         •
KDTLIST  DEVTYPE FIRSTDD,MF=L,INFOLIST=ILIST1  NON-MODIFIABLE PARAMETER
*                                               LIST
LDTLIST  EQU   *-KDTLIST
FIRSTDD  DC    CL8'SYSUT1'
ILIST1   DEVTYPE INFO=DEVTYPE
ILIST2   DEVTYPE INFO=(DASD,DEVTYPE)  REQUEST DATA AT AREA
DYNAMIC  DSECT
UCBAD    DS    A                      ADDRESS OF UCB
DTLIST   DS    CL(LDTLIST)            DEVTYPE PARAMETER LIST(MODIFIABLE)
         DS    0F                     ALIGNMENT FOR TRKCYL
AREA     DS    0CL20                  INFORMATION FROM
*                                       DEVTYPE INFO=(DASD,DEVTYPE)
TYP1     DS    0CL4                   INFORMATION FROM
*                                       DEVTYPE INFO=DEVTYPE (UCBTYP)
*                                       FOR FIRSTDD
NUMCYL   DS    F                      NUMBER OF CYLINDERS ON VOLUME
TRKCYL   DS    F                      NUMBER OF TRACKS PER CYLINDER
         DS    CL8                    MISCELLANEOUS
TYP2     DS    CL4                    UCBTYP FROM UCB POINTED TO
*                                       FROM UCBAD
```

Example 2 of DEVTYPE builds a parameter list in dynamically-acquired storage so the program can be reentrant. It then supplies additional parameters in the first execute form and overrides some of them in the second execute form. In effect, the first specification of DEVTYPE is:

```
DEVTYPE FIRSTDD,(AREA,L'AREA),INFOLIST=ILIST1
```

ILIST1 describes four bytes to be returned. They are at the beginning of a 20-byte area. The list form at KDTLIST specifies parameters that will not be overridden by

the first execute form. The execute form specifies parameters that are determined during execution. In effect, the second specification of DEVTYPE is:

```
DEVTYPE ,(AREA,L'AREA),INFOLIST=ILIST2,UCBLIST=(UCBAD,1)
```

The INFOLIST describes 20 bytes to be returned.

The first execute form illustrates an unusual technique of coding a keyword parameter (MF) before two positional parameters. The first positional value is null and the second is (AREA,L'AREA). This generally is not a good technique because it is confusing. It is used here only to show the flexibility that Assembler H and High Level Assembler allow.

For another example of DEVTYPE, see Figure 72 on page 208.

Reading and Modifying a Job File Control Block

To accomplish the functions that are performed as a result of an OPEN macro instruction, the open routine requires access to information that you have supplied in a data definition (DD) statement. This information is stored by the system in a job file control block (JFCB).

In certain applications, you might find it necessary to modify the contents of a JFCB (previously specified in the allocation parameters) before issuing an OPEN macro instruction against a data set. For example, suppose you are adding records to the end of a sequential data set. You might want to add a secondary allocation quantity to allow the existing data set to be extended when the space currently allocated is exhausted. To assist you, the system provides the RDJFCB macro instruction. This macro instruction causes a JFCB to be moved to an area specified in an exit list. Use of the RDJFCB macro instruction with an exit list is shown under “Example” on page 181. When you subsequently issue the OPEN macro instruction, you can specify the TYPE=J operand to open the data set using the JFCB in the area you specified.

You can use RDJFCB and a DCB to learn the data set name, AMP parameters and volume serials of a VSAM data set. You can use any valid combination of MACRF and DSORG in the DCB. The simplest would be DSORG=PS,MACRF=R. You cannot use the JFCB with OPEN TYPE=J to open a VSAM data set.

The RDJFCB macro also allows you to retrieve allocation information (all JFCBs and all volume serial numbers) for the data sets in a concatenation. You can either select data sets or, by default, retrieve the information for all data sets in the concatenation. “Type 07 JFCB Exit List Entry” on page 182 describes how you can use RDJFCB to retrieve this information.

Note: If you set the bit JFCNWRIT in the JFCBTSDM field to 1 before you issue the OPEN macro instruction, the JFCB is not written back at the conclusion of open processing. OPEN TYPE=J normally moves your program's modified copy of the JFCB, to replace the system copy. To ensure that this move is done, your program must set bit zero of the JFCBMASK+4 field to 1. IBM recommends not setting on JFCNWRIT. If the user JFCB (which the system used to open the data set) is not written back, errors can occur during termination processing for EOV, CLOSE, or the job/step because OPEN might have updated information in the user JFCB which will not be reflected in the system copy of the JFCB. For example, when a nonspecific

tape data set is opened, OPEN will update the user supplied JFCB with the volume serial number of the tape selected. However, the system copy of the JFCB will not reflect this volume serial number. This could cause errors during termination processing for EOV, CLOSE, or the job/step (for example, the data set might not be cataloged even though the job requested it).

Some of the modifications that can be made to the JFCB include:

- Moving the creation and expiration date fields of the DSCB into the JFCB
- Modifying the number-of-volumes field in the JFCB
- Moving the DCB fields from the DSCB into the JFCB
- Adding volume serial numbers to the JFCB (see “RDJFCB Security” on page 183)

  Volume serial numbers in excess of five are written to the JFCBX (extension). The JFCBX cannot be modified by user programs.
- Modifying the data set sequence number field in the JFCB
- Changing the data set name field or member field in the JFCB. See “RDJFCB Security” on page 183 and “RDJFCB Use by Authorized Programs” on page 183.
- Setting bit JFCDQDSP in field JFCBFLG3 to invoke the tape volume DEQ at demount facility (see “DEQ at Demount Facility for Tape Volumes” on page 190)
- Modifying the JFCRBIDO field in the JFCB to cause high-speed positioning to a specific data block on a tape volume on a device that supports cartridge tapes (see “High-Speed Cartridge Tape Positioning” on page 192)
- Setting bit JFCNDSCB to prevent OPEN from merging fields from the data set label to the JFCB. This means that you must supply the information from other sources, such as the JFCB or the DCB. A data set label is a DSCB or standard tape label. This bit does not control all fields. For example it does not control the creation and expiration dates. Note that if you set this bit on, it can cause incorrect positioning in the data set after an automatic step restart when using DISP=MOD or the OPEN macro with the EXTEND option. Setting this bit on can interfere with the system correcting JFCBDSCB (TTR of DSCB on first volume). Setting this bit on is not a good programming practice because it depends on internal system logic.
- Setting bit JFCNDCB to prevent OPEN from merging fields from the DCB to the JFCB. This interferes with correct information being set in the data set label and it interferes with OPEN doing certain checks. Setting this bit on is not a good programming practice because it depends on internal system logic.

Note: The secondary allocation quantity will be moved from the DSCB into the JFCB unless prevented by the setting of JFCNWRIT or JFCNDSCB.

RDJFCB Macro Specification

The RDJFCB macro instruction moves a job file control block (JFCB) into an area of your choice as identified by the EXLST parameter of the DCB macro for each data control block specified.

The format of the RDJFCB macro is:

```
►►──┬──────────┬──RDJFCB──┬─(dcb_addr)──────────────────────────────┬──►◄
    └─label────┘          │                ┌──────────────────────┐
                          └─(──┬─dcb_addr──┬─,──────────────────────┴─)─┘
                               ┌──────────┘ ├─,─┤ option1 ├──────┤
                               │            ├─,(─┤ option1 ├─)────┤
                               │            ├─,(─┤ option2 ├─)────┤
                               │            └─,(,─┤ option2 ├─)────┘
```

option1:

```
├──┬─INPUT──┬──────────────────────────────────────────────────────┤
   ├─EXTEND─┤
   ├─OUTPUT─┤
   ├─INOUT──┤
   ├─OUTIN──┤
   ├─OUTINX─┤
   ├─RDBACK─┤
   └─UPDAT──┘
```

option2:

```
├──┬─,DISP──┬──────────────────────────────────────────────────────┤
   ├─,LEAVE─┤
   ├─,REREAD┤
   └─,REWIND┘
```

Note: If you wish to have multiple DCBs with or without options, code each DCB (and options) as shown in the diagram and precede each additional DCB with a comma. Examples of the standard form of the RDJFCB macro are:

```
RDJFCB (DCB1)
RDJFCB (DCB1,INPUT)
RDJFCB (DCB1,(INPUT))
RDJFCB (DCB1,(INPUT,REREAD))
RDJFCB (DCB1,,DCB2)
RDJFCB (DCB1,,DCB2,(INPUT,REREAD),DCB3,INPUT)
```

Figure 63. Examples of Standard Form of the RDJFCB macro.

*dcb_address,*or *(options)*

(same as the *dcb_address*, *option1*, and *option2* operands of the OPEN macro instruction, as shown in *DFSMS/MVS Macro Instructions for Data Sets*), except for the MODE operand, which is not valid with the RDJFCB macro.

The option operands do not affect RDJFCB processing. You can, however, specify them in the list form of the RDJFCB macro instruction and refer to the generated parameter list with the execute form of the macro.

Notes:

1. You can also use the MF parameter on an RDJFCB macro. Its syntax, use, and effect are the same as is documented for the OPEN macro in *DFSMS/MVS Macro Instructions for Data Sets* . In addition, you can code an execute-form RDJFCB macro that refers to a list-form OPEN macro that does not have MODE=31.
2. The RDJFCB parameter list, the DCB, and the JFCB area specified in the exit list as well as the exit list itself must reside below 16MB, although the calling program can be above 16M.

Example

In Figure 64, the macro instruction at EX1 creates a parameter list for two data control blocks: INVEN and MASTER. In creating the list, both data control blocks are assumed to be opened for input; option2 for both blocks is assumed to be DISP. The macro instruction at EX2 moves the system-created JFCBs for INVEN and MASTER into the area you specified, thus making the JFCBs available to your problem program for modification. The macro instruction at EX3 modifies the parameter list entry for the data control block named INVEN and indicates, through the TYPE=J operand, that the problem program is supplying the JFCBs for system use.

```
EX1       RDJFCB (INVEN,,MASTER),MF=L
          .
          .
          .
EX2       RDJFCB MF=(E,EX1)
          .
          .
          .
EX3       OPEN (,(RDBACK,LEAVE)),TYPE=J,MF=(E,EX1)
          .
          .
          .
INVEN     DCB      EXLST=LSTA,...
MASTER    DCB      EXLST=LSTB,...
LSTA      DS       0F
          DC       X'07'
          DC       AL3(JFCBAREA)
          .
          .
          .
JFCBAREA  DS       0F,176C
          .
          .
          .
LSTB      DS       0F
          .
          .
          .
```

Figure 64. Example Code Using RDJFCB Macro

Multiple data control block addresses and associated options can be specified in the RDJFCB macro instruction. This facility makes it possible to read several job file control blocks in parallel.

An exit list address must be provided in each DCB specified by an RDJFCB macro instruction. Each exit list must contain an active entry of either or both types supported by RDJFCB.

RDJFCB processes the first of each of the two types of its entries in the exit list. For example, in a three-entry list containing types 07, 07 and 13, RDJFCB will process the first and third entries and ignore the second entry. An ignored entry has no effect on the RDJFCB return code.

Each of the entries is briefly explained in the following text. A full discussion of the exit list and its use is contained in *DFSMS/MVS Using Data Sets* .

After RDJFCB is performed register 15 contains one of the following codes:

Figure 65. Return Codes from the RDJFCB Macro

Return Code	Meaning
0 (X'00')	RDJFCB function completed successfully.
4 (X'04')	One or more DCBs encountered one of the following conditions and were skipped. DCBs that were not skipped were processed successfully. • The DCB was being processed by Open/Close/EOV or a similar function. • No data set with the DDNAME that is in the DCB is allocated. • The DCB is not open and its DDNAME is blank.
8 (X'08')	One or more DCBs had an ARL that could not be processed. Each ARL contains a reason code describing its status. One or more DCBs might have encountered a condition described under return code 4. This type of ARL does not contain a reason code.

Type 07 JFCB Exit List Entry

The type 07 JFCB exit list entry allows you to perform a variety of tasks, as described in the following text.

The format of the type 07 JFCB exit list entry is:

The virtual storage area into which the JFCB is read must be:

- Located within the user's region
- On a word boundary
- At least 176 bytes long.

Note: Users running in 31-bit addressing mode must ensure that this area is located below 16MB virtual. Each exit list entry must be 4 bytes long. The system recognizes only the first occurrence of an exit list entry code. Indicate the end of the exit list by setting the high-order bit in the entry code byte to 1.

The DCB can be either open or closed when this macro instruction is executed. If accessing concatenated sequential data sets and the DCB is open, the RDJFCB routine reads the JFCB for the current data set. In all other cases, the RDJFCB routine reads the JFCB for the first or only data set.

If the RDJFCB routine fails while processing a DCB associated with your RDJFCB request, or you do not provide a virtual storage address in the three low-order bytes of the exit list entry, your task is abnormally terminated. None of the options available through the DCB ABEND exit, as described in *DFSMS/MVS Using Data Sets*, are available when a RDJFCB macro instruction is issued.

RDJFCB Security: OPEN TYPE=J compares the volume serial numbers specified in the user-supplied JFCB are compared with the volume serial numbers in the system's copy of the JFCB. Each different volume serial number will be enqueued exclusively. The volumes stay enqueued until the job step terminates, because the CLOSE routines will not dequeue the volumes. If the job step already has the volume open, OPEN TYPE=J continues. If the volume is enqueued by another job step, an ABEND 413 occurs with a return code of `X'04'`.

Some JFCB modifications can compromise the security of existing password-protected or RACF-protected data sets. The following modifications are specifically not allowed, unless the program making the modifications is authorized or can supply the password:

- Changing the disposition of a password-protected data set from OLD or MOD to NEW.
- Changing the data set name or one or more of the volume serial numbers when the disposition is NEW.
- Changing the label processing specifications to bypass label processing.

Notes:

1. An authorized program is one that is either in supervisor state, executing in one of the system protection keys (keys 0 through 7), or authorized under the authorized program facility.
2. Changing the data set name in the JFCB has no effect on the name of the data set. You are just referring to a different data set.

RDJFCB Use by Authorized Programs: Except when opening a VTOC, if you change the data set name in the JFCB and your job step is APF-authorized, you should do a system enqueue on the major name of “SYSDSN” for the substituted data set name. Issuing an ENQ macro for a major name of SYSDSN requires authorization.

If your program changes the data set name or the volume serial number and is not authorized, OPEN TYPE=J calls dynamic allocation for the new name. CLOSE will automatically unallocate the data set.

To use the correct interface with other system functions (for example, partial release), the ENQ macro should include the TCB of the initiator and the length of the data set name (with no trailing blanks). When you complete processing of the data set, you should use the DEQ macro to release the resources. If your program is not authorized and issues an OPEN TYPE=J macro, and the substituted data set

name is already enqueued by another job step, an ABEND 913 occurs with a return code of X'1C'.

To open a VTOC data set to change its contents (that is, open it for OUTPUT, OUTIN, INOUT, UPDAT, OUTINX, or EXTEND), your program must be authorized under the Authorized Program Facility (APF). APF provides security and integrity for your data sets and programs. Details on how to authorize your program are provided in *OS/390 MVS Assembler Services Guide* and *OS/390 MVS Assembler Services Reference*.

You cannot extend a VTOC by this means.

Note: Do not change the data set name to NULLFILE (signifying a dummy data set). Changing the name to NULLFILE can prevent the device allocated for the data set specified on the DD statement from being deallocated at job/step termination.

Using RDJFCB to Process a Multivolume Direct Data Set: This use of RDJFCB and OPEN TYPE=J permits your program to process a multivolume data set. To do so, your program must cause the open routines to build a DEB for each volume and issue mount messages for them. Your program must use the RDJFCB macro to read in the JFCB, and open each volume of the data set. The coding example in Figure 66 on page 185 illustrates the procedure.

Return code 0 indicates that the JFCB has been read successfully for all DCBs in the parameter list. Otherwise, see Figure 65 on page 182.

```
         RDJFCB      DCB1              READS IN THE JFCB
         SR          R3,R3             CLEARS REG 3; IT WILL
*                                      HOLD COUNT OF VOLS TO
*                                      BE OPENED
         IC          R3,JFCBNVOL       PUTS # OF VOLS
*                                      IN REG 3
         LA          R4,DCB1           R4 POINTS TO DCB FOR
*                                      VOL TO BE OPENED
         LA          R5,1              PUTS SEQUENCE # OF
*                                      FIRST VOL TO BE
*                                      OPENED IN REG 5
LOOP     EQU         *
         STH         R5,JFCBVLSQ       PUTS SEQ # OF VOL
*                                      TO BE OPENED WHERE
*                                      OPEN RTNS LOOK
         OPEN ((R4),OUTPUT),TYPE=J     OPENS ONE VOL
*    NOTE THAT THE TYPE=J OPTION OF THE MACRO MUST BE USED
         LA          R4,DCB2-DCB1(R4)  INCREMENT REG 4 TO
*                                      POINT TO THE DCB FOR
*                                      THE NEXT VOL TO BE
*                                      OPENED
         LA          R5,1(R5)          INCREMENT TO SEQ # OF
*                                      NEXT VOL TO BE OPENED
         BCT         R3,LOOP           LOOP UNTIL ALL VOLS
*                                      OPEN
         .
         .
         .
JFCB     DS          CL176            JFCB READ IN HERE
         ORG         JFCB+70
JFCBVLSQ DS          H                SEQ # OF VOL TO BE
*                                     OPENED
         ORG         JFCB+117
JFCBNVOL DS          FL1              # OF VOLS IN DATA SET
         ORG
* MAPPING MACRO IEFJFCBN MAY ALSO BE USED
DCB1 DCB DDNAME=SYSUT1,MACRF=(E),EXLST=EXITS,DSORG=PS
DCB2 DCB DDNAME=SYSUT1,MACRF=(E),EXLST=EXITS,DSORG=PS
DCB3 DCB DDNAME=SYSUT1,MACRF=(E),EXLST=EXITS,DSORG=PS
DCB4 DCB DDNAME=SYSUT1,MACRF=(E),EXLST=EXITS,DSORG=PS
DCB5 DCB DDNAME=SYSUT1,MACRF=(E),EXLST=EXITS,DSORG=PS
DCB6 DCB DDNAME=SYSUT1,MACRF=(E),EXLST=EXITS,DSORG=PS
DCB7 DCB DDNAME=SYSUT1,MACRF=(E),EXLST=EXITS,DSORG=PS
* THIS PROCEDURE WORKS FOR 5 VOLS OR LESS.
EXITS    DS          0F
         DC          X'87',AL3(JFCB)   87 IDENTIFIES THIS AS
*                                      THE EXIT LIST ENTRY
*                                      THAT SHOWS WHERE JFCB
*                                      WILL BE READ IN
```

Figure 66. Processing a Multivolume Direct Data Set with RDJFCB

Type 13 JFCB Exit List Entry

The type 13 JFCB exit list entry allows you to retrieve selected allocation information, as described in the following text. To determine whether the level of your system supports exit list entry type 13, set the ARLAREA field in the ARL to zeros before issuing RDJFCB. If the ARLAREA is still zeros and the ARLRCODE field contains reason code 00 when RDJFCB returns control to your program, the level of DFP in your system is earlier than MVS/XA DFP* Version 2 Release 3. Note that in this case you must supply a type 07 entry. The system will also accept both types, even if both are active.

The format of the type 13 JFCB exit list entry is:

Using RDJFCB to Retrieve Allocation Information: RDJFCB uses DCB exit list entry type 13 to retrieve allocation information (JFCBs and volume serial numbers) for concatenated data sets. The exit list entry code is X'13', and is defined as "retrieve allocation information." The second through fourth bytes of this entry must point to an allocation retrieval list (ARL), as described in Figure 67 on page 187. If you issue RDJFCB, this DCB exit list entry causes retrieval of JFCBs for the specified concatenated data sets, and lists of all volume serial numbers for these data sets. You can either select JFCBs in the concatenation or, by default, retrieve all of them. RDJFCB uses the parameter list to receive and return information about the request. See Figure 69 on page 189 for an example of usage.

Note: OPEN TYPE=J does not recognize this exit list entry. You can use the IHAARL macro (shown here) to generate and map the ARL. Your program might issue a GETMAIN or STORAGE macro for the ARL, or, if you specify DSECT=NO, the ARL is generated within your program's storage. The ARL must be below 16MB. The allocation retrieval area (ARA), acquired by RDJFCB through a GETMAIN macro, can be above or below 16MB.

The format of the IHAARL macro is:

```
>>──┬──────────┬──IHAARL──┬──────────────────────────┬──┬──────────────────────────┬──>
    └─label────┘          │          ┌─YES─┐         │  └─,PREFIX=prefix───────────┘
                          └─DSECT=───┴─NO──┴─────────┘

>──┬──────────────────────────────────────────────────┬──><
   └─,DESCR=──┬─NO──┬──────────────────────────────────┘
              └─YES─┘
```

DSECT=YES or **NO**
specifies whether the symbol at the beginning of the generated area appears on a DSECT instruction or a DC instruction. For DSECT=NO, the symbol appears on a DC instruction. The default is DSECT=YES.

PREFIX=*prefix*
allows you to invoke the macro more than once per assembly. Specifies a character string with which all generated symbols are to be prefixed. Do not specify delimiters, such as quotation marks. If you omit this operand or specify a null value, the prefix defaults to the characters ARL.

DESCR=YES or **NO**
specifies whether the macro expansion includes the macro description (prolog). The default is DESCR=NO.

Figure 67 on page 187 and Figure 68 on page 188 describe the formats of the allocation retrieval *list* and allocation retrieval *area*, respectively.

Figure 67 (Page 1 of 2). Format of the Allocation Retrieval List

Name	Offset	Bytes	Description
The Following Fields Are Set by the Caller of RDJFCB.			
ARLLEN	0 (X'00')	2	Length of this area. Value should be 36.
ARLIDENT	2 (X'02')	2	EBCDIC 'AR'
ARLOPT1	4 (X'04')	1	Option byte.
ARLLANY		0	ARA must be below 16MB.
		1	ARA can be above 16MB.
		. xxx xxxx	Reserved. Must be zero.
ARLRSVD1	5 (X'05')	7	Reserved. Must be zero.
ARLRETRV	12 (X'0C')	2	Number of data sets for which to retrieve information. If 0, retrieve all in the concatenation.
ARLFIRST	14 (X'0E')	2	Number of first data set in concatenation for which to retrieve information. 0 or 1 specifies retrieval of information beginning with first data set in the concatenation.
The Following Fields Are Set by RDJFCB.			
ARLAREA	16 (X'10')	4	Address of ARA. See Figure 68 on page 188.
ARLPOOL	20 (X'14')	1	Storage subpool containing ARA.
ARLRLEN	21 (X'15')	3	Length of ARA.
ARLRTRVD	24 (X'18')	2	Number of concatenated data sets for which JFCBs were retrieved.
ARLCONC	26 (X'1A')	2	Number of concatenated data sets. If no concatenation, this value is 1.
ARLRCODE	28 (X'1C')	1	Reason Code: 0 = Requested information was read. The following reason codes are related to return code 8: 4 = ARLFIRST is greater than ARLCONC. 8 = Insufficient storage to read information. ARLPOOL and ARLRLEN describe what could not be obtained.

Figure 67 (Page 2 of 2). Format of the Allocation Retrieval List

Name	Offset	Bytes	Description
ARLRSVD2	29 (X'1D')	7	Reserved. Used by RDJFCB.

Figure 68. Format of the Allocation Retrieval Area

Offset	Bytes	Description
0 (X'00')	2	Length of the information for this data set (including this field). The length is rounded up to a multiple of four so the starting address of the ARA plus the value in the length field designates the address of the ARA for the next data set in the concat-enation, if requested.
2 (X'02')	2	Reserved. Set to zero.
4 (X'04')	176(Dec)	JFCB
180 (X'B4')	variable	Sixth and subsequent volume serial numbers. Determined by the value in JFCBNVOL. If the number of volume serial numbers is fewer than the specified volume count, entries at the end of the list might contain all blanks. If the first byte of an entry is X'FF', the JCL specified VOL=REF and the volume could not be deter-mined.

When you have finished using information from the retrieval areas, you should issue FREEMAIN or STORAGE macro to free any areas that were acquired through GETMAIN (including the ARA, acquired by RDJFCB). When coding the FREEMAIN or STORAGE macro, specify the *length*, *subpool*, and *address* values from the ARLRLEN, ARLPOOL, and ARLAREA fields, respectively. Code the FREEMAIN macro as shown:

```
FREEMAIN RU,LV=length,SP=subpool,A=address
```

If RDJFCB successfully fills in the ARL field register 15 is set to zero. Otherwise see Figure 65 on page 182.

Example: In Figure 69 on page 189, the macro instruction at ALLOCINF creates a parameter list for one DCB (INDCB), assumed to be open for input. The JFCBs and volume serial numbers are retrieved for all data sets allocated to DDNAME SYSLIB.

```
  ***JCL FOR FOLLOWING INVOCATION OF RDJFCB:

//SYSLIB DD    DISP=SHR,DSN=DEPT61.ROUTINES
//       DD    DISP=SHR,DSN=CORPORAT.ROUTINES
//       DD    DISP=SHR,DSN=SYS1.FORTLIB


  ***EXAMPLE CODE TO INVOKE RDJFCB ALLOCATION INFORMATION RETRIEVAL:

*   GET A COPY OF THE JFCB FOR THE FIRST OR ONLY DATA SET ALLOCATED
*   TO SYSLIB AND TRY TO READ THE JFCBS AND VOLUME SERIAL NUMBERS
*   FOR ALL DATA SETS ALLOCATED TO SYSLIB.
*
ALLOCINF RDJFCB (INDCB)
         LTR   R15,R15          TEST RDJFCB RETURN CODE
         BNZ   NOJFCB           BRANCH IF INFORMATION NOT AVAILABLE
         ICM   R1,X'F',SLBAREA  GET AND TEST ADDRESS OF ARL
         BZ    OLDSYSTM         GO IF SYSTEM DOES NOT SUPPORT ARL
         CLI   SLBRCODE,0       TEST RDJFCB REASON CODE
         BNE   NOJFCB           BRANCH IF INFORMATION NOT AVAILABLE
*
*  LOOP THROUGH THE JFCBS IN THE AREA TO WHICH SLBAREA POINTS.
*  CODE CAN BE INSERTED HERE TO PRINT THE DATA SET NAMES AND
*  VOLUME SERIAL NUMBERS.
         .
         .
         SR    R2,R2
         IC    R2,SLBPOOL
         SR    R0,R0
         ICM   R0,B'0111',SLBRLEN
         FREEMAIN RU,LV=(0),SP=(R2),A=SLBAREA
         .
         .
OLDSYSTM DS    0H               ROUTINE TO HANDLE JUST LIBJFCB
         .
         .
*
NOJFCB   DS    0H               ROUTINE TO HANDLE INABILITY TO GET THE
*                               JFCB. THE DATA SET MAY NOT BE ALLOCATED.
         .
         .
*
```

Figure 69 (Part 1 of 2). Sample Code Retrieving Allocation Information

```
SLBOPNX  DS    0H               DCB OPEN EXIT ROUTINE FOR SYSLIB.
*                               HANDLES RECFM, LRECL, AND BLKSIZE.
         .
         .
INDCB    DCB   DSORG=PO,DDNAME=SYSLIB,MACRF=R,SYNAD=INERROR,           X
               EXLST=INEXLST
INEXLST  DC    0F'0',X'05'      ENTRY CODE FOR OPEN EXIT ROUTINE
         DC    AL3(SLBOPNX)     ADDR OF DCB OPEN EXIT ROUTINE
         DC    X'13'            ENTRY CODE TO RETRIEVE ALLOCATION
*                                INFORMATION
         DC    AL3(SLBSTRT)     ADDR OF ALLOCATION RETRIEVAL LIST
         DC    X'87'            ENTRY CODE TO RETRIEVE FIRST JFCB AND
*                                INDICATE LAST ENTRY IN LIST
         DC    AL3(LIBJFCB)     ADDR OF JFCB FOR FIRST DATA SET
*
*  AN ALLOCATION RETRIEVAL LIST FOLLOWS, POINTED TO BY DCB EXIT LIST.
*
SLBSTRT  IHAARL DSECT=NO,PREFIX=SLB
         DC    0F'0'
LIBJFCB  DC    CL176' '         FIRST JFCB
         .
         .
```

Figure 69 (Part 2 of 2). Sample Code Retrieving Allocation Information

DEQ at Demount Facility for Tape Volumes

This facility is intended to be used by long-running programs that create an indefinitely long tape data set (such as a log tape). Use of this facility by such a program permits the processed volumes to be allocated to another job for processing (such as data reduction). This processing is otherwise prohibited unless the indefinitely long data set is closed and dynamically deallocated.

You can invoke this facility only through the RDJFCB/OPEN TYPE=J interface by setting bit JFCDQDSP (bit 0) in field JFCBFLG3 at offset 163 (X'A3') to 1. The volume serial of the tape is dequeued when the volume is demounted by OPEN or EOV with message IEC502E when all the following conditions are present:

- The tape volume is verified (where a tape is considered *verified* after file protect, label type, and density conflicts have been resolved) for use by OPEN or EOV (see page 191 for more information concerning *verified*).
- JFCDQDSP is set to 1.
- The program is APF authorized (protect key and supervisor/problem state are not relevant).
- The tape volume is to be immediately processed for output. That is, either OPEN verifies the volume and the OPEN option is OUTPUT, OUTIN, or OUTINX; or EOV verifies the volume and the DCB is opened for OUTPUT, OUTIN, INOUT, or EXTEND, and the last operation against the data set was an output operation (DCBOFLWR is set to 1).

Note that, for EOV to find JFCDQDSP set to 1, the program must not inhibit the rewrite of the JFCB by setting bit 4 of JFCBTSDM to 1.

The tape volume is considered verified after file protect, label type, and density conflicts have been resolved. The volume is dequeued when demounted after this verification, even if further into OPEN or EOV processing the volume is rejected because of expiration date, security protection, checkpoint data set protection, or an I/O error.

When the volume serial is dequeued, the volume becomes available for allocation to another job. However, because the volume DEQ is performed without deallocating the volume, care must be exercised both by the authorized program and the installation to prevent misuse of the DEQ at demount facility. A discussion of such misuse follows:

- The authorized program must not close and reopen the data set using the tape volume DEQ at demount facility. If it does, one of the following can occur:

  1. The dequeued volume can be mounted and in use by another job. When the volume is requested for mounting, for the authorized program, the operator is unable to satisfy the mount. Therefore, the operator must either cancel the requesting job, cancel the job using the volume, wait for the requesting job to time out, or wait for the job using the volume to terminate.
  2. The dequeued volume can be allocated to another job, but not yet in use. The operator mounts the volume to satisfy the mount request of the authorized job. When the volume is requested for mounting by the other job, the operator is unable to satisfy the mount request, and is faced with the same choices as in the previous item.
  3. The dequeued volume can not yet be allocated to another job and the volume is mounted to satisfy the mount request of the authorized job. Another job can allocate the volume and, when the volume is requested for mounting, the situation is the same as in the previous item.

  It is the responsibility of the installation that permits a program to run with APF authorization to ensure that it does not close and reopen a data set using the DEQ at demount facility.

- Care should be exercised when an authorized program uses the DEQ at demount facility (data set 1), but processes another tape data set (data set 2). Assume the same volume serial numbers have been coded in the DD statements for data set 1 and data set 2. As the volumes of data set 1 are demounted, they are dequeued even though those volumes still might be requested for data set 2. All the problems explained in items 1, 2, and 3 can occur as data set 2 and another job contend for a dequeued volume.

  This problem should not occur, given the intended use of the DEQ at demount facility; that is, a long-running application creating an indefinitely long tape data set. This type of application is not normally invoked through batch execution with user-written DD statements.

- After a volume has been demounted and dequeued because of the DEQ at demount facility, the volume is not automatically rejected by the control program when mounted in response to a specific or nonspecific mount request. Without the use of the facility, the control program can recognize (by the ENQ) that the volume is in use, and reject the volume. Therefore, operations procedures, in effect to prevent incorrect volumes from being mounted, should be reviewed in the light of reduced control program protection from such errors when the DEQ at demount facility is used. Specifically, if a volume is remounted for an authorized program and the volume had been used previ-

ously by that authorized program, duplicate volume serial numbers will exist in the JFCB, and the control program will be unable to release the volume during EOV processing.

- Checkpoint/restart considerations are discussed in *DFSMS/MVS Checkpoint/Restart* .

High-Speed Cartridge Tape Positioning

High-speed positioning for cartridge tape is available when opening a tape data set on an IBM standard-labeled tape for either EXTEND (OUTINX, EXTEND, or DISP=MOD), or when opening to the beginning of such a data set. To invoke high-speed positioning, your program must modify certain fields in the JFCB and use OPEN TYPE=J to open the data set.

Use the following procedure to modify the JFCB:

1. Issue the RDJFCB macro to have the system move the JFCB into your work area.
2. Set the JFCPOSID flag in the JFCBFLG3 flag byte to indicate that you are providing a block ID for a high-speed search.
3. Move the block ID into the JFCRBIDO field of the JFCB. If you are opening to the beginning of a data set, use the block ID of the first header label record of that data set. If you are opening to the end of a data set (for example, to extend it), use the block ID of the tape mark immediately following the last block of user data in that data set.
4. Issue the OPEN TYPE=J macro to have the system use your modified JFCB.

After the tape is positioned, OPEN processes the trailer labels for the data set being extended.

If you set the JFCPOSID flag off, OPEN positions the volume normally, as though the high-speed positioning feature were not active.

If you set the JFCPOSID flag on, but do not provide a block ID in the JFCRBIDO field, OPEN positions the volume normally and does one of the following:

- If you are opening to the beginning of a data set, OPEN inserts the block ID of the first header label record of that data set into the JFCRBIDO field.
- If you are opening to the end of the data set, OPEN inserts the block ID of the tape mark immediately following the last block of user data for that data set into the JFCRBIDO field.

OPEN does not update your copy of the JFCB. To retrieve the new value in the system's copy of the JFCB, issue RDJFCB after OPEN.

If the JFCPOSID flag is on during CLOSE processing, (because you set it on before OPEN), CLOSE inserts the block ID for the first header label record of the next data set (which might not exist) into the JFCRBIDC field. Therefore, if you unallocate the cartridge tape device and want to use the current block ID for subsequent processing, save the block ID before you close the data set.

OPEN resets the JFCPOSID flag if any one of the following conditions exists:

- Your program issues an OPEN which is not TYPE=J

- The requested tape volume is not an IBM standard-labeled volume
- The requested unit is not a buffered tape device.

Notes:

1. If you specify dynamic deallocation (with SVC99, FREE=CLOSE on the DD statement, or the FREE option on the CLOSE macro), the block ID for the next data set will not be available to your program.
2. When using high-speed positioning, specify the data set sequence number normally, either explicitly by LABEL=(seqno,SL) on the DD statement, or by default.

After the system routines have used the JFCRBIDO field for high-speed positioning, they clear JFCRBIDO in the system's copy of the JFCB to prevent misinterpretation during a subsequent OPEN.

OPEN—Initialize Data Control Block for Processing the JFCB

The OPEN macro instruction initializes one or more data control blocks (DCBs) so that their associated data sets can be processed.

A full explanation of the operands of the OPEN macro instruction is contained in the publication *DFSMS/MVS Macro Instructions for Data Sets*. The TYPE=J option, because it is used in conjunction with modifying a JFCB, should be used only by the system programmer or under the system programmer's supervision.

The format of the OPEN TYPE=J macro is:

```
►►──┬──────────┬──OPEN──┬─(dcb_addr)──────────────────────────────────►◄
    └─label────┘        │   ┌──────────────────────────────────┐
                        │   ↓                                  │
                        ├─(─dcb_addr──┬──────────────────────┬─┴─)──┤
                        │             ├─,─┤ option1 ├──────────┤      │
                        │             ├─,(─┤ option1 ├─)────────┤      │
                        │             ├─,(─┤ option2 ├─)────────┤      │
                        │             └─,(,─┤ option2 ├─)──────┘      │
                        └─,TYPE=J──────────────────────────────────┘
option1:
    ┌─INPUT──┐
├───┼─EXTEND─┼──────────────────────────────────────────────────────┤
    ├─OUTPUT─┤
    ├─INOUT──┤
    ├─OUTIN──┤
    ├─OUTINX─┤
    ├─RDBACK─┤
    └─UPDAT──┘
option2:
    ┌─,DISP──┐
├───┼─,LEAVE─┼──────────────────────────────────────────────────────┤
    ├─,REREAD┤
    └─,REWIND┘
```

Note: If you wish to have multiple DCBs with or without options, code each DCB (and options) as shown in the diagram and precede each additional DCB with a comma.

```
OPEN (DCB1),TYPE=J
OPEN (DCB1,INPUT),TYPE=J
OPEN (DCB1,(INPUT)),TYPE=J
OPEN (DCB1,(INPUT,REREAD)),TYPE=J
OPEN (DCB1,,DCB2),TYPE=J
OPEN (DCB1,,DCB2,(INPUT,REREAD),DCB3,INPUT),TYPE=J
```

Figure 70. Examples of Standard Form of the OPEN TYPE=J Macro

TYPE=J

specifies that, for each DCB referred to, you have supplied a job file control block (JFCB) to be used during initialization. A JFCB is an internal representation of information in a DD statement.

During initialization of a data control block, its associated JFCB can be modified with information from the DCB or an existing data set label or with system control information.

When the TYPE=J operand is specified, also supply a DD statement. However, the amount of information given in the DD statement is at your discretion, because you can modify many fields of the system-created JFCB. If you specify DUMMY on your DD statement, the open routine ignores the JFCB DSNAME and opens the data set as dummy. (See Figure 64 on page 181 for an example of coding that modifies a system-created JFCB.)

Notes:

1. The DD statement must specify at least:
   - Device allocation (see *OS/390 MVS JCL User's Guide* for methods of preventing share status)
   - A ddname corresponding to the associated DCB DCBDDNAM field.
2. The MODE operand is not shown here because it is not allowed with the TYPE=J operand of the OPEN macro instruction.
3. Since OPEN with TYPE=J does not accept a JFCBX from the caller, you cannot change volume serials after the first five volumes.
4. OPEN TYPE=J will not change the volume attributes (PRIVATE, PUBLIC, or STORAGE) which are assigned to the volume during allocation. For example, if a volume status of PRIVATE is needed but allocation is going to assign a status of PUBLIC, then VOL=PRIVATE should be specified on the DD statement.

Purging and Restoring I/O Requests

The system's purge routines perform either a halt or a quiesce operation. In a halt operation, the purge routines stop the processing of specified I/O requests initiated with an EXCP or EXCPVR macro instruction. In a quiesce operation, the purge routines:

- Allow the completion of I/O requests (initiated with an EXCP or EXCPVR macro instruction) that were passed to the system for execution and are executing

- Stop the processing of requests that have not yet been initiated or passed to the system, but save the IOBs of the requests so they can be reprocessed (restored) later.

The system's restore routines make it possible to reprocess I/O requests that are quiesced.

Note: Purge and restore processing performed for I/O requests that are not initiated by an EXCP or EXCPVR macro is not covered here. User applications that use the PURGE and RESTORE macros with the sequential access method (SAM) against partitioned data sets (PDSs) (for example, to synchronize the I/O) cannot do so against PDSEs, sequential extended format data sets, or HFS files, because SAM does not use EXCP or EXCPVR to access these types of data.

To pass control to the purge and restore routines, build a parameter list and place its address in register 1, then issue the macro instruction.

24-bit or 31-bit addressing mode can be used for the PURGE or RESTORE macro (and the parameter list).

PURGE Macro Specification

The PURGE macro is used to halt or finish I/O requests.

The format of the PURGE macro is:

```
►►──┬──────────┬──PURGE──parameter_list_address──────────────────────►◄
    └─label────┘
```

parameter_list_address—RX-type address, (2-12) or (1)
address of a parameter list, 12 or 16 bytes long, that you have built on a word boundary in storage. The parameter list address can be specified as an RX-type address or in registers 2 through 12, or 1. The name of the mapping macro is IECDPPL.

The format and contents of the parameter list are as follows:

Byte	Contents	
0	A byte that specifies the actions of the purge routines. The bit settings and their meanings are:	
	1...	Purge I/O requests to a single data set. The setting of this bit only takes effect if bit 2 of byte 12 is 0 and bit 6 of byte 0 is 0.
	0...	Either purge I/O requests associated with a TCB or address space, or purge I/O requests to more than one data set. If bit 2 of byte 12 is 1, then the request is to purge I/O associated with an address space. If bit 2 of byte 12 is 0 and bit 6 of byte 0 is 1, then the request is to purge I/O associated with a TCB. If bit 2 of byte 12 is 0 and bit 6 of byte 0 is 0, then the request is to purge I/O to more than one data set.

.1..	Post ECBs associated with purged I/O requests.
..1.	Halt I/O-request processing. (Quiesce I/O-request processing, if 0.)
...1	Purge related requests. (Only valid if a data-set purge is requested.)
.... 0...	Reserved—must be zero.
.... .1..	Do not purge the TCB request-block chain of asynchronously scheduled processing.
.... ..1.	Purge I/O requests associated with a TCB. The setting of this bit will only take affect if bit 2 of byte 12 is 0.
....1	This is a 16-byte parameter list. Additional purge options are specified in bytes 12 to 15. (If this bit is off, the list is 12 bytes long, and the purge routines do not put a return code in byte 4 of this list or in register 15.)

1,2,3 The address of a DEB when purging I/O requests to a single data set. The address of the first DEB in a chain of DEBs when purging I/O requests to more than one data set. (The next-to-the-last word of each DEB must point to the next DEB in the chain; the second word of the last DEB must contain zeros.)

4 A byte of zeros. (If bit 7 of byte 0 is on, the purge routines will put a code in byte: `X'7F'` when the purge operation is successful; `X'40'` when it is not successful. If bit 7 of byte 0 is off, then `X'7F'` appears in this byte.)

5,6,7 If you turned on bit 6 of byte 0, the address of the TCB associated with the I/O requests you want purged. Will be zeros if the TCB is the one you are running under.

8 Value of `X'00'` or `X'02'` means that EXCP is the owner.

9,10,11 The address of a word in storage or the address of the DEBUSPRG field (that is `X'11'` bytes more than the DEB address in this parameter list). At the address you specify, the purge routines store a pointer to the purged I/O restore list, that in turn contains a pointer to the first IOB in the chain of IOBs. The location of the pointer and format of the chain are shown in Figure 71 on page 198.

Note: This field is only relevant for quiesce options.

12 A byte that allows you to specify additional purge options. The bit settings and their meanings are:

Note: The following applies only if bit 7 of byte 0 is set to 1.

..1.	Purge I/O requests associated with an address space. (Your program must be in supervisor state.) The setting of this bit will take affect regardless of the setting of bit 6 of byte 0 and bit 0 of byte 0.
...1	If this is a data-set purge, check the validity of all the DEBs associated with the purge operation. Validate this parameter list, whatever the type of purge operation, by ensuring that there are no inconsistencies in the selection of purge options.

		(If your program is in problem state, these actions are taken regardless of the bit setting.)
	.... 1...	Ensure that I/O requests will be reprocessed (restored) under their original TCB. (If zero, and bit 7 of byte 0 is on, the I/O requests are reprocessed under the TCB of the program making the restore request.)
	.... .0..	Must be zero.
13	A byte of zeros.	
14,15	If bit 2 of byte 12 is on, the 2-byte ID of the address space associated with the I/O requests you want purged.	

Control is returned to your program at the instruction following the PURGE macro instruction.

Return Codes from PURGE

If the purge operation was successful, register 15 contains zeros. Otherwise, register 15 contains one of the following return codes:

Return Code	Meaning
4 (X'04')	Your request to purge I/O requests associated with a given TCB was not honored because that TCB did not point to the job step TCB, while the requester is in problem state.
8 (X'08')	Either you requested an address-space purge operation, but were not in supervisor state, or you requested a data set purge operation, but failed to supply a DEB address in bytes 1, 2, and 3 of the purge parameter list.
20 (X'14')	Another purge request has preempted your request. You might want to reissue your purge request in a time-controlled loop.

Note: If you set bit 7 in byte 0 of the parameter list to zero, register 15 will contain zeros, regardless of the outcome of the purge operation.

Modifying the IOB Chain

This procedure is not recommended. However, to change the order in which purged I/O requests are restored or prevent a purged request from being restored, you can change the sequence of IOBs in the IOB chain or remove an IOB from the chain. The address of the IOB chain can be obtained from the purge I/O restore list (see Figure 71 on page 198). (The address of the purge I/O restore list is shown at bytes 9 through 11 of the purge parameter list.) Note that some IOBs could be in a different protection key.

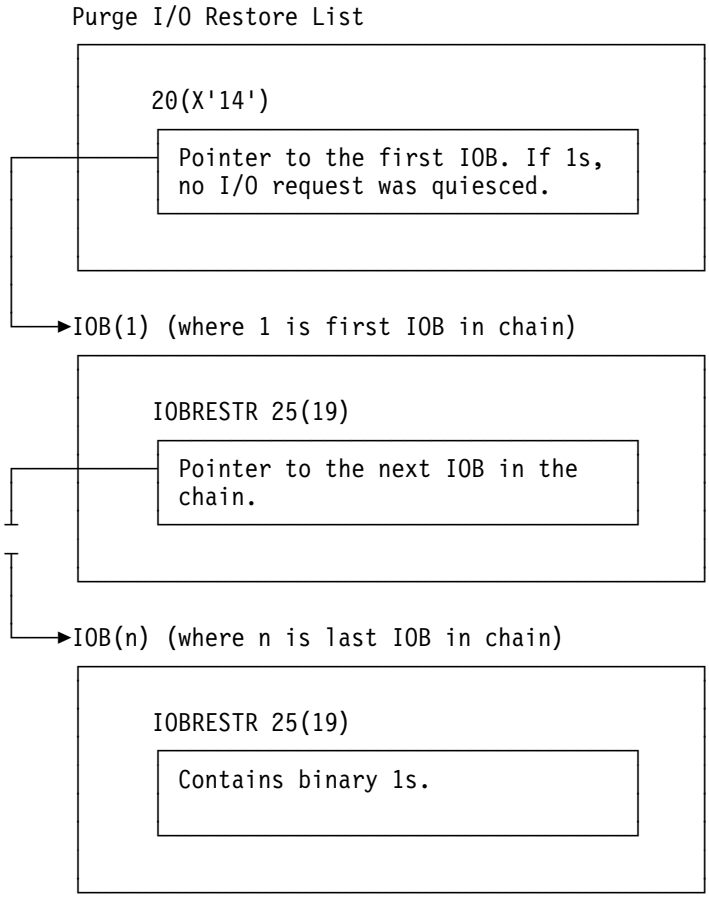


Figure 71. The IOB Chain

RESTORE Macro Specification

The RESTORE macro is used to reprocess I/O requests.

The format of the RESTORE macro is:

```
►►──┬──────────┬──RESTORE──restore_address──────────────────────────►◄
    └─label────┘
```

restore_address—RX-type address, (2-12) or (1)
address you specified at byte 9 of the purge parameter list. See 196 for information about byte 9.

Performing Track Calculations

The TRKCALC macro performs DASD track capacity calculations. Using TRKCALC, you can:

- Perform track capacity calculations
- Determine the number of records that can be written on a full track or on the remainder of a track

- Perform track balance calculations as follows:
  - Determine if a record can be written in the space remaining on the track and return the new track balance
  - Determine the maximum size record that can be written on the track if the given record does not fit
  - Determine the track balance if the last physical record is removed from the track.

The TRKCALC routine issues no SVC instructions. TRKCALC can be called in an SRB routine or in TCB mode. It can be called in 24-bit or 31-bit addressing mode and in supervisor or problem state.

Non-EXCP user applications cannot expect consistent information from TRKCALC for PDSEs, because of the unique structure and format of PDSEs. However, processing will complete without error indications.

TRKCALC does not support HFS files. You will receive unpredictable results if you use TRKCALC for HFS files.

TRKCALC Macro Specification

The standard, list, execute, and DSECT forms of the macro are described. Examples of the TRKCALC macro follow the macro descriptions.

TRKCALC—Standard Form

The format of the TRKCALC macro is:

```
►►──listname──TRKCALC──FUNCTN=──┬─TRKBAL─┬──┬─,DEVTAB=addr─┬──────────►
                                └─TRKCAP─┘  ├─,UCB=addr────┤
                                            └─,TYPE=addr───┘

►──┬─,BALANCE=addr─┬──┬──────────────┬──┬──────────────┬──────────────►
   │               │  │   ┌─BELOW─┐  │  │   ┌─NO─┐     │
   │               │  └─,LOC=──┴─ANY───┴──┘  └─,REMOVE=──┴─YES──┴──┘

►──┬──────────────┬──┬─,RKDD=addr──────────────────────┬──────────────►
   │   ┌─NO─┐     │  └─,R=addr,K=addr,DD=addr──────────┘
   └─,MAXSIZE=──┴─YES──┴──┘

►──┬──────────────┬──┬──────────────┬──────────────────────────────►◄
   │   ┌─NO─┐     │  │   ┌─I─┐     │
   └─,REGSAVE=──┴─YES──┴──┘  └─,MF=──┴───┴──┘
```

FUNCTN=TRKBAL or **TRKCAP**
specifies the function to be performed.

Note: Specify one of the three keywords, DEVTAB, UCB, or TYPE, to provide the information source for the macro.

TRKBAL
if REMOVE=NO is specified, TRKBAL calculates whether an additional record fits on the track and what the new track balance would be if the record were added. If REMOVE=YES is specified, TRKBAL calculates what the track balance would be if a record were removed from the track. The

record to be added or removed from the track is defined by the RKDD parameter, or by the R, K, and DD parameters.

If R is equal to 1 (or the R value in the RKDD parameter is 1) and REMOVE=NO is specified, record 1 is added to an empty track; if R is equal to 1 and REMOVE=YES is specified, record 1 is deleted from the track, leaving an empty track.

If R is not equal to 1, the specified record is added to or removed from the track. If the input track balance is not supplied through the BALANCE parameter, it is assumed that the track contains equal-sized records as specified in the RKDD parameter (or in the R, K, and DD parameters).

When REMOVE=NO is specified, one of the following occurs:

- If the record fits on the track, register 0 contains the new track balance.
- If the record does not fit on the track and MAXSIZE=NO is specified, a record does not fit return code is placed in register 15.
- If the record does not fit and MAXSIZE=YES is specified, one of the following happens:
  - The data length of the largest record that fits in the remaining space is returned in register 0.
  - A code is returned that indicates no record fits in the remaining space.

When REMOVE=YES is specified, one of the following occurs:

- If R is equal to 1, register 0 contains the track capacity.
- If R is not equal to 1, registers 0 contains the input track balance (supplied through the BALANCE parameter) incremented by the track balance used by the input record. If the input balance is not supplied, register 0 contains the track capacity left after R–1 records are written on the track.

TRKCAP
calculates, and returns in register 0, the number of fixed-length records that can be written on a whole track (R is equal to 1) or on a partially-filled track (R is not equal to 1). The records are defined by the K and DD values of the RKDD parameter, or by the K and DD parameters.

One of the following occurs:

- If R is equal to 1, the BALANCE parameter is ignored and the calculation is made on an empty track.
- If R is not equal to 1 and the BALANCE parameter is omitted, the calculation is made for a track that already contains R–1 records of the length defined by the K and DD values.
- If R is not equal to 1 and the BALANCE parameter is supplied, the calculation is made for a track whose remaining track balance is the value of the BALANCE parameter.

DEVTAB=*addr*—RX-type address, (2-12), (0), (14)
addr specifies a word that contains the address of the device characteristics table entry (DCTE). If you specify a register, it contains the actual address of

the DCTE. The address of the DCTE can be found in the word beginning at the DCBDVTBL field of an opened DCB.

UCB=*addr*—RX-type address, (2-12), (0), (14)
addr specifies the address of a word that contains the address of the UCB. If you specify a register, it contains the actual address of the UCB.

Note: The TRKCALC macro accepts the address of a UCB or UCB copy. Unauthorized programs can get a copy of the UCB by using the UCBSCAN macro and specifying the COPY and UCBAREA keywords. See *OS/390 HCD Planning* for more information.

TYPE=*addr*—RX-type address, (2-12), (0), (14)
you can specify the address of the UCB device type (UCBTBYT4), or you can specify the 1-byte UCB device type in the low-order byte of a register.

LOC=BELOW or **ANY**
optional parameter indicating whether the value passed by the UCB parameter is a 4-byte or a 3-byte address. This parameter only applies to callers running in AMODE 31. If the caller is running in AMODE 24, this parameter is ignored, i.e., the high-order byte is treated as `X'00'`.

BELOW
The UCB parameter contains a UCB address for a UCB which resides in storage below 16 megabytes, or a captured UCB. This is the default.

Note: If LOC=BELOW is specified, the high-order byte of the UCB address will be treated as `X'00'`.

ANY
The address passed in the UCB parameter contains a 3-byte or 4-byte UCB address.

Note: If LOC=ANY is specified when invoking in 31-bit mode, TRKCALC will treat the UCB address as a 31-bit address.

BALANCE=*addr*—RX-type address, (2-12), (0), (14)
you can specify either the address of a halfword containing the current track balance, or you can specify the balance in the low-order two bytes of a register. The value supplied could be the value returned when you last issued TRKCALC. If R is equal to 1, the balance is reset to track capacity by TRKCALC, and your supplied value is ignored. This is an input value and is not modified by the TRKCALC macro. The resulting track balance is returned in register 0 and in the TRKCALC parameter list field STARBAL. The value you supply for this parameter must be a valid value for the device type in use.

REMOVE=YES or **NO**
indicates if a record is to be deleted from the track.

YES
specifies that the record identified by the record number (specified in the R keyword) is being deleted from the track. The track balance is incremented instead of decremented.

Note: YES is valid only on a FUNCTN=TRKBAL call.

NO
specifies that a record is not to be deleted from the track. NO is the default.

MAXSIZE=YES or **NO**

YES
If the specified record does not fit, the largest length of a record with the specified key length that fits is returned (register 0).

Note: YES is valid only on a FUNCTN=TRKBAL call.

NO
Maximum size is not returned. NO is the default.

RKDD=*addr*—RX-type address, (2-12), (0), (14)
addr specifies a word containing a record number (1 byte), key length (1 byte), and data length (2 bytes) (bytes 0, 1, and 2 and 3, respectively) or a register containing the record number, key length, and data length. R, K, and DD can be specified by this keyword, or you can use the following three keywords instead.

R=*addr*—RX-type address, (2-12), (0), (14), or n
you can specify either the address of the record number, or you can specify the record number using the low-order byte of a register or immediate data (n). Specify a decimal digit for n (immediate data).

K=*addr*—RX-type address, (2-12), (0), (14), or n
you can specify either the address of a field containing the hexadecimal value of the record's key length, or you can specify the record's key length using the low-order byte of a register or immediate data (n). Specify a decimal digit for n (immediate data).

DD=*addr*—RX-type address, (2-12), (0), (14), or n
you can specify either the address of a field containing the hexadecimal value of the record's data length, or you can specify the record's data length using the low-order two bytes of a register or immediate data (n). Specify a decimal digit for n (immediate data).

REGSAVE=YES or **NO**
specifies whether registers are to be saved.

YES
specifies registers 1 through 14 are saved and restored in the caller-provided save area (pointed to by register 13) across the TRKCALC call. Otherwise, registers 1, 9, 10, 11, and 14 are modified. Registers 0 and 15 are always modified by a TRKCALC call.

NO
specifies registers are not saved across a TRKCALC call. NO is the default.

MF=I
specifies storage definition for the TRKCALC parameter list and parameter list initialization, using the given keywords, then calling the TRKCALC function. MF=I is the default.

TRKCALC—Execute Form

A remote parameter list is referred to and can be modified by the execute form of the TRKCALC macro. The TRKCALC routine is called. The function of the operands is the same as for the standard form.

The format of the execute form of the TRKCALC macro is:

```
►►──┬──────────┬──TRKCALC──┬──────────────────────────────────────────►
    └─label────┘           └─FUNCTN=──┬─TRKBAL─┬──┬──────────────────►
                                      └─TRKCAP─┘  │
►──┬──────────────────────────────────────────────┬──┬──────────────────►
   │  ┌──────────────────────────────────────────┐ │  │  ┌─BELOW─┐    │
   │  ├─,DEVTAB=──┬─addr─┬──┬──────────────────┤ │  │  └─,LOC=──┴─ANY───┘
   │  ├─,UCB=─────┬─addr─┬──┤                  │ │  │
   │  │           └─*────┘  │                  │ │  │
   │  └─,TYPE=────┬─addr─┬──┘                  │ │  │
   │              └─*────┘                     │ │  │
►──┴──┬──────────────────────────────────────┬──┴──┬──────────────────►
      │  ┌──────────────────────────────────┐ │     │
      └─,BALANCE=──┬─addr─┬──┬──────────────┤ │     │  ┌─NO──┐
                   └─*────┘  │              │ │     └─,REMOVE=──┴─YES─┘
►──┬──────────────────────────────────────────┬──┬──────────────────────►
   │  ┌─NO──┐                                 │  │  ┌──────────────────┐
   └─,MAXSIZE=──┴─YES─┘                       │  └──┴─,RKDD=addr────────┴
   │                                          │     ├─,R=addr,K=addr,DD=addr─┤
►──┴──────────────────────────────────────────┴──,MF=(E,addr)──────────►◄
      ┌─NO──┐
      ├─,REGSAVE=──┴─YES─┘
      │
```

FUNCTN=TRKBAL or **TRKCAP**
is coded as shown in the standard form. If this keyword is omitted, any specification of REMOVE, MAXSIZE, LAST, and the RX form of BALANCE is ignored. In addition, DEVTAB is assumed if UCB is coded and a failure occurs, or if TYPE is specified. When you use FUNCTN, one of the keywords (DEVTAB, UCB, or TYPE) must be specified to provide an information source.

DEVTAB=*addr or *—RX-type address, (2-12), (0), (14)*
is coded as shown in the standard form except for the * subparameter. Specify an * when you have inserted the address of the device characteristics table entry (DCTE) in the parameter list.

UCB=*addr or *.—RX-type address, (2-12), (0), (14)*
is coded as shown in the standard form except for the * subparameter. Specify an * when you have inserted the address of the UCB in the parameter list.

Note: The TRKCALC macro accepts the address of a UCB or UCB copy. Unauthorized programs can get a copy of the UCB by using the UCBSCAN macro and specifying the COPY and UCBAREA keywords. See *OS/390 HCD Planning* for more information.

TYPE=*addr or *—RX-type address, (2-12), (0), (14)*
is coded as shown in the standard form except for the * subparameter. Specify an * when you have inserted the address of the UCB type (UCBTYP) in the parameter list.

LOC=BELOW or **ANY**
is coded as shown in the standard form.

BALANCE=*addr or *—RX-type address, (2-12), (0), (14)*
is coded as shown in the standard form except for the * subparameter. Specify an * when you have inserted the balance in the parameter list.

REMOVE=YES or **NO**
is coded as shown in the standard form.

MAXSIZE=YES or **NO**
is coded as shown in the standard form.

RKDD=*addr*—RX-type address, (2-12), (0), (14)
is coded as shown in the standard form.

R=*addr*—RX-type address, (2-12), (0), (14) or n
is coded as shown in the standard form.

K=*addr*—RX-type address, (2-12), (0), (14), or n
is coded as shown in the standard form.

DD=*addr*—RX-type address, (2-12), (0), (14), or n
is coded as shown in the standard form.

REGSAVE=YES or **NO**
is coded as shown in the standard form.

MF=(E,*addr***)**
This operand specifies that the execute form of the TRKCALC macro instruction and an existing data management parameter list are used.

E Coded as shown.

addr—RX-type address, (0), (1), (2-12), or (14)
specifies the address of the parameter list.

TRKCALC—List Form

The list form of the TRKCALC macro constructs an empty, in-line parameter list. By coding only MF=L, you construct a parameter list, and the actual values can be supplied by the execute form of the TRKCALC macro. Any parameters other than MF=L are ignored.

The format of the list form of the TRKCALC macro is:

```
►►──┬──────────┬──TRKCALC──MF=L──────────────────────────────────►◄
    └─label────┘
```

TRKCALC—DSECT Only

This call gives a symbolic expansion of the parameter list for the TRKCALC macro. No DSECT statement is generated. If a name is specified on the macro call, it applies, after any necessary boundary alignment, to the beginning of the list. The macro-generated symbols all begin with the characters STAR.

The format of the DSECT form of the TRKCALC macro is:

```
►►──┬──────────┬──TRKCALC──MF=D──────────────────────────────────────►◄
    └─label────┘
```

Input Register Usage for All Forms of MF

Register	Use
0, 2-12, 14	Available to provide input for keywords.
1	Only to provide the address of the parameter list for an MF=E call.
13	Input for keywords if REGSAVE=YES is not specified.
15	Work register to build the TRKCALC parameter list for the MF=E call; it is not available as an input register.

Output from TRKCALC

FUNCTN=TRKBAL

Output	Meaning
R15=X'00'	The record fits on the track. Register 0 and STARBAL contain the new track balance.
R15=X'04'	Record does not fit on the track. If MAXSIZE=YES is specified, a partial record does not fit either. Register 0 and STARBAL are set to zero.
R15=X'08'	Record does not fit on the track. MAXSIZE=YES is specified, and a partial record does fit. Register 0 and STARBAL are set to the maximum number of data bytes that fit on the remainder of the track with the specified key length. **Note:** The key length is excluded from the count of maximum data bytes.
R15=X'0C'	The user supplied a device type, but the device characteristics table indicated that no device of that type was generated on the system. Register 0 is set to zero.
STARBAL	This is the track balance field of the TRKCALC parameter list. This field is first set to the track capacity if R is equal to 1, or to the supplied BALANCE value if R is not equal to 1, or to the calculated balance if R is not equal to 1 and BALANCE are omitted. STARBAL is updated to the new track balance if the record fits; otherwise, STARBAL is left with the input track balance value.

FUNCTN=TRKCAP

Output	Meaning
R15=X'00'	Register 0 contains the number of records that fit on the track if R is equal to 1, or the number of records that fit on the remainder of the track if R is not equal to 1.
R15=X'04'	No records of the length specified fit on a full track (R is equal to 1) or a partial track (R is not equal to 1). Register 0 is set to zero.

R15=X'0C' The user supplied a device type, but the device characteristics table indicated that no device of that type was generated on the system. Register 0 is set to zero.

STARBAL This is the track balance field of the TRKCALC parameter list. This field is first set to the track capacity if R is equal to 1, or to the supplied BALANCE value if R is not equal to 1, or to the calculated balance if R is not equal to 1 and BALANCE is omitted.

Return Codes from TRKCALC

The TRKCALC macro passes a return code in register 15. The return codes and their meanings are as follows:

Return Code	Meaning
0 (X'00')	Indicates that register 0 contains the new track balance.
4 (X'04')	Indicates that the record did not fit (register 0 = 0).
8 (X'08')	Indicates that the record did not fit. (Register 0 contains the maximum data length that does fit.)
12 (X'0C')	The system could not find an entry in the device characteristics table whose attributes match those of the user-specified device type. Register 0 is set to zero.

TRKCALC Macro Examples

In this example, TRKCALC is coded to determine how many records of a given size with 10-byte keys fit on an IBM 3380 track. After issuing the macro, the number of records is saved in NUMREC:

```
         TRKCALC  FUNCTN=TRKCAP,TYPE=UTYPE,R=1,K=10,DD=DL,            X
               MF=(E,(1))
         .
         .
         ST   0,NUMREC   SAVE NUMBER OF RECORDS
         .
         .
DL       DC   H'xxxx'    DATA LENGTH
UTYPE    DC   X'0E'
NUMREC   DS   F          MAX # OF RECORDS
```

In this example, TRKCALC is coded to determine whether another record can fit on a track of a 3380, given a track balance.

```
         TRKCALC  FUNCTN=TRKBAL,TYPE=UTYPE,R=REC,K=KL,DD=DD,          X
               BALANCE=BAL,MAXSIZE=YES,MF=(E,(1))
         .
         .
UTYPE    DC   X'0E'
REC      DC   X'xx'
KL       DC   X'xx'
DD       DC   H'xxxx'
BAL      DC   H'xxxx'
```

Determining Version, Release, and Modification Level of DFSMS/MVS

You can use the IHADFA mapping macro to determine the level of DFSMS/MVS. It maps the data facilities area. Use the CVT mapping macro to define symbol CVTDFA, which points to the DFA. The DFARELS field in the DFA is four bytes that designate the product level.

If the first byte of DFARELS contains a binary value of 0, then your program is not executing on DFSMS/MVS; it is executing on MVS/XA DFP Version 2 or MVS/DFP Version 3 and the following three bytes also contain zeroes. On those two products you can determine the release level by examining the two-byte field DFAREL. DFAREL is described in the comments in IHADFA.

If the first byte of DFARELS contains a value of 1, then your program is running on DFSMS/MVS and the following three bytes designate the version, release and modification level of DFSMS/MVS. A value of X'01010200' in DFARELS designates DFSMS/MVS Version 1, Release 2, Modification level 0.

IBM intends that for any future level of the DFA, the 4-byte DFARELS will not contain a smaller value. If your purpose in testing DFARELS is to determine whether a particular feature of DFSMS/MVS is available, then we suggests that your program test all four bytes of DFARELS. IBM intends that if one of the low-order three bytes of DFARELS contains a value that is smaller than the corresponding byte in the prior release, then a higher order byte will contain a larger value.

For compatibility with programs that were designed to run on MVS/XA DFP Version 2 or MVS/DFP Version 3, DFSMS/MVS sets DFAREL to the value X'3321', which designates MVS/DFP Version 3, Release 3, modification level 2. The last digit indicates that the system actually is at a higher level than DFP 3.3.2.

See also “Call for DFSMS/MVS Level Determination” on page 226 for an alternative method of determining the level of DFSMS/MVS.

See “Data Facilities Area (DFA) Fields” on page 425 for a layout of the fields of the Data Facilities Area (DFA) control block.

Determining DFARELS During Assembler Macro Phase

Your program can test the DFARELS field during execution as described earlier. This does not allow you to assemble a program that optionally uses a new macro parameter that is available only on a certain level of the system. Your program receives syntax error messages if assembled on an older level of the system.

A solution is to test a macro variable symbol set by the IHADFA macro. The name of the symbol is &IHADFARELS and it is a character type of global variable symbol. Your program's test of its value must follow the IHADFA invocation.

The other system facilities determine whether your program can run on a different release than the one on which it was assembled. For some new functions the older release will ignore the new function. Other new functions will fail on an older release.

The IHADFA macro sets the variable symbol &IHADFARELS to an eight-character value. Each pair of characters in the value represents the decimal value of one byte

in DFARELS. They are not hexadecimal digits because the EBCDIC values of “A” to “F” are not in proper collating sequence with the numeric digits.

This is an example of a program using &IHADFARELS:

```
 xxxx    CSECT
           .
           .
           .
         GBLC  &IHADFARELS    Set by IHADFA macro to be system level
         IHADFA ,             Set &IHADFARELS and define DFARELS
 xxxx    CSECT                Reset CSECT
           .
           .
           .
* Expand one of two macro invocations.  Either works on any DFSMS/MVS
* release.  If in 31-bit mode on 1.3 or later, then ANY means a UCB
* may be above the line.  Neither works on DFP Version 3 when assembled
* on DFSMS/MVS.
         AIF   ('&IHADFARELS' LT '01010300').OLD
* If executing in 31-bit mode on 1.3 or later, this requires that each
* UCB address be 31-bit.  They may point below the line.  On an older
* level of DFSMS/MVS, the ANY has no effect.  DFP 3.x will reject it.
         DEVTYPE UCBLIST=(MYLIST,1,ANY),MF=(E,DEVTLIST)
         AGO   .CONT
.OLD     DEVTYPE UCBLIST=(MYLIST,1),MF=(E,DEVTLIST)
.CONT    ANOP
          .
MYLIST   DC    A(0)
DEVTLIST DEVTYPE ,(DEVINFO,24),MF=L
```

Figure 72. Sample &IHADFARELS Program

The IHADFA macro as shipped prior to DFSMS/MVS Version 1.3 did not set &IHADFARELS, but IBM reserves the right to do so. You can use the technique in the example even if IHADFA does not set &IHADFARELS.

This technique of using IHADFA to decide on another macro invocation assumes that IHADFA resides in a complete macro library for the same release as the other macro. It might not work properly with a macro from a different release or product.

Following is an example of determining whether a mapping macro has defined a symbol that is needed during the assembly. During execution, the program tests DFARELS to determine how to execute.

```
         GBLC  &IHADFARELS   Set by IHADFA macro to be system level
         IHADFA ,            Learn release of assembly & execution
TRKLIST  DSECT
         TRKCALC MF=D        DSECT for TRKCALC parameter list
         SPACE 2
* If global symbol &IHADFARELS has a null value or is less than
* 01010300, then TRKCALC did not define a certain symbol.  Since
* other parts of this program use it, it must be defined.
         AIF   ('&IHADFARELS' GE '01010300').GOTBIT   Go if newer
STARLOC  EQU   X'01'     LOC=ANY.  DEVTAB or UCB may be above line
.GOTBIT  ANOP
```

Figure 73. Example of Determining Symbol Definition

Chapter 8. Displaying Messages on Cartridge Magnetic Tape Subsystems

This chapter contains information about using the MSGDISP macro to display messages on magnetic tape devices that have displays. With MSGDISP, you can specify the message to be displayed and how to display it (for example, steady or flashing). The standard, execute, and list forms of the macro are described here. The six main parameters of the macro and their functions are:

Value	Meaning
MOUNT	Displays an `'M'` in position 1 of the display area during a mount request until a volume is loaded and made ready. The `'M'` is followed by the volume serial number and label type.
VERIFY	Shows that a volume has been accepted by displaying its serial number and label type in positions 2 through 8.
RDY	Displays text in positions 2 through 7 while a data set is open.
DEMOUNT	Places a volume disposition indicator in position 1 of the display until a volume is demounted.
RESET	Clears the display area.
GEN	Provides the full range of display options, including the option to alternate two messages.

All except the RDY parameter require that the caller be in supervisor state, have a storage protect key of 0 through 7, or be authorized by the authorized program facility.

You can issue the MSGDISP macro in 24- or 31-bit addressing mode. When you use 31-bit addressing mode, all addresses must be valid 31-bit addresses.

The MSGDISP macro generates a parameter list as input to the message display service routine. You can code an installation exit routine named IGXMSGEX, which gains control when MSGDISP is processing MOUNT, DEMOUNT, VERIFY, or GEN requests. The exit can change the message text displayed (two 8-byte strings) and 1 bit of the format control byte. See the publication *DFSMS/MVS Installation Exits* for details.

MSGDISP Macro Specification

The formats for specifying MSGDISP with the six main parameters, and the return codes generated by MSGDISP, are given in the following sections.

MSGDISP—Displaying a Mount Message

The format of the MSGDISP macro with the MOUNT parameter is:

© Copyright IBM Corp. 1979, 1999

```
►►──┬──────────┬──MSGDISP──MOUNT──,UCB=(reg)──┬──────────────────────────┬──►
    └─label────┘                              └──,LABEL=──┬─'S'─┬──┘
                                                          ├─'A'─┤
                                                          ├─'N'─┤
                                                          ├─'X'─┤
                                                          └─addr┘

►──┬──────────────────────┬──┬──────────────────────┬──────────────────────►
   └──,MF=──┬─L──────────┬──┘  └──,SER=──┬─'volser'─┬──┘
            └─(E,addr)───┘               └─addr─────┘

►──┬──────────────────────┬──┬──────────────────────┬──►◄
   └──,TEST=──┬─NO─┬──┘  └──,WAIT=──┬─YES─┬──┘
              └─YES┘                 └─NO──┘
```

MOUNT
displays an `'M'` in position 1 of the display area during a mount request. The `'M'` is followed by a volume serial number and label type. The display flashes on and off until a volume is loaded and ready. If the device is ready at the time a mount request is issued, the `'M'` is not displayed.

UCB=(*reg***)**—(2-12)
specifies a register containing the UCB address for the device. Use the address of a UCB, not a UCB copy.

LABEL='A' or **'N'** or **'S'** or **'X'** or *addr*
displays the label type of the mounted volume in position 8. If you specify an unknown label type other than a blank, a “?” is displayed.

'A'
specifies ISO/ANSI (AL) or ISO/ANSI with user labels (AUL). Apostrophes are required.

'N'
specifies no labels (NL), LTM (VSE), or bypass label processing (BLP) Apostrophes are required.

'S'
specifies IBM Standard (SL) or IBM Standard with user labels (SUL). Apostrophes are required.

'X'
specifies nonstandard labels (NSL). Apostrophes are required.

addr—RX-type address, A-type address, or (2-12)
specifies the address of an area containing an “A”, “N”, “S”, or “X”. (See the following explanations of these characters.) For MF=L, you can only specify an A-type address.

MF=L or **(E,***addr***)**
specifies either the execute or the list form of MSGDISP. If you do not specify this parameter, the standard form of the macro is used.

L specifies the list form of MSGDISP. This generates a parameter list that can be used as input to the execute form. The execute form can modify the parameter list.

(E,*addr*)
specifies that the execute form of the macro and an existing parameter list are used.

addr—RX-type address, (1), or (2-12)
specifies the address of the parameter list.

SER='*volser*' or *addr*
specifies the serial number of the volume to be mounted. The serial number is displayed in positions 2 through 7. If you do not specify SER, the system supplies the volume serial number. If the serial number is not available, a scratch volume is used, unless the volume use attribute indicates a default of "PRIVAT."

'*volser*'
specifies the volume serial number as a literal. Specify in apostrophes.

addr—RX-type address, A-type address, or (2-12)
specifies the address of the volume serial number. For MF=L, you can only specify an A-type address.

TEST=NO or **YES**
specifies whether the macro expansion is to include code that tests the UCB to determine whether message display is supported. If the result of the test is that the message display is not supported, an SVC is not invoked.

NO
specifies that the macro expansion is not to include code that tests the UCB to determine whether the device supports message display.

YES
specifies testing the UCB by the MSGDISP macro before attempting to invoke the message display service routine.

Note: TEST=YES requires you to include the UCB mapping macro (IEFUCBOB) in the source code.

There is a restriction when using TEST=YES. Programs running in AMODE 24 and invoking the MSGDISP macro with the TEST=YES parameter cannot pass the actual address of a UCB that resides above the 16MB line. These programs must pass the captured UCB address or, if an actual address is passed, the UCB must reside below the 16MB line.

WAIT=NO or **YES**
specifies when control is returned to you.

NO
specifies that the MSGDISP function is not to wait for completion of I/O initiated on the caller's behalf. When MSGDISP returns, the I/O request might still be running. I/O return codes are not returned, and I/O errors are recorded in the same manner as any permanent error by the error recovery procedure.

YES
specifies that control is to be returned after I/O is complete.

MSGDISP—Displaying a Verify Message

The format of the MSGDISP macro with the VERIFY parameter is:

```
►►──┬──────────┬──MSGDISP──VERIFY──,UCB=(reg)──┬──────────┬──┬─'S'─┬──►
    └─label────┘                               └─,LABEL=──┘  ├─'A'─┤
                                                             ├─'N'─┤
                                                             ├─'X'─┤
                                                             └─addr┘

►──┬──────────────────────────┬──┬──────────────────┬──────────────────►
   └─,MF=──┬─L──────────────┬──┘  └─,SER=──┬─'volser'─┬──┘
           └─(E,addr)───────┘              └─addr─────┘

►──┬──────────────────────┬──┬──────────────────────┬──►◄
   └─,TEST=──┬─NO─┬──┘  └─,WAIT=──┬─YES─┬──┘
             └─YES┘                └─NO──┘
```

VERIFY
displays the serial number and label type of a volume that has been accepted for processing. The serial number is displayed on the pod in positions 2 through 7, and the 1 character label type in position 8. Position 1 remains blank. The display lasts until the next display request is executed.

UCB=(*reg***)**—(2-12)
specifies a register containing the UCB address for the device. Use the address of a UCB, not a UCB copy.

LABEL='A' or **'N'** or **'S'** or **'X'** or *addr*
specifies label type of the mounted volume in position 8 of the display. If an unknown label type other than a blank is specified, a “?” is displayed.

'A'
specifies ISO/ANSI (AL) or ISO/ANSI with user (AUL) labels. Apostrophes are required.

'N'
specifies no labels (NL), LTM (VSE), or bypass label processing (BLP). Apostrophes are required.

'S'
specifies IBM Standard (SL) or IBM Standard with user (SUL) labels. Apostrophes are required.

'X'
specifies nonstandard (NSL) labels. Apostrophes are required.

addr—RX-type address, A-type address, or (2-12)
specifies the address of an area containing an “A”, “N”, “S”, or “X” (see explanations below for these characters). For MF=L, you can only specify an A-type address.

MF=L or **(E,***addr***)**
specifies either the execute or list form of MSGDISP. If you do not specify this parameter, the standard form of the macro is used.

L specifies the list form of MSGDISP. This generates a parameter list that can be used as input to the execute form. The execute form can modify the parameter list.

(E,*addr*)
specifies that the execute form of the macro and an existing parameter list is to be used.

addr—RX-type address, (1), or (2-12)
specifies the address of the parameter list.

SER='*volser***'** or *addr*
specifies the serial number of the volume that has been verified. The serial number displays in positions 2 through 7. If you do not specify SER, the system supplies the volume serial number. If the serial number is not available, a scratch volume is used, unless the volume use attribute indicates a default of "PRIVAT".

'*volser***'**
specifies the volume serial number as a literal. Express in apostrophes.

addr—RX-type address, A-type address, or (2-12)
specifies the address of the volume serial number. For MF=L, you can only specify an A-type address.

TEST=NO or **YES**
specifies whether the macro expansion is to include code that will test the UCB to determine whether message display is supported. If the result of the test is that the message display is not supported, an SVC is not invoked.

NO
specifies that the macro expansion is not to include code that tests the UCB to determine whether the device supports message display.

YES
specifies testing the UCB by the MSGDISP macro before attempting to invoke the message display service routine.

Note: TEST=YES requires you to include the UCB mapping macro (IEFUCBOB) in the source code.

Note: If this provision isn't followed, a program check in expansion code might result. Programs running in AMODE 24 and invoking the MSGDISP macro with the TEST=YES parameter cannot pass the actual address of a UCB that resides above the 16MB line. These programs must pass the captured UCB address or, if an actual address is passed, the UCB must reside below the 16MB line.

WAIT=NO or **YES**
specifies when control is to be returned to you and that the MSGDISP function is not to wait for completion of I/O initiated on the caller's behalf. When MSGDISP returns, the I/O request might still be running.

NO
specifies that the MSGDISP function is not to wait for completion of I/O initiated on the caller's behalf. When MSGDISP returns, the I/O request might still be running. I/O return codes are not returned, and I/O errors are recorded in the same manner as any permanent error by the error recovery procedure.

YES
specifies that control is to be returned after I/O is complete.

MSGDISP—Displaying a Ready Message

The format of the MSGDISP macro with the RDY parameter is:

```
►►──┬──────────┬──MSGDISP──RDY──,DCB=addr──┬──────────────────────────┬──►
    └─label────┘                           └─,MF=──┬─L──────────┬──────┘
                                                   └─(E,addr)───┘
►──┬──────────────────────────────────────────────────────────────────────►◄
   └─,TXT=──┬─'msgtxt'──┬──┘
            └─addr───────┘
```

RDY
displays the text supplied in the TXT parameter in positions 2 through 7 while the data set is open. The display is steady (not flashing) and is enclosed in parentheses. The display is also written to the tape pool console (routing code 3, descriptor code 7).

DCB=*addr*
specifies the address of a DCB opened to a data set on the mounted volume. If multiple devices are allocated, the message display is directed to the one containing the volume currently in use.

Note: If multiple devices or multiple volumes are allocated, you can update a message display after an end-of-volume condition by using the EOV user exit specified in a DCB exit list. In the case of a concatenated data set with unlike characteristics, the DCB OPEN exit can be used to update the display.

addr—RX-type address, A-type address, or (2-12)
specifies the address of the opened DCB. For MF=L, you can only specify an A-type address.

MF=L or **(E,***addr***)**
specifies either the execute or list form of MSGDISP. If this parameter is not specified, the standard form of the macro is used.

L specifies the list form of MSGDISP. This generates a parameter list that can be used as input to the execute form. The execute form can modify the parameter list.

(E,*addr***)**
specifies that the execute form of the macro and an existing parameter list is to be used.

addr—RX-type address, (1), or (2-12)
specifies the address of the parameter list.

TXT='*msgtxt***'** or *addr*
specifies up to six characters to display in positions 2 through 7 of the display. If you do not specify TXT, blanks are displayed.

'*msgtxt***'**
specifies the text as a literal. Express in apostrophes.

addr—RX-type address, A-type address, or (2-12)
specifies the address of an area containing the text to be displayed. For MF=L, you can only specify an A-type address.

MSGDISP—Displaying a Demount Message

The format of the MSGDISP macro with the DEMOUNT parameter is:

```
►►──┬──────────┬──MSGDISP──DEMOUNT──,UCB=(reg)──┬──────────────────────────┬──►
    └─label────┘                                │                          │
                                                └─,DISP=─┬─'D'─┬──────────┘
                                                         ├─'K'─┤
                                                         ├─'R'─┤
                                                         └─addr┘
►──┬──────────────────────┬──┬──────────────────────┬──┬──────────────────────┬──►
   └─,MF=──┬─L──────────┬──┘  └─,MLABEL=─┬─'S'─┬──┘  │                      │
           └─(E,addr)───┘                ├─'A'─┤     │                      │
                                         ├─'N'─┤     │                      │
                                         ├─'X'─┤     │                      │
                                         └─addr┘     │                      │
►──┬──────────────────────────────┬──┬──────────────────────────┬──►
   └─,MSER=──┬─'volser_to_mount'─┬──┘  └─,SER=──┬─'volser'─┬──┘
             └─addr────────────┘                └─addr─────┘
►──┬──────────────────────┬──┬──────────────────────┬──►◄
   └─,TEST=──┬─NO──┬──┘  └─,WAIT=──┬─YES─┬──┘
             └─YES─┘               └─NO──┘
```

DEMOUNT

displays a volume disposition indicator in position 1 until the volume is demounted. Optionally, you can display the serial number of the volume to be demounted at the same time. The display flashes on and off. If a volume is not mounted on the device when the display request is executed, blanks are displayed.

The demount message can be displayed alternately (flashing) with a mount message for the next volume by specifying the MSER parameter.

UCB=(*reg*)—(2-12)

specifies a register containing the UCB address for the device. Use the address of a UCB, not a UCB copy.

DISP='D' or **'K'** or **'R'** or *addr*

specifies the character to display in position 1 of the pod, representing the volume disposition.

'D'

specifies demount a public volume. Apostrophes are required.

Note: “D” also displays when you specify an invalid character or when the volume use attribute is unknown (as in an automatic volume recognition (AVR) error when reading a label).

'K'

specifies keep a private volume and return it to the library. Apostrophes are required.

'R'

specifies retain a private volume near the device for further use. Apostrophes are required.

addr—RX-type address, A-type address, or (2-12)
specifies the address of an area containing a “D”, “K”, or “R”. For MF=L, you can only specify an A-type address.

MF=L or **(E,***addr***)**
specifies either the execute or list form of MSGDISP. If you do not specify this parameter, the standard form of the macro is used.

L specifies the list form of MSGDISP. This generates a parameter list that can be used as input to the execute form. The execute form can modify the parameter list.

(E,*addr***)**
specifies that the execute form of the macro and an existing parameter list is to be used.

addr—RX-type address, (1), or (2-12)
specifies the address of the parameter list.

MLABEL='A' or **'N'** or **'S'** or **'X'** or *addr*
displays the label type of the volume to be loaded and made ready following a demount, in position 8. If you specify an unknown label type other than a blank, a “?” is displayed. You can only specify this parameter if you also specify the MSER parameter.

'A'
specifies ISO/ANSI (AL) or ISO/ANSI with user (AUL) labels. Apostrophes are required.

'N'
specifies no labels (NL), LTM (leading tape mark, created by VSE), or bypass label processing (BLP). Apostrophes are required.

'S'
specifies IBM Standard (SL) or IBM Standard with user (SUL) labels. Apostrophes are required.

'X'
specifies nonstandard (NSL) labels. Apostrophes are required.

addr—RX-type address, A-type address, or (2-12)
specifies the address of an area containing an “A”, “N”, “S”, or “X” (see the following explanations of these characters). For MF=L, you can only specify an A-type address.

MSER='*volser-to-mount***'** or *addr*
displays the mount message for the next volume alternately (flashing) with the demount message. The display continues until you demount the current volume. At that time, the mount message will display (flashing) until you load the volume and make the device ready. If no volume is mounted at the time the demount and mount messages are executed, only the mount message will display (flashing) until the volume is loaded and ready.

'*volser-to-mount***'**
specifies the volume serial number of the volume to be mounted, as a literal. Apostrophes are required.

addr—RX-type address, A-type address, or (2-12)
specifies the address of the volume serial number of the volume to be mounted. For MF=L, you can only specify an A-type address.

SER='*volser***'** or *addr*
specifies the serial number of the volume to be demounted. The serial number is displayed in positions 2 through 7. If you do not specify SER, the system supplies the volume serial number. If the serial number is not available, a scratch volume is used, unless the volume use attribute indicates a default of "PRIVAT".

'*volser***'**
specifies the volume serial number as a literal. Specify with apostrophes.

addr—RX-type address, A-type address, or (2-12)
specifies the address of the volume serial number. This parameter is not valid for the MF=L form. For MF=L, you can only specify an A-type address.

TEST=NO or **YES**
specifies whether the macro expansion is to include code that tests the UCB to determine whether message display is supported. If the result of the test is that the message display is not supported, an SVC is not invoked.

NO
specifies that the macro expansion is not to include code that tests the UCB to determine whether the device supports message display.

YES
specifies testing the UCB by the MSGDISP macro before attempting to invoke the message display service routine.

Note: TEST=YES requires you to include the UCB mapping macro (IEFUCBOB) in the source code.

Note: If this provision isn't followed, a program check in expansion code might result. Programs running in AMODE 24 and invoking the MSGDISP macro with the TEST=YES parameter cannot pass the actual address of a UCB that resides above the 16MB line. These programs must pass the captured UCB address or, if an actual address is passed, the UCB must reside below the 16MB line.

WAIT=NO or **YES**
specifies when control is to be returned to you.

NO
specifies that the MSGDISP function is not to wait for completion of I/O initiated on the caller's behalf. When MSGDISP returns, the I/O request might still be running. I/O return codes are not returned, and I/O errors are recorded in the same manner as any permanent error by the error recovery procedure.

YES
specifies that control is to be returned after I/O is complete.

MSGDISP—Resetting the Message Display

The format of the MSGDISP macro with the RESET parameter is:

```
►►──┬──────────┬──MSGDISP──RESET──┬──,UCB=(reg)──┬──────────────────────────────►
    └─label────┘                  └──,UCBL=addr──┘

►──┬──,MF=──┬──L──────────┬──┬──────────────────┬──┬──────────────────┬──►◄
   │        └──(E,addr)───┘  │  ┌──NO──┐         │  │  ┌──YES──┐       │
   │                         └──,TEST=──┼──YES──┘  └──,WAIT=──┼──NO──┘
```

RESET
clears all existing data on the display. If you specify WAIT=NO and the last service requested was a demount, the display is not cleared.

After being cleared, the display shows the device's *internal* status message (for example, a message indicating that the device is ready).

UCB=(*reg***)**—(2-12)
specifies a register containing the UCB address for the device. Use the address of a UCB, not a UCB copy.

UCBL=*addr*—RX-type address, A-type address, (0), or (2-12)
specifies the address of a list containing a maximum of 64 words. Each word in the list contains the address of a UCB representing a device whose display is to be reset. The end of the list is indicated by a `'1'` in the high-order bit of the last address in the list. If an error is encountered while processing the list, register 1 points to the associated UCB when you regain control.

You cannot specify UCBL with TEST=YES and WAIT=NO.

MF=L or **(E,***addr***)**
specifies either the execute or the list form of MSGDISP. If you do not specify this parameter, the standard form of the macro is used.

L specifies the list form of MSGDISP. This generates a parameter list that can be used as input to the execute form. The execute form can modify the parameter list.

(E,*addr***)**
specifies that the execute form of the macro and an existing parameter list is to be used.

addr—RX-type address, (1), or (2-12)
specifies the address of the parameter list.

TEST=NO or **YES**
specifies whether the macro expansion is to include code that tests the UCB to determine whether message display is supported. If the result of the test is that the message display is not supported, an SVC is not invoked.

NO
specifies that the macro expansion is not to include code that tests the UCB to determine whether the device supports message display. NO is the default.

YES
specifies testing the UCB by the MSGDISP macro before attempting to invoke the message display service routine. You cannot specify TEST=YES if you also specify the UCBL parameter.

Note: TEST=YES requires you to include the UCB mapping macro (IEFUCBOB) in the source code.

Note: If this provision isn't followed, a program check in expansion code might result. Programs running in AMODE 24 and invoking the MSGDISP macro with the TEST=YES parameter cannot pass the actual address of a UCB that resides above the 16MB line. These programs must pass the captured UCB address or, if an actual address is passed, the UCB must reside below the 16MB line.

WAIT=NO or **YES**
specifies when control is to be returned to you.

NO
specifies that the MSGDISP function is not to wait for completion of I/O initiated on the caller's behalf. When MSGDISP returns, the I/O request might still be running. I/O return codes are not returned, and I/O errors are recorded in the same manner as any permanent error by the error recovery procedure.

You cannot specify WAIT=NO if you also specify the UCBL parameter.

YES
specifies that control is to be returned after I/O is complete.

Note: Demount messages can be reset only if WAIT=YES is specified.

MSGDISP—Providing the Full Range of Display Options

The format of the MSGDISP macro with the GEN parameter is:

```
►►──┬──────────┬──MSGDISP──GEN──,UCB=(reg)──┬──────────────────────────────────────────►
    └─label────┘                            │                                          │
                                            └─,FLASH=─┬─BLINK──────┬──┘
                                                      ├─STEADY─────┤
                                                      ├─STEADY2────┤
                                                      ├─BLINK2─────┤
                                                      └─ALT────────┘

►──┬──────────────────┬──┬──────────────┬──┬──────────────────────────────────────────►
   │                  │  │              │  │                                          │
   └─,MF=─┬─L─────────┬┘  └─,TEST=─┬─NO──┬┘
          └─E,addr────┘            └─YES─┘

►──┬──────────────────────────┬──┬──────────────────────────────────────────────────►
   │                          │  │                                                  │
   └─,TXT=─┬─'msgtxt'─────────┬┘  └─,TXT2=─┬─'altmsgtxt'──────────┬┘
           └─addr──────────────┘            └─addr─────────────────┘

►──┬──────────────────────────┬──┬──────────────────────────────────────────────────►◄
   │                          │  │                                                  │
   └─,VOL=─┬─STATIC───────────┬┘  └─,WAIT=─┬─YES─┬┘
           ├─REMOVE───────────┤            └─NO──┘
           ├─INSERT───────────┤
           └─SWAP─────────────┘
```

GEN
specifies the full range of display options.

UCB=(*reg*)—(2-12)
specifies a register containing the UCB address for the device. Use the address of a UCB, not a UCB copy.

FLASH=STEADY or **STEADY2** or **BLINK** or **BLINK2** or **ALT**
specifies message display mode.

Note: If you specify VOL=SWAP, messages will always be displayed as if you had specified FLASH=ALT.

STEADY
specifies that the primary message (TXT) is to be displayed without flashing.

STEADY2
specifies that the alternate message (TXT2) is to be displayed without flashing.

BLINK
specifies that the primary message (TXT) flash on and off at a rate of approximately two seconds on and one-half second off.

BLINK2
specifies that the alternate message (TXT2) flash on and off at a rate of approximately two seconds on and one-half second off.

ALT
specifies that the primary and alternate messages (TXT and TXT2) flash on and off alternately, at a rate of approximately two seconds on and one-half second off.

MF=L or **(E,*addr*)**
specifies either the execute or the list form of MSGDISP. If you do not specify this parameter, the standard form of the macro is used.

L specifies the list form of MSGDISP. This generates a parameter list that can be used as input to the execute form. The execute form can modify the parameter list.

(E,*addr*)
specifies that the execute form of the macro and an existing parameter list is to be used.

addr
specifies the address of the parameter list. Specify either an RX-type address or a register in the range of 2 through 12.

TEST=NO or **YES**
specifies whether to test the UCB to determine if the device is capable of displaying messages.

NO
specifies that the macro expansion is not to include code that tests the UCB to determine whether the device supports message display.

YES
specifies testing the UCB by the MSGDISP macro before attempting to invoke the message display service routine.

Note: TEST=YES requires you to include the UCB mapping macro (IEFUCBOB) in the source code.

Note: If this provision isn't followed, a program check in expansion code might result. Programs running in AMODE 24 and invoking the MSGDISP macro with the TEST=YES parameter cannot pass the actual address of a UCB that resides above the 16MB line. These programs must pass the captured UCB address or, if an actual address is passed, the UCB must reside below the 16MB line.

TXT='*msgtxt***'** or *addr*
specifies 8 characters to be shown in positions 1 through 8 of the display. If you do not specify TXT, blanks are displayed.

'*msgtxt***'**
specifies the 8 characters as literals. Apostrophes are required.

addr—RX-type address, A-type address, or (2-12)
specifies the address of an area containing the 8 characters. For MF=L, you can only specify an A-type address.

TXT2='*altmsgtxt***'** or *addr*
specifies 8 alternate characters to display in positions 1 through 8 of the display. If you do not specify TXT2, blanks are displayed.

'*altmsgtxt***'**
specifies the 8 characters as literals. Apostrophes are required.

addr—RX-type address, A-type address, or (2-12)
specifies the address of an area containing the 8 characters. For MF=L, you can only specify an A-type address.

VOL=STATIC or **REMOVE** or **INSERT** or **SWAP**
specifies message display mode, based on volume status.

STATIC
specifies that messages display without regard to volume status until the next message request is executed, or until the next command initiates volume movement.

REMOVE
specifies that messages display until the current volume is demounted. This parameter is ignored if a volume is not mounted when the request is executed.

INSERT
specifies that messages display until a volume is present, the tape threaded, and the active/inactive switch is in the active position. This parameter is ignored if a volume is loaded and ready when the request is executed.

SWAP
specifies that messages always display as if FLASH=ALT were specified. The data from TXT and TXT2 displays alternately (flashing) until the current volume has been demounted. Then only TXT2 displays (flashing) until a new volume is loaded and ready. If no volume is mounted when this parameter is specified, only TXT2 data displays (flashing) until a new volume is loaded and ready.

WAIT=NO or **YES**
specifies when control is to be returned to you.

NO
specifies that the MSGDISP function is not to wait for completion of I/O initiated on the caller's behalf. When MSGDISP returns, the I/O request might still be running. I/O return codes are not returned, and I/O errors are recorded in the same manner as any permanent error by the error recovery procedure.

YES
specifies that control is to be returned after I/O is complete. This is the default.

Return Codes from MSGDISP

When the system returns control to the problem program, the low-order byte of register 15 contains a return code. The low-order byte of register 0 can contain a reason code. These codes are described in the following table:

Return Code	Reason Code	Meaning
0 (X'00')		Successful completion.
4 (X'04')		Device does not support MSGDISP.
8 (X'08')	1 (X'01')	Invalid input parameter.
	2 (X'02')	Invalid DCB or DEBCHK error.
	3 (X'03'	Environmental error.
	4 (X'04')	Authorization (TESTAUTH) violation.
	5 (X'05')	Invalid UCB. Requires the address of a UCB, not a UCB copy.
	6 (X'06')	Invalid request.
	11 (X'0B')	Unsuccessful ESTAE macro call.
	12 (X'0C')	Unsuccessful GETMAIN request.
12 (X'0C')		I/O error (The system posted the request for an error).

Note: An I/O error occurs for load display if the drive display has a hardware failure.

If you get return code X'04' or X'0C' on a RESET UCBL operation, when you regain control, register 1 points to the UCB associated with the error.

Chapter 9. Using DFSMSdfp Callable Services

This chapter describes the DFSMSdfp-related callable services. Callable services reside in SYS1.CSSLIB, the callable system service library. They are invoked from a user program by issuing a CALL statement, accompanied by a parameter identifying the desired service and a list of service-specific arguments and storage areas. The intent is that if you link edit any of the routines described in this chapter with your program, the routine continues to execute correctly on future levels of the operating system.

The DFSMSdfp callable services support 14 callable system services. Five of these, the IGWASYS, IGWASMS, IGWABWO, IGWLSHR, and IGWARLS, can be invoked by the high-level languages supported by Systems Application Architecture and by assembler language callers. The CDRA routines can be called from any language that supports LE/370.

Note: IGWAQSMS is superseded by these services and is no longer available. The other nine functions are related to using the character data representation architecture (CDRA) identifiers. They are application program interfaces (APIs) that are needed to consistently and correctly process graphic character data. We have listed them here (see “Character Data Representation Architecture (CDRA) APIs” on page 226), but for detailed information on their use, refer to *Character Data Representation Architecture Reference and Registry*.

The services enable programs written in assembler language or high-level languages to use:

- IGWASYS to determine the version, release, and modification level of DFSMSdfp on your system, and the status of the SMS subsystem.
- IGWASMS to return certain data set attributes. These data set attributes are:
  - SMSDATA—SMS class names, that is, storage class, management class, and data class.
  - DSTYPE—Currently indicates whether the data set is a PDSE-type data set, HFS, or neither.
- IGWABWO to communicate with DFSMSdfp to retrieve or set various data set–related indicators. Through the use of these indicators, your program can determine if a data set is eligible for backup while it is open for update and, if eligible, what action can or should be taken. See Figure 77 on page 237 for an explanation of the indicators.
- IGWLSHR to determine the DFSMSdfp share attributes in use on the current system.
- IGWARLS to query the information in the catalog for record-level sharing (RLS), including the values of the LOG, BWO, and LOGSTREAMID parameters, the VSAM_QUIESCED indicator, the RLS_RECOVERY_TIMESTAMP fields, and whether the sphere requires forward recovery.

This section describes calling the service from a nonreentrant program written in assembler language. See *OS/390 MVS Assembler Services Reference* for information on using CALL in a reentrant program. For information on using CALL in programs written in high-level languages, see the applicable language documentation.

© Copyright IBM Corp. 1979, 1999

Your program can call the DFSMSdfp callable services in either 24- or 31-bit AMODE. The program can be executed in any protection key and in either supervisor or problem state. When you invoke any of the callable services, your program must provide the address of a standard 18 word save area in register 13. The syntax diagrams here show a CALL statement in assembler language. These callable services can be invoked in two ways:

- A CALL statement is coded in the invoking application. The callable services IGWASYS, IGWASMS, IGWABWO, IGWLSHR or IGWARLS, among others, are in SYS1.CSSLIB. When link-editing the invoking application, specify SYS1.CSSLIB in the library concatenation.
- The invoking application can issue a LINK or LOAD/CALL to the desired service, IGWASYS, IGWASMS, IGWABWO, IGWLSHR or IGWARLS. This technique ensures that your program always gets the latest copy of the routine but it requires either that SYS1.CSSLIB be in the link list concatenation or that it be available using JOBLIB, STEPLIB, or the task library. Your system programmer can tell you whether SYS1.CSSLIB is in the link list concatenation. For an example of using a LINK macro, see page 231. For an example of using a CALL macro, see pages 228 and 233. Page 233 shows an example of a LOAD and CALL.

Character Data Representation Architecture (CDRA) APIs

The following CDRA APIs are included in the DFSMS/MVS product library. For more detailed description of both the APIs and their use, see *Character Data Representation Architecture Reference and Registry* (*SC09-2190*).

API	Description
CDRGESP	Get Encoding Scheme, Character Set, and Code Page Elements
CDRSCSP	Get Short Form (CCSID) from Specified ES (CS, CP)
CDRGESE	Get Encoding Scheme Element and its Subelements
CDRGCTL	Get Control Function Definition
CDRSMXC	Get Short Form (CCSID) with maximal CS for Specified ES, CP
CDRMSCI	Multiple-Step Convert Initialize
CDRMSCP	Multiple-Step Convert Perform
CDRMSCC	Multiple-Step Convert Clean Up
CDRXSRF	Extract Status and Reason Codes from Feedback Code

Call for DFSMS/MVS Level Determination

The DFSMS/MVS level determination call (IGWASYS) returns the DFSMS/MVS level on your system and the status of the SMS subsystem.

An alternative technique to determine the level of the system is described in “Determining Version, Release, and Modification Level of DFSMS/MVS” on page 207.

Format

The format of the system attribute IGWASYS call statement is:

```
►►──┬──────────┬──CALL──IGWASYS──,──return_code──,──(──reason_code──,──level_indicator──,──►
    └──label───┘
►──system_level──,──system_attr──)──────────────────────────────────────────────────────►◄
```

Parameters

return_code
Return code from IGWASYS. The return code is also returned in register 15. For an explanation, see “IGWASYS, IGWASMS, IGWABWO, IGWLSHR Return and Reason Codes” on page 239. This is an output argument that must be defined as an integer.

reason_code
Reason code from the IGWASYS service. The reason code is also returned in register 0. For an explanation, see “IGWASYS, IGWASMS, IGWABWO, IGWLSHR Return and Reason Codes” on page 239. This is an output argument that must be defined as an integer.

level_indicator
The product whose level information is requested. This is an input argument. Define level_indicator as an integer. Code a value of 1 to request the level of MVS/XA DFP Version 2 or MVS/DFP Version 3 or code a 2 to request the level of DFSMS/MVS. This value affects what the system returns in system_level, which is described below.

system_level
The product level that is installed on the system invoking the service. This is an output argument and is a four-element array of integers. The array elements consist of the following:

- Version number
- Release number
- Modification level
- Special indicator

If you pass a value of 1 for *level_indicator*, then DFSMS/MVS returns the *system_level* values as 3, 3, 2 and 1. The first three values reflect the fact that DFSMS/MVS contains the functions of MVS/DFP Version 3 Release 3 modification level 2. The system sets the fourth value to 1 to indicate that the system is actually at a level higher than 3.3.2. If your program were executing on MVS/DFP 3.3.2, then these four values would have been returned as 3, 3, 2 and 0.

If you pass a value of 2 for *level_indicator*, then DFSMS/MVS sets the first *system_level* word to 1 (to indicate Version 1) and sets the second and third words to the release and modification level values that are appropriate for your level of DFSMS/MVS. Currently DFSMS/MVS sets the fourth value to 1 to indicate the name “DFSMS/MVS.”

The intent is that in any future level of the system, your program can determine whether it is running on a particular level or it is running on some later level. If that is what you wish your program to do, then it is suggested that your program test all four words of system level. First test the fourth word to ensure that it has a value of 1 or greater. Then test the other three words to see

whether they designate the appropriate version, release and modification level. It is IBM's intent that if one of the values in the first column of the figure below is smaller than was returned for the corresponding value in the prior release, then one of the values in the second column contains a larger value than was returned in the prior release:

```
Third word (modification level)   Fourth, first or second word
Second word (release)             Fourth or first word
First word (version)              Fourth word
```

system_attr

System attributes are returned in the *system_attr* array. This is an output argument. The array elements are defined as follows:

1 Status of the Storage Management Subsystem. A value of 0 indicates inactive; 1 indicates active.

2–4 Reserved elements; 0 is returned.

Define as a four-element array of integers.

Return Codes

See Figure 78 on page 239 for the IGWASYS return and reason codes.

Example

The following example shows a system attribute call using a CALL statement.

```
            .
            .
         CALL  IGWASYS,(RC1,RS1,CODE1,LEVEL,ATTR)
         LTR   R15,R15        Test return code
         BNZ   BADSYS
         CLC   LEVEL,=F'2'      Test for MVS/XA DFP Version 2
         BE    OLDSYS
         MVC   SYSNAME,UNKNAME  Assume name is unknown
         BL    BADSYS           Branch if unknown system
         MVC   SYSNAME,=CL10'MVS/DFP'   Show we are on MVS/DFP
         CLC   LEVEL+12(4),=F'1'        See if after MVS/DFP
         BL    OLDSYS           Branch if before DFSMS/MVS
         CALL  IGWASYS,(RC1,RS1,CODE2,LEVEL,ATTR)
         LTR   R15,R15          Branch if environment or
         BNZ   BADSYS             system error
         MVC   SYSNAME,UNKNAME      Assume unknown product name
         CLC   LEVEL+12(4),=F'1'    Test for DFSMS/MVS code
         BL    OLDSYS           Branch if unexpected code
         BH    NEWENUF          Branch if newer than DFSMS/MVS
         MVC   SYSNAME,=CL10'DFSMS/MVS'  Set product name for code 1
         CLC   LEVEL(12),=F'1,3,0'  Branch if earlier than
         BL    OLDSYS                 DFSMS/MVS Version 1, Rel. 3.0
NEWENUF  EQU   *
          .
          .
RC1      DC    F'0'    Return code
RS1      DC    F'0'    Reason code
CODE1    DC    F'1'    Ask for pre-DFSMS/MVS DFP level
CODE2    DC    F'2'    Ask for level of DFSMS/MVS or later
LEVEL    DC    4F'0'   Version, release, modification level and code
ATTR     DC    4F'0'   SMS attributes
SYSNAME  DC    CL10'MVS/XA DFP'  Name of product
UNKNAME  DC    (L'SYSNAME)C'?'   Constant for unknown name
```

Figure 74. Example of an IGWASYS Call Statement

Call for Data Set Attribute Retrieval

The data set attribute retrieval call (IGWASMS), returns the names of the data set's related SMS classes and whether it is a PDSE, or an HFS data set, or neither.

Format

The format of the data set attribute IGWASMS call statement is:

```
►►──┬──────────┬──CALL──IGWASMS──,──(──return_code──,──reason_code──,──prob_det──,──────►
    └──label───┘
►──dsname_length──,──dsname──,──sms_data──,──ds_type──)──────────────────────────────►◄
```

Parameters

return_code
Return code from IGWASMS. The return code is also returned in register 15. Return codes are explained in “IGWASYS, IGWASMS, IGWABWO, IGWLSHR Return and Reason Codes” on page 239. This is an output argument that must be defined as an integer.

reason_code
Reason code from IGWASMS. The reason code is also returned in register 0. Reason codes are explained in “IGWASYS, IGWASMS, IGWABWO, IGWLSHR Return and Reason Codes” on page 239. This is an output argument that must be defined as an integer.

prob_det
Problem determination data. See “IGWASYS, IGWASMS, IGWABWO, IGWLSHR Return and Reason Codes” on page 239 for more information about problem determination data. This is an output argument that must be defined as a two-element array of integers.

dsname_length
Length, in bytes, of the data set name provided by the caller in *dsname*. The value can be a number from 1 to 44. This is a required input argument that must be defined as an integer.

dsname
Name of the data set that the IGWASMS service operates on. For VSAM data sets, the cluster name must be specified. This is a required input argument that must be defined as EBCDIC character data of length *dsname_length*.

sms_data
The SMS class names associated with the specified data set returned, left-justified with blanks padded on the right. The array elements are returned as follows:

- Storage class name, or blanks if the data set is not an SMS data set.
- Management class name, or blanks if the data set has no associated management class.
- Data class name, or blanks if the data set has no associated data class.

This is an output argument that must be defined as a three-element array, where each entry is a 30 (byte) character EBCDIC string.

ds_type
The type of data set, *dsname*, is returned. A value of 1 indicates the data set is a PDSE-type data set. A value of 2 indicates the data set is an HFS-type data set. An HFS data set defines a file system and is not a file within the file system. A value of 0 indicates that it is neither. No other values are currently defined. This is an output argument that must be defined as an integer.

Return Codes

See Figure 78 on page 239 for the IGWASMS return and reason codes.

Example

The following example shows sample coding for a data set attribute call using a LINK statement.

```
            .
            .
          LINK EP=IGWASMS,MF=(E,ASMSLIST)
            .
            .

RC2       DC   F'0'
RS2       DC   F'0'
PROB1     DC   2F'0'
DSNLEN1   DC   A(L'DSN1)
DSN1      DC   CL12'THIS.DATASET'
SMSDATA   DC   3CL30' '
DSTYPE    DC   F'0'
ASMSLIST  DC   A(RC2)
          DC   A(RS2)
          DC   A(PROB1)
          DC   A(DSNLEN1)
          DC   A(DSN1)
          DC   A(SMSDATA)
          DC   A(DSTYPE)
```

Figure 75. Example of an IGWASMS Call LINK Statement

Call for Data Set Backup-While-Open Support

The data set backup-while-open support call (IGWABWO) communicates with DFSMSdfp to retrieve or set indicators related to taking data set backups while they are open for update.

Format

The format of the IGWABWO callable service is:

```
►►──┬──────────┬──CALL──IGWABWO──,──(──return_code──,──reason_code──,──prob_det──,──────►
    └──label───┘
►──read_write──,──dsname_length──,──dsname──,──select──,──bwo_flags──,──bwo_recov──,──────►
►──bwo_resrv──,──)──────────────────────────────────────────────────────────────────────►◄
```

Parameters

return_code
Return code from IGWABWO. The return code is also returned in register 15. Return codes are explained in “IGWASYS, IGWASMS, IGWABWO, IGWLSHR Return and Reason Codes” on page 239. This is an output argument that must be defined as an integer.

reason_code
Reason code from IGWABWO. The reason code is also returned in register 0. Reason codes are explained in “IGWASYS, IGWASMS, IGWABWO, IGWLSHR Return and Reason Codes” on page 239. This is an output argument that must be defined as an integer.

prob_det
Problem determination data. See “IGWASYS, IGWASMS, IGWABWO, IGWLSHR Return and Reason Codes” on page 239 for more information about problem determination data. This is an output argument that must be defined as a two-element array of integers.

read_write
Function indicator for this service. A caller-supplied value of 0 indicates that this is a READ-type request for the backup-while-open (BWO) data of the specified data set. A value of 1 indicates a WRITE-type request, that is, an initialization or update of the specified data set's BWO data with the supplied arguments *bwo_flags, bwo_recov,* and *bwo_resrv*. The select argument indicates which arguments are to be processed. This is a required input argument that must be defined as an integer.

dsname_length
Length, in bytes, of the data set name provided by the caller in *dsname*. The value can be a number from 1 to 44. This is a required input argument that must be defined as an integer.

dsname
Name of the data set that the IGWABWO service operates on. Only system-managed VSAM-type data sets are eligible to be backed up while they are open for update. The *dsname* specified must be the base cluster name of a VSAM data set. This is a required input argument that must be defined as EBCDIC character data of length *dsname_length*.

select
Indicates which of the following arguments will be processed. Arguments are specified by selecting the appropriate value. This is a required input argument.

1 to process *bwo_flags*
2 to process *bwo_recov*
3 to process *bwo_flags* and *bwo_recov*

Note: Regardless of how many arguments are to be processed, all three fields (*bwo_flags, bwo_recov*, and the reserved *bwo_resrv* field) must be defined in your program and included in the invocation. Those not selected will receive no value in a READ-type request. The values of those not selected will be ignored in a WRITE-type request.

bwo_flags
This argument is a three-element array, whose elements correspond to the three BWO flags associated with an SMS data set. *bwo_flags* is an output

argument for *read_write=0* type requests, and a required input argument for *read_write=1* type requests.

1. The first element is associated with flag, BWO1. 1 is on, 0 is off.
2. This element corresponds to flag, BWO2. 1 is on, 0 is off.
3. This element corresponds to flag, BWO3. 1 is on, 0 is off.

Define as a three-element array of integers.

bwo_recov
This argument is an 8-byte storage area containing the recovery timestamp associated with a data set that is eligible for BWO. *bwo_recov* is an output argument for read_write=0 type requests, and a required input argument for read_write=1 type requests. The format of the timestamp for CICS VSAM data sets is as follows:

- The first word contains the date in packed decimal format, 0CYYDDDF, where:

  0C is the century - 00 represents 19YY, 01 represents 20YY
  YY is the last two digits of the year
  DDD is the day of the year (Julian date)
  F is the sign (F for positive number)

- The second word contains the time in packed decimal format, HHMMSSTF, where:

  HH is hours, based on a 24-hour clock
  MM is minutes
  SS is seconds
  T is tenths of a second
  F is the sign (F for positive number)

bwo_resrv
This argument is reserved for future use. While the *bwo_resrv* argument cannot be written or read, it must be defined in your program and included in the invocation. Define as EBCDIC character data of length 16 bytes.

Return Codes

See Figure 78 on page 239 for the IGWABWO return and reason codes.

Example

The following example shows sample coding for WRITE and READ-type backup-while-open calls using LOAD and CALL statements.

```
                .
                .
          LOAD EP=IGWABWO
          LR   R2,R0
          CALL (R2),(RC1,RS1,PROB1,RW1,DSNLEN1,DSN1,SEL1,BWOF1,BWOR1,BWRE)
          CALL (R2),(RC2,RS2,PROB2,RW2,DSNLEN2,DSN2,SEL2,BWOF2,BWOR2,BWRE)
                .
                .

*  ARGUMENTS FOR WRITING
RC1       DC   F'0'
RS1       DC   F'0'
PROB1     DC   2F'0'
RW1       DC   F'1'            WRITE
DSNLEN1   DC   F'11'
DSN1      DC   CL44'THAT.VSAM01'
SEL1      DC   F'3'            WRITE BWO_FLAGS AND BWO_RECOV
BWOF1     DC   F'0'
          DC   F'1'
          DC   F'0'
BWOR1     DS   0F
          DC   X'0096137F'     INPUT DATE IN 0CYYDDDF FORMAT
          DC   X'1045301F'     INPUT TIME IN HHMMSSTF FORMAT
BWRE1     DC   CL16' '
*  ARGUMENTS FOR READING
RC2       DC   F'0'
RS2       DC   F'0'
PROB2     DC   2F'0'
RW2       DC   F'0'            READ
DSNLEN2   DC   F'11'
DSN2      DC   CL44'THAT.VSAM01'
SEL2      DC   F'3'            READ BWO_FLAGS AND BWO_RECOV
BWOF2     DC   3F'0'
BWOR2     DC   CL8' '
BWRE2     DC   CL16' '
*
```

Figure 76. Example of IGWABWO Using LOAD and CALL Statements

Using the Backup-While-Open Facility

The following section describes the usage of the BWO facility. BWO flags and the BWO recovery field can be retrieved or updated using the IGWABWO service described in “Call for Data Set Backup-While-Open Support” on page 231. The BWO indicators are described in Figure 77 on page 237.

For environments that require high-availability, it might not be possible or desirable to stop or quiesce an application to produce consistent backup copies of the application's data sets.

For these environments DFSMSdfp provides support to allow SMS-managed VSAM data sets that are open for output to be backed up. The support is only useful for applications (such as database systems) that can recover a restored database to a point of consistency. This is typically done from a log (forward recovery log) maintained by the application that contains record images of all changed (added, deleted, or updated) records. These images can then be reapplied to a backup

copy of the database, logically recreating the status of the database at a particular point in time.

The support provided by BWO might not be necessary for online applications that can quiesce the database data sets to ensure no output or update activity against the data set while the backup is in progress. Quiescing a data set in this context means the data set is closed and unallocated.

The following discussion of the operation of this support uses these terms:

- Database manager—the application that controls access to the data sets to be processed. In order for BWO support to be effective, the database manager must have some logging facility to allow point-in-time reconstruction of a database.
- Backup manager—the applications or products that perform the backup and restore functions, such as DFSMShsm and DFSMSdss.
- Recovery manager—the component that manages the inventory of recovery logs and applies the changes from the appropriate log(s) to the restored data set.

The following paragraphs describe the relationships between the BWO support and a user of BWO. Refer to Figure 77 on page 237 for the various states of the BWO flags in the following discussion. The BWO indicators are retained in the catalog. The BWO flag states are set or reset using the DFSMSdfp callable system service IGWABWO.

- At initial allocation (IDCAMS DEFINE, IDCAMS or TSO ALLOCATE, JCL and dynamic allocation), the data set is not enabled for BWO (default, BWO flag state is 000).
- The database manager should check the BWO flags prior to opening the data set to ensure it is not downlevel (BWO flag state is 101 or 001). If the data set is downlevel, the recovery manager must be used to apply log changes to the data set.
- The database manager must set flag BWO1 (BWO flag state 100) on for each data set that is allowed to be backed up while open for output. This authorizes the backup manager to initiate backups without serializing the data set, whether or not it is being accessed by the application.
- The backup manager must retrieve the BWO flags prior to the start of the backup. If BWO1 is on, then a backup can be taken without any serialization; otherwise, normal data set serialization must be performed by the backup manager.

  When the backup completes, the backup manager must retrieve the BWO flags again. If the BWO flag state has changed, then at some point during the backup an action occurred that prevented creation of a valid backup. The backup manager should discard the backup just created.
- When the data set needs to be recovered, it is first restored using the backup manager. Data sets are serialized during restore to prevent applications from accessing them. The backup manager must set the BWO flags at the completion of the restore to indicate whether the restore was done using a backup copy that was created with or without serialization.

  - If the backup was taken without serialization, the BWO flags must be set to 101.
  - If the backup was taken with serialization, the BWO flags must be set to 000.

  In either case, the application administrator should decide whether or not to apply recovery logs.

  The database manager should not allow access to the data set until the recovery manager has completed processing.

- The recovery manager should change the BWO flags to 001 before opening the data set, apply the logs, and then set the flags to 000 to indicate that the data set has been processed and is consistent. The database manager can then reestablish normal access to the data set (i.e., open the data set for use).
- For a data set that is enabled for BWO, in certain instances the system prevents starting a backup copy without serialization (for example, during a CI/CA split) by setting the BWO flags to 010. This indicates a backup should not be started without serialization (BWO1 off), and that a backup that is currently in process should be considered invalid. When the condition that prevented the starting of a backup is ended, the system resets the BWO flags to 110. This indicates that a backup can now be started without serialization, and that any backup in progress should be discarded.
- If the database data sets are accessed by batch programs (when the database manager is not accessing the data sets) that do not create forward recovery logs, the database manager should clear the BWO1 flag and set the BWO3 flag at close (i.e., OX1). The setting of the BWO2 flag should not be changed. If the backup manager discovers this BWO state at the end of backup without serialization, the backup is not valid and should be discarded. The backup manager can start a backup with serialization if the BWO flag state is 011, but the flags should be reset to 000.

Notes:

1. Since backups can result in heavy I/O activity, you might want to take backups during the time of least activity against the data set to avoid affecting the application response time.
2. The BWO flags are not locked between reading and setting them with IGWABWO. The application is responsible for providing serialization when the settings of the flags are changed.
3. KSDS data sets require special consideration. Inserts and updates can result in control interval (CI) and control area (CA) splits. Backups taken without serialization during CI and CA splits are discarded by the IBM products to prevent missing or duplicate records in the backup copies.

   The frequency of CI and CA splits depends on the insert activity, the update activity that increases the lengths of records, and the amount of free space in CI and CA. For KSDS data sets that are BWO enabled, run backups during periods of low inserts and updates or ensure adequate free space in control intervals and control areas.
4. Backups without serialization can be taken (on data sets defined with share option 1 or 2) when:
   - The database contains alternate indices in the upgrade set.

- The database is accessed via pathnames, or with *ddname* or *dsname* sharing.

5. Data sets can be opened regardless of the setting of the BWO flags. It is the database manager's responsibility to determine whether the contents of the data set are consistent.
6. The database manager can use the BWO recovery field to store information (such as the log RBA or log timestamp) for the recovery manager to use in locating the appropriate recovery logs.
7. BWO concept only applies to logical data set backup/restore. It does not apply to physical data set backup/restore or full volume dump/restore.

Figure 77 describes the meaning of each of the BWO (BWO1, BWO2, & BWO3) indicators.

Figure 77. Backup-While-Open Indicators

BWO Setting	Description
000	Data set does not support backup without serialization.
001	Forward recovery in progress. Reset to 000 after recovery.
010	A CI/CA split for a BWO data set is in progress. Do not start backup. If backup in progress, discard at end. This state can exist at open if the database manager abended during split. Database action depends on database manager. The data set might need restore and forward recovery or backout of changes if AIXs are present.
011	The database manager closed a BWO data set and a CI/CA split had occurred when it was previously open. Backup manager should reset it to 000 and serialize to back up, not a BWO candidate. The database manager should change this state to 110 at open.
100	A BWO data set has been opened by the database manager. Back up without serialization.
101	Data set has been restored and requires forward recovery before it can be used. Reset it to 001 before forward recovery.
110	A CI/CA split has occurred and completed on a BWO data set. This state can exist at open if the database manager abended. Back up without serialization. Reset it to 100 before backup.
111	An invalid state.

Call for DFSMSdfp Share Attributes

The DFSMSdfp share attributes call (IGWLSHR) returns the DFSMSdfp share attributes currently in use.

Format

The format of the IGWLSHR callable service is:

```
►►──┬──────────┬──CALL──IGWLSHR──,──(──return_code──,──reason_code──,──prob_det──,──────────►
    └──label───┘
►──share_attr_selector──,──share_attr_array_length──,──share_attr_array──)──────────────►◄
```

Parameters

return_code
Return code from IGWLSHR. The return code is also returned in register 15. Return codes are explained in "IGWASYS, IGWASMS, IGWABWO, IGWLSHR Return and Reason Codes" on page 239. This is an output argument that must be defined as an integer.

reason_code
Reason code from IGWLSHR. The reason code is also returned in register 0. Reason codes are explained in "IGWASYS, IGWASMS, IGWABWO, IGWLSHR Return and Reason Codes" on page 239. This is an output argument that must be defined as an integer.

prob_det
Problem determination data. See "IGWASYS, IGWASMS, IGWABWO, IGWLSHR Return and Reason Codes" on page 239 for more information about problem determination data. This is an output argument that must be defined as a two-element array of integers.

share_attr_selector
Use to specify which DFSMSdfp share attributes are requested. Code a value of 1 to request the PDSE sharing protocol attributes. This is a required input argument that must be defined as an integer.

share_attr_array_length
Use to specify the size of the share attributes array. This length represents the number of array elements in the share_attr_array. Code the value required for the chosen *share_attr_selector* value. The *share_attr_array_length* must minimally be 1 when the share_attr_selector is 1. This is a required input argument that must be defined as an integer.

share_attr_array
Returns DFSMSdfp share attributes. This is an output argument. Define as an array of integers of length *share_attr_array_length*, where *share_attr_array_length* is the length required for the chosen *share_attr_selector* value. The array elements for the *share_attr_selector* value are returned as follows:

1 Status of PDSE sharing protocol. A value of 0 indicates PDSE support is unavailable; that is, the system supports the call, but SMS PDSE support is not active. If the system does not support this call (as with a previous release), then a return code of 36 is returned.

  A value of 1 indicates normal PDSE sharing protocol in use. A value of 2 indicates extended PDSE sharing protocol in use.

2–4 Reserved elements; 0 is returned.

Return Codes

See Figure 78 on page 239 for the IGWLSHR return and reason codes.

IGWASYS, IGWASMS, IGWABWO, IGWLSHR Return and Reason Codes

When IGWASYS, IGWASMS, IGWABWO, or IGWLSHR returns control to the calling program, it provides both a return code and a reason code. IGWASMS and IGWABWO can return additional data useful for problem determination in the *prob_det* array. IGWLSHR can return additional data regarding *share_attr_selector* and *share_attr_array_length* arguments. The following table identifies return code and reason code combinations, tells what each means, explains what and when additional problem determination data is returned, and recommends what action should be taken.

Figure 78 (Page 1 of 2). IGWASYS, IGWASMS, IGWABWO, IGWLSHR Return and Reason Codes

Return Code Dec (Hex)	Reason Code Dec (Hex)	Description
0 (0)	0 (0)	The operation was successful.
4 (4)	4 (4)	The operation was successful, but the *bwo_recov* argument has no valid value for the data set specified in *dsname*. This is because it was created under DFP 3.1.0, and no *bwo_recov* has been added to the data set. Add *bwo_recov* to the data set as appropriate.
8 (8)	4 (4)	An invalid *dsname_length* or *share_attr_selector* was specified. Correct the argument and retry the request.
8 (8)	8 (8)	An invalid *dsname* of blanks or invalid *share_attr_array_length* was specified. Correct the argument and retry the request.
8 (8)	12 (C)	An invalid *read_write* was specified. A value of 0 or 1 must be supplied. Correct the *read_write* argument and retry the request.
8 (8)	16 (10)	The values supplied for *bwo_flags* are not valid. BWO1, BWO2, and BWO3 must have a value of either 0 or 1. Correct the *bwo_flags* argument and retry the request.
8 (8)	20 (14)	BWO is only supported for VSAM-type data sets. The name specified was not a VSAM cluster name. Specify the name of a VSAM cluster in the *dsname* argument and retry the request.
8 (8)	24 (18)	An invalid *level_indicator* was specified. Correct the *level_indicator* argument and retry the request.
8 (8)	28 (1C)	An invalid *select* argument was specified. A value between 1 and 3 must be specified. Correct the *select* argument and retry the request.
8 (8)	32 (20)	The data set specified in *dsname* is not an SMS-managed data set. Correct the *dsname* argument and retry the request.
12 (C)	8 (8)	There is insufficient virtual storage to process the request. Free some virtual storage and retry the request. If the condition persists, contact IBM for programming assistance.
12 (C)	12 (C)	The data set specified in *dsname* could not be found. Verify that the data set exists and has been correctly specified in *dsname*.
16 (10)	4 (4)	An error occurred on a call to catalog management. The catalog return code is in the first element of *prob_det* and the catalog reason code is in the second element of *prob_det*. See message IDC3009I for an explanation of the catalog return code and reason code. **Note:** A catalog management return code of 8 indicates that the specified data set was not found. If you get this return code, correct *dsname* and retry the request.

Figure 78 (Page 2 of 2). IGWASYS, IGWASMS, IGWABWO, IGWLSHR Return and Reason Codes

Return Code Dec (Hex)	Reason Code Dec (Hex)	Description
20 (14)	4 (4)	A system error occurred during IGWASYS/SMS/BWO or IGWLSHR processing. The elements of *prob_det* contain additional diagnostic data. Contact IBM for programming assistance and provide them with the IGWASYS/SMS/BWO *return_code, reason_code,* and *prob_det* values.
36 (24)	4 (4)	Linkage cannot be established to the IGWLSHR service module or to IGWASYS/SMS/BWO service modules, IGWAMCS1 and IGWAMCS2. Either the wrong level of the operating system is being used, or the callable system service library, SYS1.CSSLIB, is missing the required services. Contact your installation system programmer for assistance.

Call for Record-Level Sharing Query

IGWARLS is used to query the information in the catalog for record-level sharing.

Format

The format of the IGWARLS call statement is:

```
►►──┬──────────┬──CALL──IGWARLS──,──(──return_code──,──reason_code──,──prob_det──,──────►
    └──label───┘
►──dsname_length──,──dsname──,──recovery_status──,──log_type──,──────────────────────►
►──logstreamid_length──,──logstreamid──,──rls_recovery_timestamp_utc──,──────────────►
►──rls_recovery_timestamp_local──,──vsam_quiesced──,──bwo──)──────────────────────►◄
```

Parameters

return_code
Return code from IGWARLS. The return code is also returned in register 15. Return codes are explained in “Return Codes” on page 242. This is an output argument. Define *return_code* as an integer.

reason_code
Reason code from IGWARLS. The reason code is also returned in register 0. Reason codes are explained in “Return Codes” on page 242. This is an output argument. Define *reason_code* as an integer.

prob_det
Problem determination data. See “Return Codes” on page 242 for more information about problem determination data. This is an output argument. Define *prob_det* as a two element array of integers.

dsname_length
Length, in bytes, of the data set name provided by the caller in *dsname*. The value can be a number from 1 to 44. This is a required input argument. Define *dsname_length* as an integer.

dsname
Name of the base cluster that the IGWARLS service will operate on. This is a required input argument. Define dsname as EBCDIC character data of length *dsname_length*.

recovery_status
Returns an indication as to whether the sphere is marked as requiring forward recovery.

- 0 - RLS recovery required is not pending for the VSAM sphere.
- 1 - RLS recovery required is pending.

recovery_status is an output argument that is defined as an integer.

log_type
The specification of the LOG= parameter on DEFINE CLUSTER is returned.

- 1 - LOG parameter undefined
- 2 - LOG=NONE
- 3 - LOG=UNDO
- 4 - LOG=ALL

log_type is an output argument that is defined as an integer.

logstreamid_length
Length, in bytes, of the LOGSTREAMID field. This is a required input parameter. Define logstreamid_length as an integer. The value of logstreamid_length should be at least 26 bytes.

logstreamid
The specification of the LOGSTREAMID on DEFINE CLUSTER is returned. If the parameter is undefined, blanks are returned. The caller can determine the size of the returned LOGSTREAMID field by scanning from right to left looking for a non blank character or until the entire field has been scanned. *logstreamid* is an output argument that is defined as an EBCDIC character data field of length logstreamid_length.

rls_recovery_timestamp_utc
An output argument which represents the UTC time (formally known as GMT) of the dump/copy in STCK format. The field is defined as an eight-byte unsigned integer. Your program might regard this field as either an integer or a character string.

rls_recovery_timestamp_local
An output argument which represents the local time of the dump/copy in STCK format. The field is defined as an eight-byte unsigned integer. Your program might regard this field as either an integer or a character string.

vsam_quiesced
An output argument which indicates whether the sphere is marked as VSAM_QUIESCED.

- 0 - The sphere is not marked VSAM_QUIESCED.
- 1 - The sphere is marked VSAM_QUIESCED.

vsam_quiesced is an output argument, defined as an integer.

bwo
An output argument which indicates whether the value of the BWO parameter on define cluster.

- 1 - BWO parameter is undefined.
- 2 - BWO = TYPECICS processing allowed.
- 3 - BWO = NO. BWO processing is not allowed.
- 4 - BWO = TYPEIMS processing allowed.
- 5 - BWO = TYPEOTHER processing allowed.

Return Codes

When IGWARLS returns control to the calling program, it provides both a return code and a reason code. IGWARLS can return additional data useful for problem determination in the *prob_det* array. Figure 79 identifies return code and reason code combinations, tells what each means, explains what and when additional problem determination data is returned, and recommends what action should be taken.

Figure 79 (Page 1 of 2). IGWARLS Return and Reason Codes

Return Code Dec (Hex)	Reason Code Dec (Hex)	Description
0 (0)	0 (0)	The operation was successful.
8 (8)	4 (4)	An invalid *dsname_length* was specified. Correct the *dsname_length* argument and retry the request.
8 (8)	8 (8)	An invalid *dsname* of blanks was specified. Correct the *dsname* argument and retry the request.
8 (8)	20 (14)	IGWARLS is only supported for VSAM data sets. The name specified was not the name of the base cluster. Specify the name of the base cluster in the dsname argument and retry the request.
8 (8)	32 (20)	The data set specified in *dsname* is not an SMS managed data set. Correct the *dsname* argument and retry the request.
8 (8)	40 (28)	For IGWARLS, the logstreamid_length specified was invalid (<=0) or was not large enough to return the requested logstreamid. Correct the *logstreamid_length* argument and retry the request.
12 (C)	8 (8)	There is insufficient virtual storage to process the request. Retry the request. If the condition persists, contact IBM for programming assistance.
12 (C)	12 (C)	The data set specified in *dsname* could not be found. Verify that the data set exists and has been correctly specified in *dsname*.
12 (C)	14 (E)	The data set specified in *dsname* was found in the catalog but its attributes were not available. Verify that the data set has been correctly specified in *dsname*.
12 (C)	16 (10)	Cannot access the data set that is specified in *dsname*. The data set has been HSM migrated. HRECALL the data set and retry the request.
16 (10)	4 (4)	An error occurred on a call to catalog management. The catalog return code is in the first element of *prob_det* and the catalog reason code is in the second element of *prob_det*. See message IDC3009I for an explanation of the catalog return code and reason code. **Note:** A catalog management return code of 8 indicates that the specified data set was not found. If you get this return code, correct *dsname* and retry the request.

Figure 79 (Page 2 of 2). IGWARLS Return and Reason Codes

Return Code Dec (Hex)	Reason Code Dec (Hex)	Description
20 (14)	4 (4)	A system error occurred during IGWARLS processing. The elements of *prob_det* contain additional diagnostic data. Contact IBM for programming assistance and provide them with the IGWARLS *return_code, reason_code,* and *prob_det* values.
36 (24)	4 (4)	Linkage cannot be established to the IGWRLS service module, IGWAMCS4. Either the wrong level of the operating system is being used, or the callable system service library, SYS1.CSSLIB, is missing the required services. Contact your installation system programmer for assistance.

Example

The following example shows the RLS query call using LOAD and CALL statements:

```
.
.

         LOAD EP=IGWARLS
         LR   R9,R0
         CALL (R9),(RC1,RS1,PROB1,DSNLEN1,DSN1,RECSTAT,LOGTYPE,      X
                   LOGSTRML,LOGSTRM,RCVTMG,RCVTML,VSAMQUIS,BWO)

RC1      DC   F'0'
RS1      DC   F'0'
PROB1    DC   2F'0'
DSNLEN1  DC   A(L'DSN1)
DSN1     DC   CL12'BASE.CLUSTER'
RECSTAT  DC   F'0'
LOGTYPE  DC   F'0'
LOGSTRML DC   A(L'LOGSTRM)
LOGSTRM  DC   CL26' '
RCVTMG   DC   XL8'00'
RCVTML   DC   XL8'00'
VSAMQUIS DC   F'0'
BWO      DC   F'0'
```

Figure 80. Example of the IGWARLS Query Call Using LOAD and CALL Statements

Chapter 10. Establishing Installation Defaults for Binder Options

The binder converts the output of language translators and compilers into an executable program unit that can either be read directly into virtual storage for execution or stored in a program library. For additional information on binding program modules and how to use them, see *DFSMS/MVS Program Management*, order number SC26-4916.

The system programmer can tailor binder options to the installation's needs by replacing a data-only CSECT in SYS1.LINKLIB. The system programmer can use CSECT IEWBODEF in load module IEWBLINK to specify default values for one or more options. The CSECT is initially set to the null string but can be modified to contain binder option default settings in the same form they would be specified in the JCL PARM field, without enclosing apostrophes or parentheses. The system programmer must set a halfword length field preceding the string to the current length of the string.

Once this CSECT has been assembled it must be linked into module IEWBLINK, using either the linkage editor or binder, replacing the null CSECT. IBM recommends using SMP/E to replace IEWBODEF instead of using the linkage editor or binder directly.

© Copyright IBM Corp. 1979, 1999

Chapter 11. Using the DESERV Exit

For more detail and explanation of the DESERV functions, see *DFSMS/MVS Using Data Sets* ; for their macros, see *DFSMS/MVS Macro Instructions for Data Sets*.

With the introduction of the interface DESERV (GET, PUT, DELETE, RENAME and UPDATE) as a functional replacement for BLDL and STOW, two facilities used by advanced applications and vendor products were disabled. These facilities are called SVC Screening and SVCUPDTE (as they apply to the BLDL and STOW macros). The DESERV exit can be used along with the SVC screening and SVCUPDTE facilities to monitor accesses and updates to PDS and PDSE directories. The system calls the DESERV GET function when:

- The binder is used to bind a program object or a load module and it is searching for member names to be included. (Note the linkage editor does not use DESERV).
- The system is searching for modules to load into storage while processing the ATTACH, LINK, LOAD or XCTL functions.

In these situations the system uses DESERV GET rather than issuing BLDL. However, in some situations DESERV GET issues BLDL to perform the directory search (the BLDL function does not issue DESERV calls). The system calls the DESERV PUT function when:

- The binder is creating a program object in a PDSE (note the linkage editor does not use DESERV, nor does the binder use DESERV PUT when creating a load module in a PDS).
- An IEBCOPY job is loading a program object from an IEBCOPY unloaded data set.
- An IEBCOPY job is copying a member to a PDSE where one of the members names is greater than 63 bytes long.

In these situations the system uses DESERV PUT rather than issuing STOW. The DESERV PUT and STOW code do not interact. STOW does not issue DESERV PUT nor does DESERV PUT issue STOW.

Currently the system does not use the RENAME or UPDATE functions. SMP/E is the only known user of the DELETE function. The rename, update and delete functions and the STOW code do not interact.

With SVC screening, an SVC screen table is associated with a task control block (TCB). The table marks specific SVC numbers as not valid. The table also defines the address of a routine that gets control when an SVC that is not valid is issued. Then it's possible for the screen routine to inhibit the function, perform the function itself, or temporarily disable the SVC screening and re-issue the SVC. This technique provides a front and back end mechanism for SVC routines.

With SVCUPDTE, an application can dynamically replace or delete SVC table entries for the system or obtain the SVC number of a routine at a specified entry point. One specific use of the replace function of SVCUPDTE would use a scenario like the following to replace an IBM supplied SVC routine.

1. Extract the SVC entry for SVC 18 (BLDL) from the SVC table.

© Copyright IBM Corp. 1979, 1999

2. Issue SVCUPDTE to install the vendor's version of the BLDL function.
3. When an SVC 18 is issued, the vendor's BLDL module gets control.
4. The vendor's BLDL either performs the function and returns to the caller, or branch enters the IBM supplied BLDL code whose address was obtained earlier from the SVC table.

DESERV provides a task level exit for an interface that is similar to SVC screening for the SVC routines BLDL and STOW. DESERV also provides a global exit for an interface that is similar to the SVCUPDTE replace option. For more information on using SVC Screening and SVCUPDTE, refer to *OS/390 MVS Authorized Assembler Services Reference SET-WTO* and *OS/390 MVS Authorized Assembler Services Guide*.

Describing Task and Global Exits

Task Level Exit

The task level DESERV exit can be established for any TCB following the LPA's initialization. Once established for a task, the DESERV exit is given control for the DESERV functions that are issued under the TCB. The exit entry point is given control twice for each GET, PUT, RENAME, DELETE, or UPDATE invocation; once before DESERV executes and once immediately before the return from the DESERV function. The parameters passed to the exit indicate whether this call happens before or after the DESERV function executes.

If a DESERV FUNC=GET call is made for which there are no PDSEs in the concatenation, the DESERV code issues BLDL to search for the requested names. The task level exit is called in this case. However, if SVC screening is also active, the exit might:

- Not process this DESERV call (i.e. lets the SVC screen routine process the BLDL request) or
- Process this DESERV call with the pre- and post-processing exits, but also disable the SVC screening for BLDL in the pre-processing exit and enable the BLDL screening in the post-processing exit. A parameter passed to the exit indicates whether a BLDL will be or has been issued.

Just as with the SVC screening facility, the DESERV task level exit function enables the user to indicate that the exit should be propagated to subsequently attached tasks.

The first DESERV call for a task searches the TCB chain for a DESERV task level exit routine with propagate specified. DESERV searches the TCB chain following the originating TCB pointer (TCBOTC). If none exists, the task is marked to indicate that no DESERV task level exit exists. Therefore, for the propagate option to work, the exit routine must be established before issuing the ATTACH macro. This is roughly consistent with implementing SVC screening. The difference is that the SVC screening table is propagated at the time of the ATTACH, while the DESERV exit might not be propagated at ATTACH (for example, if the attached program is found in the job pack queue, no directory search, i.e. DESERV GET, is done to find the module).

Global Exit

The global DESERV exit can be established for the system following the initialization of LPA. When establishing a global exit, obtain the DST (DESERV Screen Table) storage in common storage. The DST is used to identify a DESERV exit. Once established, the DESERV exit gets control anytime DESERV GET, PUT, RENAME, DELETE, or UPDATE functions are called. The exit entry point is given control twice for each invocation, once before DESERV is executed, and once immediately after the return from the DESERV function. The call to the exit indicates whether this call is before or after DESERV executes. The global exit routine must reside in commonly addressable storage.

If a DESERV FUNC=GET call is made for which there are no PDSEs in the concatenation, the DESERV code issues BLDL to search for the requested names. The global exit is called in this case. If both the SVCUPDTE facility and the DESERV global exit are used, before implementing the global DESERV exit consider the interactions of the DESERV global exit and the routine that is given control when the SVC is issued.

Interactions Between the Task Level and Global Exits

If both task level and global DESERV exits have been defined, there is a prescribed calling sequence. The task level exit is called first. If the task level exit indicates that the DESERV function should be terminated (via a return code 4 from the exit), DESERV returns immediately to its caller. However, the global exit is given control when the task level exit returned with return code 0. The global exit can indicate (via return code 4) that control should pass back immediately to the DESERV caller. In this case before returning to the DESERV caller, the task level exit is given control indicating that the DESERV function is complete. After returning from the task level exit, DESERV returns to the caller.

If the global exit returns with return code 0, the DESERV function executes, making the post-processing exit calls. The post processing exit calls are made first to the global exit and second to the task level processing exit. This sequence (the reverse of the pre-processing exit sequence) is chosen to simulate the return sequence that would have been seen if both SVC screen routine and updated SVC routine were in place. The following diagram illustrates the exit routine call sequence:

```
DESERV GET, PUT, RENAME, DELETE or UPDATE is issued
  enter DESERV GET, PUT, RENAME, DELETE, or UPDATE
    call task exit for pre-processing
    if return_code = 0 then
      call global exit for preprocessing
      if return_code = 0 then
        process GET, PUT, RENAME, DELETE, or UPDATE
        GOTO post_process_global
      else if return_code = 4 then
        GOTO post_process_task
    else if return_code = 4 then
      return to DESERV caller

  Post_process_global:
    call global exit for post-processing

  Post_process_task:
    call task exit for post-processing
    return to DESERV caller
```

Figure 81. Exit Routine Call Sequence

Establishing Multiple Task level or Multiple Global Exits

The system identifies a DESERV exit by a DST (DESERV Screen Table). The system maintains at most one task level DST for each task, and at most one DST for the system (which represents the global exit). However, multiple DESERV exits can be established. To support multiple exits of a given type (task or global), when a DESERV FUNC=EXIT is issued with EXIT_OPTION=REPLACE, DESERV returns the address of the DST that was replaced (or zero if no DST was replaced). Then it is the responsibility of the newly defined exit to pass control to the previously defined exit.

Issuing DESERV FUNC=EXIT (invocation environment)

INTERRUPTS: enabled

STATE and KEY: supervisor state, or system key (0-7)

ASC Mode: P=H=S

AMODE, RMODE: no restrictions

LOCKS: none held

REGISTERS:

- All register contents except registers 15 and 0 are restored on return.
- Register 15 contains the return code and register 0 contains the reason code.
- No save area is required.

SERIALIZATION REQUIREMENTS: None. DESERV maintains serialization. Therefore the caller does not need to provide any ENQ-like serialization.

Invocation Syntax

The following figure illustrates the syntax for the DESERV EXIT function:

```
►►──┬──────────┬──DESERV FUNC=EXIT──┬──────────────────────────────────────►
    └─label────┘                    │                   ┌─TASK───┐
                                    └─,EXIT_SCOPE=──┬───┬─┴─GLOBAL─┴──┬──┘
                                                    │   │             │
►──┬──────────────────────────────────────────────────┴───────────────┴──────►
   │                   ┌─NOREPLACE─┐
   └─,EXIT_OPTION=─┬───┬─REPLACE───┴──┬──┘
                   │   ├─DELETE────┤
                   │   └─QUERY─────┘
►──┬──────────────────────────────────────────────────────────────────────►
   │                   ┌─dst_address─┐
   └─,EXIT_DST=────┬───┬─────────────┴──┬──┘
                   │   └─(2-12)──────┘
►──,EXIT_PREV_DSTPTR=──┬─dstPTR_address─┬──────────────────────────────────►
                       └─(2-12)─────────┘
►──┬──────────────────────────────────────────────────────────────────────►
   │                   ┌─S──────────────────────────────┐
   └─,MF=──────────┬───┬─L──────────────────────────────┴──┬──┘
                   │   │                   ┌─,COMPLETE─┐   │
                   │   └──(E,──┬─(1-12)──┬─┴─,NOCHECK────┴─)─┴──┘
                   │           └─label───┘
►──┬──────────────────────────────────────────────────────────────────────►◄
   │ ┌─,RETCODE=retcode─┐   │ ┌─,RSNCODE=rsncode─┐
   └─┴──────────────────┘   └─┴──────────────────┘
```

FUNC=EXIT
requests the DESERV function which operates on a DESERV exit. This keyword is always required except when MF=E is coded with "NOCHECK" and no other keywords are coded or MF=L is coded with no other keywords. For the MF=E case, the FUNC keyword and other keywords specified on the MF=L DESERV macro invocation are assumed to have been coded completely.

EXIT_SCOPE=GLOBAL or **TASK**
specifies whether the exit specified will be of a TASK level or of a GLOBAL level.

EXIT_OPTION=REPLACE or **NOREPLACE** or **DELETE** or **QUERY**
If EXIT_OPTION=REPLACE or NOREPLACE is coded, this specifies whether this invocation of the DESERV FUNC=EXIT should replace an existing DESERV EXIT (TASK or GLOBAL as specified by the EXIT_SCOPE parameter). EXIT_PREV_DST will return the existing exit or be set to zero if one does not exist.

If EXIT_OPTION=DELETE is coded, this indicates that the current exit is to be deleted (EXIT_OPTION=DELETE). In this case the EXIT_DST parameter specifies the address of the DST which is to be deleted. This address will be used as the compare value in a compare and swap operation, and only the currently active exit can be deleted. The DST address specified via the EXIT_PREV_DSTPTR parameter will be used as the swap value.

If EXIT_OPTION=QUERY is coded, this indicates that the current exit DST address is to be returned via the EXIT_PREV_DSTPTR parameter.

EXIT_DST=*deserv_exit_screen_table* **RX-Type Address** or **(2-12)** specifies the address of the DESERV Screen Table (DST).
The screen table is mapped by DST DSECT of the IGWDES mapping macro. For an EXIT_OPTION of either NOREPLACE or REPLACE, this parameter defines the DST and defines the exit routine address which becomes the currently active exit if the operation is successful. For an EXIT_OPTION of DELETE, this parameter defines the DST whose address is used as a compare value in a compare and swap operation when deleting the current DST (the address of the input DST is used as the compare value). If the compare fails, DESERV returns an error return and reason code. If EXIT_OPTION=QUERY is coded, this parameter is not required.

EXIT_PREV_DSTPTR=*addr_of_deserv_exit_screen_table* **RX-Type Address** or **(2-12)**
The EXIT_PREV_DSTPTR is an output parameter when an EXIT_OPTION of NOREPLACE, REPLACE or QUERY is specified, and an input parameter if an EXIT_OPTION of DELETE is specified.

For EXIT_OPTION=NOREPLACE or QUERY this parameter specifies a four byte field into which DESERV will return the address of the current DST (or zero if no DST exists).

For EXIT_OPTION=REPLACE, this parameter specifies a four byte field into which DESERV will return the address of the DST which was successfully replaced (or zero if no previous DST existed).

For EXIT_OPTION=DELETE, this parameter specifies a four byte field that points to the DST that DESERV restore as the current DST. This address is the swap value for the compare and swap operation.

MF=S or **L** or **{(E,{(1-12)** or **label}{,COMPLETE** or **NOCHECK})}** RX-Type Address or (1-12) - for MF=E second argument Default=S - if the MF keyword is not specified Default=COMPLETE - if MF=E is specified without the third argument (COMPLETE or NOCHECK).
specifies the format of the macro expansion.

The Standard form, S, checks all required keywords and keywords that are not valid. This form generates a complete inline expansion of the parameter list and code to call the Directory Entry Services routine. The standard form is for programs that are not reenterable, or for programs that do not change values in the parameter list.

L specifies the List form of the macro. This form generates a remote parameter list. Only keywords of argument type KEY or SYM can be coded. Registers are not valid because code generation does not occur, adcons are generated. Invalid keyword checking is done.

Keywords with defaults that are set by MF=L invocation are not reset to their default during MF=E invocation.

E specifies the Execute form of the macro. This form updates the remote parameter list (MF=L) and transfers control to the DESERV routine.

The second parameter for MF=E format is the address of the parameter list created by the MF=L DESERV invocation. This parameter must be specified as either an RX type of address (possibly the label from MF=L macro invocation) or a register enclosed in parentheses.

The third parameter, COMPLETE or NOCHECK, is optional. Default is COMPLETE. This argument specifies whether required keyword checking will be done. If MF=E is coded with the NOCHECK argument then no, some, or all allowed keywords can be specified, assuming that any missing keywords were coded on the MF=L macro invocation. If MF=E is coded with the COMPLETE argument or allowed to default, the parameter list will be zeroed out (except for the parameter list header) This sets all defaults because the defaults for the DESERV macro are 0. All required keywords must be specified.

RETCODE=retcode RX-Type Address or **(2-12)**
specifies the address where the return code returned by DESERV will be stored. Can not be specified on MF=L macro format. The default is not to store the return code in virtual storage. The return code is always returned in register 15 without regard to whether RETCODE is coded.

RSNCODE=rsncode RX-Type Address or **(2-12)**
specifies the address where the reason code returned by DESERV will be stored. Can not be specified on MF=L macro format. The default is not to store the return code in virtual storage. The return code is always returned in register 0 without regard to whether RETCODE is coded.

DESERV code is available and used by the system starting with DFSMS/MVS Version 1.1. However, invoking the EXIT function requires that the appropriate PTF be applied to the system to enable the support. Your program can test to determine if the appropriate level of DFSMS/MVS or PTF is installed.

If you have DFSMS/MVS Version 1.1 or 1.2 installed, you can check the DFA bit DFADEEXT to determine if the appropriate DESERV EXIT support code is installed. If DFADEEXT is on, the DESERV EXIT support code is available.

If you have DFSMS/MVS Version 1.3 or higher installed, the DESERV EXIT support code is available. The DFA field of DFARELS contains the release number of DFSMS/MVS. Your exit can test the DFSMS/MVS release by examining DFARELS. DFADEEXT will always be on.

If the DESERV FUNC=EXIT interface is called on DFSMS/MVS without the support code being available, DESERV returns an error return and reason code.

Installing or Replacing the DESERV Exit

Your program, while operating in task mode, can establish DESERV exits by issuing the DESERV macro with FUNC=EXIT and appropriate parameters.

If EXIT_OPTION =	then:	and these parameters...
REPLACE	The new exit definition replaces the most recently defined exit. The DST (DESERV Screen Table) address of the previously defined exit's DST is returned to the caller.	EXIT_DST and EXIT_PREV_DSTPTR must be specified.

If EXIT_OPTION =	then:	and these parameters...
NOREPLACE (with no exit currently existing)	The exit definition is activated. A value of zero is returned as the previous DST address.	See above.
NOREPLACE (with an exit currently existing)	The currently defined exit remains the active exit. The address of the currently defined DST is returned as the previous DST address.	See above.

Note: If EXIT_OPTION=REPLACE is specified the caller must expect that a previous DST address is returned. It is the responsibility of the caller to replace the old DST address when eventually disabling the new exit. Also the exit routine might need to be aware that there was a previous exit defined, and might choose to invoke the previously defined exit.

EXIT_DST defines the DST address. The DESERV FUNC=EXIT caller owns and manages storage for the DST. The DST can be encapsulated inside other application managed control blocks. This can help the exit routine determine the context of the application in which it was called. When the DST is used with a task level exit, the DST_FLAGS_PROP bit can be set on to indicate that this DST should be propagated to tasks that are attached by this task. The propagate function is not supported for the global exit. The format of the DST is shown in Figure 82.

Figure 82. DESERV Screen Table Structure

Offset	Length or Bit Pattern	Name	Description
0 (X'0')	20	DST	(structure)
0 (X'0')	16	DST_HEADER	(character)
0 (X'0')	8	DST_ID	Eyecatcher 'IGWDST' (character)
08 (X'08')	4	DST_LEN	Length of DST
	X'14'	DST_LEN_IV	Constant to be used with DST_LEN
12 (X'0C')	1	DST_LEV	Control block level (unsigned)
	X'11'	DST_LEV_IV	Constant to be used with DST_LEV
13 (X'0D')	1	DST_FLAGS	DST flags (unsigned)
	xxxx xxx.	-	Reserved
	1	DST_FLAGS_PROP	Propagate this DST to lower level tasks
14 (X'0E')	2	DST_RES	Reserved
16 (X'10')	4	DST_EXIT	Address of exit routine screen table (address)

EXIT_PREV_DSTPTR returns the address of the DST that was defined prior to this DESERV FUNC=EXIT call.

Deleting the DESERV Exit

An application that has established a DESERV exit can delete the currently active exit by issuing the DESERV macro with FUNC=EXIT. You can specify the following:

Figure 83. Deleting the DESERV Exit

For this parameter	specify this:
EXIT_SCOPE	(application dependent)
EXIT_OPTION	DELETE
EXIT_DST	Currently active DST to be deleted
EXIT_PREV_DSTPTR	The DST to become active

Note: The address of the DST must match that of the currently defined DST. If the addresses do not match, DESERV returns error return and reason codes to indicate the DELETE operation has failed and the active DST remains unaffected.

For a task related exit, the DESERV exit is implicitly deleted when the task ends. A global exit can only be explicitly deleted by issuing a DESERV FUNC=EXIT call.

Determining If a DESERV Exit Is Active

To determine if a DESERV exit is active issue the DESERV macro with FUNC=EXIT and choose the following options:

Figure 84. Determining If a DESERV Exit Is Active

For this parameter	choose this:
EXIT_OPTION	QUERY
EXIT_SCOPE	(application dependant)
EXIT_PREV_DSTPTR	The address of a 4 byte area into which the currently active DST address is returned (or zero if there is no currently defined DST address.)

Writing the DESERV Exit

Exit Processing

A DESERV exit gets control once prior to any DESERV GET, PUT, RENAME, UPDATE or DELETE function processing, and once immediately prior to DESERV's return to the caller. A DESERV exit receives control in key 0 and supervisor state. Register 13 points to an 18-word key 0 register save area.

The DESX DSECT maps the input to the exit routines and is defined in the IGWDES macro. Register 1 points to the DESX on entry to the exit routine. The DESX structure is shown in Figure 85.

Figure 85. DESX Structure Mapping DESERV Exit Parameter List

Offset	Length or Bit Pattern	Name	Description
0 (X'0')	36	DESX	(structure)
0 (X'0')	16	DESX_HEADER	(character)
0 (X'0')	8	DESX_ID	Eyecatcher - IGWDESX (character)
08 (X'08')	4	DESX_LEN	Length of DESX (signed)
	X'24'	DESX_LEN_IV	Constant to be used with DESX_LEN
12 (X'0C')	1	DESX_LEV	Control block level (character)
	X'01'	DESX_LEV_IV	Constant to be used with DESX_LEV
13 (X'0D')	3	-	Reserved
16 (X'10')	4	DESX_DESP_PTR	Address of caller's DESP (address)
20 (X'14')	4	DESX_DST	Address of DESERV screen table (address)
24 (X'18')	1	DESX_CALLER_KEY	Key of DESERV caller in bits 0-3 (unsigned)
25 (X'19')	1	DESX_FLAGS	(bitstring)
	1...	DESX_BLDL_BIT	DESERV issues BLDL to process this GET request
	.1..	DESX_PREV_BIT	EXIT called before DESERV PUT or GET function
	..1.	DESX_POST_BIT	EXIT called after DESERV PUT or GET function
26 (X'1A')	2	-	Reserved
28 (X'1C')	4	DESX_RETURN_CODE	Return code to be returned to DESERV caller (unsigned)
32 (X'20')	4	DESX_REASON_CODE	Reason code to be returned to DESERV caller (unsigned)

Note that the DESERV return and reason codes, with the exception of the PUT codes, can be found in *DFSMS/MVS Using Data Sets*. See Figure 100 on page 275 for the PUT return and reason codes.

Parameters Related to the GET Function

If the DESERV exit gets control for a DESERV GET function invocation, DESX_DESP_PTR points to the DESERV parameter list. If the DESP field DESP_FUNC=X'01' (DESP_FUNC_GET), this indicates a GET function parameter list. See Figure 86 on page 257 for the DESP structure for fields pertaining to a DESERV GET invocation.

DESERV GET will return information on selected members. This information is returned in a DESB structure. The DESB is mapped by the DESB DSECT in the IGWDES macro. If the storage for the DESB is provided by the DESERV GET, the DESP_AREA_PTR field contains the address of this storage. Alternatively the caller may request that DESERV GET obtain the storage for the DESB. In this case the DESP_AREAPTR_PTR field contains the address of a 4 byte area into which DESERV GET will return the address of a DESB. The DESB will be obtained in the subpool identified by DESP_SUBPOOL (or default to subpool zero). The flag DESP_SUBPOOL_FLG indicates whether the subpool was specified explicitly by the DESERV GET caller.

A DESERV GET invocation identifies the members to be searched for by a name list, a PDS format directory entry, or an SMDE. The DESP field DESP_GETTYPE defines the get type. If the get type is a PDSDE (i.e. the member to be searched for is defined by a PDS format directory entry), the DESP_PDSDE_PTR points to a directory entry as returned by the BLDL macro. The PDS2 DSECT in the IHAPDS macro maps this structure. The function of DESERV GET for a PDSDE get type depends on the type of library identified by the concatenation number in the PDS style directory entry. If the concatenation number identifies a PDS, the GET function is simply to convert the PDS style directory entry into a SMDE. If the concatenation number identifies a PDSE, the GET function is to connect to the member identified by the PDS2TTRP field, and to return the appropriate SMDE. In either case the SMDE is retuned (if in the PDSE case the member actually exists) in the data portion of the output buffer (DESB, mapped below).

If the get type is name list, the DESL area points to the names to be searched for. The DESP_NAME_LIST_PTR points to the DESL and the DESL DSECT in the IGWDES macro maps it. A DESL is an array consisting of the number of entries the DESP field DESP_NAME_LIST2 defines. The DESL parameter list is shown in Figure 87 on page 260.

If the get type is SMDE, the DESP_SMDE_PTR points to a system-managed directory entry (SMDE) as returned by DESERV GET. DESERV GET will cause a connection to the member identified by the SMDE. DESERV GET will return a copy of the SMDE in the output DESB. The SMDE returned will of course have updated connect token and connect id fields.

There are two input flags which control DESERV GET's view of the PDS or PDSE to be searched. If the DESP_C370LIB flag is on, a PDS may be viewed as a C370LIB. This means that if the PDS has a special member named @@DC370$, this member is treated as the "real" directory for the PDS. If the DESP_SYSTEM_DCB is on, this indicates that the caller (who must be authorized) has indicated that this DCB is "owned by the system" and is not on any DEB chain, therefore DEBCHK should not be done.

Note: The name of the special member @@DC370$ might not display correctly on your screen or printer. The first two characters are `X'7C'` and the last character is `X'5B'`.

Figure 86 (Page 1 of 4). Structure of DESP for DESERV GET Invocations

Offset	Length or Bit Pattern	Name	Description
00 (`X'00'`)	104	DESP	DE Services parameter list (structure)

Figure 86 (Page 2 of 4). Structure of DESP for DESERV GET Invocations

Offset	Length or Bit Pattern	Name	Description
00 (X'00')	16	DESP_HEADER	(character) 'IGWDESP'
00 (X'00')	8	DESP_ID	Eyecatcher IGWDESP (character)
08 (X'08')	24	DESP_LEN	Length of DESP (signed)
	X'04'	DESP_LEN_IV	Constant to be used with DESP_LEN
12 (X'0C')	1	DESP_LEV	Control block level (character)
	X'01'	DESP_LEV_IV	Constant to be used with DESP_LEV
13 (X'0D')	3	-	Reserved
16 (X'10')	1	DESP_FUNC	Function type (unsigned)
	X'07'	DESP_FUNC_DELETE	Function is DELETE
	X'08'	DESP_FUNC_RENAME	Function is RENAME
	X'09'	DESP_FUNC_UPDATE	Function is UPDATE
	X'04'	DESP_FUNC_PUT	Function is PUT
	X'01'	DESP_FUNC_GET	Function is GET
	X'00'	DESP_FUNC_OMITTED	Function is omitted
17 (X'11')	3	-	Reserved
20 (X'14')	4	-	Reserved
24 (X'18')	12	DESP_DATA	Function data (character)
24 (X'18')	2	DESP_FLAGS	Flags (bitstring)
	1...	DESP_BYPASS_LLA	0=USE LLA, 1=BYPASS LLA
	.x..	-	Reserved
	..1.	DESP_SUBPOOL_FLG	0=SUBPOOL not specified, 1=SUBPOOL specified
	...1	DESP_C370LIB	1=treat PDSs as C370LIB if @@DC370$ member exists
	.... xx..	-	Reserved
	.... ..1.	DESP_SYSTEM_DCB	1=treat DCB as a system DCB
26 (X'1A')	1	-	Reserved
27 (X'1B')	1	-	Reserved
28 (X'1C')	1	DESP_LIBTYPE	Indicates whether a DCB or DEB is input =X'02', DEB input=X'01') (unsigned)
	X'02'	DESP_LIBTYPE_DCB	Constant to be used with DESP_LIBTYPE
	X'01'	DESP_LIBTYPE_DEB	Constant to be used with DESP_LIBTYPE
	X'00'	DESP_LIBTYPE_OMITTED	Constant to be used with DESP_LIBTYPE

Figure 86 (Page 3 of 4). Structure of DESP for DESERV GET Invocations

Offset	Length or Bit Pattern	Name	Description
29 (X'1D')	1	DESP_GETTYPE	Indicates whether Name List or PDSDE is input. (NAME_LIST input=(X'01', PDSDE input=X'02') (unsigned)
	X'03'	DESP_GETTYPE_SMDE	Constant to be used with DESP_GETTYPE
	X'02'	DESP_GETTYPE_PDSDE	Constant to be used with DESP_GETTYPE
	X'01'	DESP_GETTYPE_NAME_LIST	Constant to be used with DESP_GETTYPE
	X'00'	DESP_GETTYPE_OMITTED	Constant to be used with DESP_GETTYPE
30 (X'1E')	1	-	Reserved
31 (X'1F')	1	-	Reserved
32 (X'20')	1	-	Reserved
33 (X'21')	1	DESP_SUBPOOL	Subpool number for getting DESB.
34 (X'22')	1	DESP_CONN_INTENT	Connect intent (unsigned)
	X'03'	DESP_CONN_INTENT_INPUT	INPUT
	X'02'	DESP_CONN_INTENT_EXEC	EXEC
	X'01'	DESP_CONN_INTENT_HOLD	HOLD
	X'00'	DESP_CONN_INTENT_NONE	None
35 (X'23')	1	-	Reserved
36 (X'24')	4	DESP_DCB_PTR	DCB address, valid if DESP_LIBTYPE=X'02' (address)
40 (X'28')	4	DESP_DEB_PTR	DEB address, valid if DESP_LIBTYPE=X'01' (address)
44 (X'2C')	4	DESP_CONN_ID_PTR	Connect identifier address
48 (X'30')	4	DESP_AREAPTR_PTR	Address of output field for buffer address
52 (X'34')	4	DESP_AREA_PTR	Buffer address
56 (X'38')	4	DESP_AREA2	Buffer length (unsigned)
60 (X'3C')	4	-	Reserved
64 (X'40')	4	-	Reserved
68 (X'44')	4	DESP_ENTRY_GAP	Entry gap size (signed)
72 (X'48')	4	-	Reserved
76 (X'4C')	4	-	Reserved
80 (X'50')	4	DESP_NAME_LIST_PTR	Name list address, valid if DESP_GETTYPE=X'01' (address)
84 (X'54')	4	DESP_NAME_LIST2	Input list number of entries, valid if DESP_GETTYPE=X'01' (unsigned)
88 (X'58')	4	-	Reserved

Figure 86 (Page 4 of 4). Structure of DESP for DESERV GET Invocations

Offset	Length or Bit Pattern	Name	Description
92 (X'5C')	4	DESP_PDSDE_PTR	BLDL directory entry address, valid if DESP_GETTYPE=X'02' (address)
92 (X'5C')	4	DESP_SMDE_PTR	SMDE directory entry address, valid if DESP_GETTYPE=X'03' (address)
96 (X'60')	4	-	Reserved
100 (X'64')	4	-	Reserved

Figure 87. DESL Structure

Offset	Length or Bit Pattern	Name	Description
00 (X'00')	16	DESL	Name list (structure)
00 (X'00')	16	DESL_ENTRY	Name list entry (character)
00 (X'00')	1	DESL_FLAGS	Flags (unsigned)
	1...	DESL_MODULE_BUFFERED_LLA	Module is staged by LLA
1 (X'1')	1	DESL_CODE	Result code (New name exists=X'03', Error=X'02', Not found or not processed=X'01', Found=X'00') (unsigned)
	X'03'	DESL_CODE_NEWNAME_EXISTS	For func=rename, indicates a new name already existed in the PDSE.
	X'02'	DESL_CODE_ERROR	An unexpected error has occurred. The DESL_ERRCODE field is set to a DESRF value
	X'01'	DESL_CODE_NOTFOUND	Entry not found or entry not processed. If func=rename, old name was not found.
	X'00'	DESL_CODE_SUCC	Entry successfully processed
2 (X'2')	2	DESL_ERRCODE	Error reason code (low order halfword of DESERV reason code if error) (unsigned)
4 (X'4')	4	-	Reserved
08 (X'08')	4	DESL_SMDE_PTR	Pointer to SMDE within DESB. Output for GET function, input for UPDATE function (address)
08 (X'08')	4	DESL_NEW_NAME_PTR	Pointer to new name (DESN) descriptor for RENAME function (address)
12 (X'0C')	4	DESL_NAME_PTR	Pointer to name (DESN) descriptor for GET and DELETE functions (address)
12 (X'0C')	4	DESL_OLD_NAME_PTR	Pointer to old name (DESN) descriptor for RENAME function (address)

The DESERV GET caller will have built the DESL to point to variable length names. The DESN DSECT maps these names in the IGWDES macro. See Figure 88 on page 261 for the DESN parameter list.

Figure 88. DESN Parameter List

Offset	Length or Bit Pattern	Name	Description
00 (X'00')	variable	DESN	Name record (structure)
00 (X'00')	2	DESN_LEN	Length of name that follows (unsigned)
2 (X'02')	variable	DESN_VAL	Name data (character)

The DESP_CONN_INTENT field of the DESP indicates the connection intent requested by the caller. The connection intent only has an effect if the name is found in a PDSE. If the connection intent is DESP_CONN_INTENT_HOLD (X'01'), the effect is similar to a BLDL invocation (because the member is connected for HOLD which is not sufficient to read the member). If the connection intent is DESP_CONN_EXEC (X'02') or DESP_CONN_INTENT_INPUT (X'03'), the effect is similar to a FIND invocation (because the member is connected and sufficient control blocks are built so that the member can be read). The GET function does not currently support a connect intent of NONE.

The output from DESERV GET consists of flags and error codes in the DESL (if the get type is name list) as well as an SMDE (system managed directory entry) pointer. For a gettype of name list, the SMDE is pointed to by the DESL_SMDE_PTR field. For a gettype of PDSDE, the SMDE is in the DESB at the label DESB_DATA. The SMDE is mapped by the SMDE DSECT in the IGWSMDE macro and the PMAR DSECT in the IEWPMAR macro. The SMDE resides in the output buffer as provided by the caller of DESERV GET. The output buffer is mapped by the DESP DSECT of the IGWDES macro. The DESB structure is shown in Figure 89.

The basic SMDE format is shown in Figure 90 on page 262.

Figure 89 (Page 1 of 2). DESB Parameter List

Offset	Length or Bit Pattern	Name	Description
00 (X'00')	variable	DESB	DEServ buffer header (structure)
00 (X'00')	40	DESB_FIXED	(character)
00 (X'00')	16	DESB_HEADER	(character)
00 (X'00')	8	DESB_ID	Eyecatcher - IGWDESB (character)
08 (X'08')	4	DESB_LEN	Length of buffer (signed)
12 (X'0C')	1	DESB_LEV	Control block level (character)
	X'01'	DESB_LEV_IV	Constant to be used with DESB_LEV
13 (X'0D')	3	-	Reserved
16 (X'10')	4	DESB_NEXT	Next buffer pointer (address)

Figure 89 (Page 2 of 2). DESB Parameter List

Offset	Length or Bit Pattern	Name	Description
20 (X'14')	4	-	Reserved
24 (X'18')	4	DESB_COUNT	Count of entries in this buffer (unsigned)
28 (X'1C')	4	DESB_AVAIL	Start of free space in buffer (address)
32 (X'20')	1	-	Reserved
33 (X'21')	1	DESB_SUBPOOL	Subpool number (unsigned)
34 (X'22')	2	DESB_GAP_LEN	Length of user-requested gap (unsigned)
36 (X'24')	4	-	Reserved
40 (X'28')	variable	DESB_DATA	Start of data area (character)

Figure 90 (Page 1 of 3). SMDE Format

Offset	Length or Bit Pattern	Name	Description
00 (X'00')	variable	SMDE	Member directory entry (structure)
00 (X'00')	44	SMDE_BASIC	Start of basic section (character)
00 (X'00')	16	SMDE_HDR	Header (character)
00 (X'00')	8	SMDE_ID	Eyecatcher (character)
08 (X'08')	4	SMDE_LEN	Length of control block. This is the sum of the sizes of the SMDE sections and the size of the user data. (unsigned)
12 (X'0C')	1	SMDE_LVL	SMDE version number (unsigned)
	X'01'	SMDE_LVL_VAL	Constant to be used with SMDE_LVL
13 (X'0D')	3	-	Reserved

Figure 90 (Page 2 of 3). SMDE Format

Offset	Length or Bit Pattern	Name	Description
16 (X'10')	1	SMDE_LIBTYPE	Source library type. Possible values are declared below with names like SMDE_LIBTYPE_XXX. (unsigned)
	X'03'	SMDE_C370LIB	Constant to be used with SMDE_LIBTYPE
	X'02'	SMDE_LIBTYPE_HFS	Constant to be used with SMDE_LIBTYPE
	X'01'	SMDE_LIBTYPE_PDSE	Constant to be used with SMDE_LIBTYPE
	X'00'	SMDE_LIBTYPE_PDS	Constant to be used with SMDE_LIBTYPE
17 (X'11')	1	SMDE_FLAG	Flag byte (bitstring)
	1...	SMDE_FLAG_ALIAS	Entry is an alias
	.1..	SMDE_FLAG LMOD	Member is a program
	..xx xxxx	*	Reserved
18 (X'12')	2	-	Reserved, must be zero
20 (X'14')	5	-	Extended MLTK (character)
20 (X'14')	1	-	Reserved, must be zero
21 (X'15')	4	SMDE_MLTK	MLT and concatenation number (character)
21 (X'15')	3	SMDE_MLT	MLT of member - zero if HFS (character)
24 (X'18')	1	SMDE_CNCT	Concatenation number (unsigned)
25 (X'19')	1	SMDE_LIBF	Library flag - Z-byte (unsigned)
	X'02'	SMDE_LIBF_TASKLIB	Constant to be used with SMDE_LIBF
	X'01'	SMDE_LIBF_LINKLIB	Constant to be used with SMDE_LIBF
	X'00'	SMDE_LIBF_PRIVATE	Constant to be used with SMDE_LIBF
26 (X'1A')	2	SMDE_NAME_OFF	Name offset (signed)
28 (X'1C')	2	SMDE_USRD_LEN	User data length (signed)
28 (X'1C')	2	SMDE_PMAR_LEN	Sum of lengths of program management attribute record sections (PMAR, PMARR, PMARL) (signed)
30 (X'1E')	2	SMDE_USERD_OFF	User data offset (signed)
30 (X'1E')	2	SMDE_PMAR_OFF	Program management attribute record offset (signed)
32 (X'20')	2	SMDE_TOKEN_LEN	Token length (signed)

Figure 90 (Page 3 of 3). SMDE Format

Offset	Length or Bit Pattern	Name	Description
34 (X'22')	2	SMDE_TOKEN_OFF	Token data offset (signed)
36 (X'24')	2	SMDE_PNAME_OFF	Primary name offset, zero for non-alias SMDES or if library type is a PDS and this is not a program. (signed)
38 (X'26')	2	SMDE_NLST_CNT	Number of note list entries that exist at beginning of user data field. Always zero for non-PDS members. (signed)
40 (X'28')	4	-	Reserved
44 (X'2C')	variable	SMDE_SECTIONS	Start of entry sections (character)

Figure 91 through Figure 94 on page 265 shows the optional SMDE_SECTIONS, or extensions to the SMDE.

Figure 91. Directory Entry Name Section

Offset	Length or Bit Pattern	Name	Description
00 (X'00')	variable	SMDE_NAME	Name descriptor (structure)
00 (X'00')	2	SMDE_NAME_LEN	Length of entry name (signed)
2 (X'02')	variable	SMDE_NAME_VAL	Entry name (character)

Figure 92. Directory Entry Notelist Section (PDS Only)

Offset	Length or Bit Pattern	Name	Description
00 (X'00')	variable	SMDE_NLST	Note list extension (structure)
00 (X'00')	4	SMDE_NLST_ENTRY	Note list entries (character)
00 (X'00')	3	SMDE_NLST_RLT	Note list record location token (character)
3 (X'03')	1	SMDE_NLST_NUM	Number of RLT described by this note list block. If 0 this is not a notelist but a data block. (unsigned)

Figure 93 (Page 1 of 2). Directory Entry Token Section

Offset	Length or Bit Pattern	Name	Description
00 (X'00')	32	SMDE_TOKEN	(structure)
00 (X'00')	4	SMDE_TOKEN_CONNID	CONNECT_IDENTIFIER (unsigned)
4 (X'04')	4	SMDE_TOKEN_ITEMNO	Item number (unsigned)

Figure 93 (Page 2 of 2). Directory Entry Token Section

Offset	Length or Bit Pattern	Name	Description
08 (X'08')	24	SMDE_TOKEN_FT	File token (character)

Figure 94. Directory Entry Primary Name Section

Offset	Length or Bit Pattern	Name	Description
00 (X'00')	variable	SMDE_PNAME	Primary name descriptor (structure)
00 (X'00')	2	SMDE_PNAME_LEN	Length of primary name (signed)
2 (X'02')	variable	SMDE_PNAME_VAL	Primary name (character)

If the SMDE represents a directory entry for a program (either a load module or a program object) the program's attributes are defined by the PMAR structure. The PMAR is a subfield of the SMDE and its offset is defined by the field SMDE_PMAR_OFF. Figure 95 shows the basic PMAR definition. Figure 96 on page 268 and Figure 97 on page 271 show the PMAR extensions for program objects (PMARL) and load modules (PMARR), respectively.

If the SMDE represents a data member of a PDS or a PDSE, the SMDE_USRD_OFF field indicates the offset into the SMDE for the user data of the directory entry.

Figure 95 (Page 1 of 4). Directory Entry Name Section. Data is always present at offset SMDE_PMAR_OFF in an SMDE.

Offset	Length or Bit Pattern	Name	Description
00 (X'00')	30	PMAR	Basic section of program user data (structure)
00 (X'00')	30	PMAR_ENTRY	Alternative name for the PMAR section (character)
00 (X'00')	2	PMAR_SLEN	Section length (unsigned)
2 (X'02')	1	PMAR_LVL	PMAR format level (unsigned)
	X'02'	PMAR_LVL_VAL	Constant to be used with PMAR
	X'01'	PMAR_PM1_VAL	Constant to be used with PMAR
	X'02'	PMAR_PM2_VAL	Constant to be used with PMAR

Figure 95 (Page 2 of 4). Directory Entry Name Section. Data is always present at offset SMDE_PMAR_OFF in an SMDE.

Offset	Length or Bit Pattern	Name	Description
3 (X'03')	1	PMAR_PLVL	Bind processor creating object 1 - E-level linkage editor 2 - F-level linkage editor 3 - (VS1/VS2) linkage editor 4 - XA linkage editor 5 - binder version 1. (unsigned)
	X'01'	PMAR_PLVL_E_VAL	Constant to be used with PMAR_PLVL
	X'02'	PMAR_PLVL_F_VAL	Constant to be used with PMAR_PLVL
	X'03'	PMAR_PLVL_AOS_VAL	Constant to be used with PMAR_PLVL
	X'04'	PMAR_PLVL_XA_VAL	Constant to be used with PMAR_PLVL
	X'05'	PMAR_PLVL_B1_VAL	Constant to be used with PMAR_PLVL
	X'06'	PMAR_PLVL_B2_VAL	Constant to be used with PMAR_PLVL
4 (X'04')	4	PMAR_ATR	Attribute bytes (character)
4 (X'04')	1	PMAR_ATR1	First attribute byte. These flags must be at the same offsets as the corre-sponding flags in PDS2ATR1 declared by macro IHAPDS. (bitstring)
	1...	PMAR_RENT	Reenterable
	.1..	PMAR_REUS	Reusable
	..1.	PMAR_OVLY	Overlay structure
	...1	PMAR_TEST	Module to be tested - TSO/E TEST
	.... 1...	PMAR LOAD	Only loadable
	.... .1..	PMAR_SCTR	Scatter format
	.... ..1.	PMAR_EXEC	Executable
	1	PMAR_1BLK	Load module contains only one block of text data and has no RLD data.

Figure 95 (Page 3 of 4). Directory Entry Name Section. Data is always present at offset SMDE_PMAR_OFF in an SMDE.

Offset	Length or Bit Pattern	Name	Description
5 (X'05')	1	PMAR_ATR2	Second attribute byte. These flags must be at the same offsets as the corresponding flags in PDS2ATR2 declared by macro IHAPDS. (bitstring)
	1...	PMAR_FLVL	If on, the program cannot be processed by the E level linkage editor. If off, the program can be processed by any level of the linkage editor or the binder.
	.1..	PMAR_ORGO	Linkage editor assigned origin of first block of text is zero.
	..x.	-	Reserved
	...1	PMAR_NRLD	Program contains no RLD items
	.... 1...	PMAR_NREP	Module cannot be reprocessed by the linkage editor
	.... .1..	PMAR_TSTN	Module contains TSO/E TEST symbol cards
	.... ..x.	-	Reserved
	1	PMAR REFR	Refreshable program
6 (X'06')	1	PMAR_ATR3	Third attribute byte. (bitstring)
6 (X'06')	1	PMAR_FTB1	Alternative name for flags byte. These flags must be at the same offsets as the corresponding flags in PDS2FTB1 declared by macro IHAPDS. (bitstring)
	x...	-	Reserved
	.1..	PMAR_BIG	This program requires 16MB or more of virtual storage.
	..1.	PMAR_PAGA	Page alignment is required
	...1	PMAR_XSSI	SSI information present
	.... 1...	PMAR_XAPF	APF information present
	.... .1..	PMAR_LFMT	PMARL follows PMAR.
	xx	-	Reserved
7 (X'07')	1	PMAR_ATR4	Fourth attribute byte (bitstring)

Figure 95 (Page 4 of 4). Directory Entry Name Section. Data is always present at offset SMDE_PMAR_OFF in an SMDE.

Offset	Length or Bit Pattern	Name	Description
7 (X'07')	1	PMAR_FTB2	Alternative name for flags byte. These flags must be at the same offsets as the corresponding flags in PDS2FTB2 declared by macro IHAPDS. (bitstring)
	1...	PMAR_ALTP	Alternate primary flag. If on for a primary name, indicates primary name was generated by the binder. If on for an alias, indicates the long alias name was specified as the primary name on the bind.
	.xx.	-	Reserved
	...1	PMAR_RMOD	RMODE is ANY.
	.... xx..	PMAR_AAMD	Alias entry point addressing mode. If B'00', AMODE is 24. If B'10', AMODE is 31. If B'11', AMODE is ANY.
	xx	PMAR_MAMD	Main entry point addressing mode. If B'00' AMODE is 24. If B'10', AMODE is 31. If B'11', AMODE is ANY.
08 (X'08')	1	-	Reserved
9 (X'09')	1	PMAR_AC	APF authorization code (unsigned)
10 (X'0A')	4	PMAR_STOR	Virtual storage required (unsigned)
14 (X'0E')	4	PMAR_EPM	Main entry point offset (unsigned)
18 (X'12')	4	PMAR_EPA	This entry point offset (unsigned)
22 (X'16')	4	PMAR_SSI	SSI information (bitstring)
22 (X'16')	1	PMAR_CHLV	Change level of member (unsigned)
23 (X'17')	1	PMAR_SSFB	SSI flag byte (bitstring)
24 (X'18')	2	PMAR_MSER	Member serial number (Reserved)
26 (X'1A')	4	-	Reserved
30 (X'1E')	variable	PMAR_END	End of basic section (character)

Figure 96. LSLoader Attributes Unique to Program Objects. If PMAR_LFMT=ON this section follows the PMAR basic section.

Offset	Length or Bit Pattern	Name	Description
00 (X'00')	50	PMARL	LSLoader section for program objects (structure)
00 (X'00')	2	PMARL_SLEN	Section length (unsigned)
2 (X'02')	48	PMARL_DATA	Section data (character)
2 (X'02')	4	PMARL_ATR	Attribute bytes (character)
2 (X'02')	1	PMARL_ATR1	Fifth attribute byte (bitstring)
	1...	PMARL_NMIG	This program object cannot be converted directly to PDS load module format.
	.1..	PMARL_PRIM	FETCHOPT PRIME option
	..1.	PMARL_PACK	FETCHOPT PACK option
	...x xxxx	-	Reserved
3 (X'03')	1	PMARL_ATR2	Sixth attribute byte (bitstring)
	1...	PMARL_CMPR	Compressed format module
	.1..	PMARL_1RMOD	1st segment is RMODE Any, set for PM2-level PO only
	..1.	PMARL_2RMOD	2nd segment is RMODE Any, set for PM2-level PO if there are at least two segments.
	...1	PMARL_SEGM	Loader data includes a Segment Table with >1 loadable entry or a Gas Table, set for PM2-level PO only.
	.... 1...	PMARL_1ALIN	1st segment is page-aligned, set for PM2-level PO only
	.... .1..	PMARL_2ALIN	2nd segment is page-aligned, set for PM2-level PO if there are at least 2 segments.
	.... ..1.	PMARL_FILL	FILL option specified set for PM2-level PO only
	x	-	Reserved
4 (X'04')	1	PMARL_FILLVAL	FILL character value set for PM2-level PO only
5 (X'05')	1	-	Reserved

THE FOLLOWING NOTED FIELDS ARE NOT INTENDED FOR USE. INCLUDED HERE FOR INFORMATION PURPOSES ONLY.

6 (X'06')	4	PMARL_MPGS	Total length of program on DASD in pages (unsigned)
10 (X'0A')	40	PMARL_MDAT	DASD program descriptors (character)
10 (X'0A')	4	PMARL_TXTL	Length of text (unsigned)
14 (X'0E')	4	PMARL_TXTO	Offset to text (address)
18 (X'12')	4	PMARL_BDRL	Length of binder index (unsigned)
22 (X'16')	4	PMARL_BDRO	Offset to binder index (address)
26 (X'1A')	4	PMARL_RDTL	Length of PRDT (unsigned)
30 (X'1E')	4	PMARL_RDTO	Offset to PRDT (address)
34 (X'22')	4	PMARL_RATL	Length of PRAT (unsigned)
38 (X'26')	4	PMARL_RATO	Offset to PRAT (address)
42 (X'2A')	4	PMARL_NVSPGS	Number of virtual storage pages to contain program object, for PM2-level PO
42 (X'2A')	4	PMARL_LMDL	Length of LSLoader data (unsigned) for PM1-level PO
46 (X'2E')	4	PMARL_LMDO	Offset to LSLoader data (address)
50 (X'32')	2	PMARL_NSEG	Number of loadable segments
52 (X'34')	2	PMARL_NGAS	Count of entries in Gas Table
54 (X'36')	4	PMARL_1STOR	Virtual storage required for first loadable segment, valid when PMARL_NSEG > 1.
58 (X'3A')	4	PMARL_2STOR	Virtual storage required for second loadable segment, valid when PMARL_NSEG > 1.
62 (X'3E')	4	PMARL_2TXTO	Offset to second txt segment including gas, valid when PMARL_NSEG > 1.

END INFORMATION ONLY FIELDS.

66 (X'42')	16	PMARL_TRACE	AUDIT trace data
66 (X'42')	4	PMARL_DATE	Date saved
70 (X'46')	4	PMARL_TIME	Time saved
74 (X'4A')	8	PMARL_USER	User or job identification
82 (X'52')	variable	PMARL_END	End of LSLoader section (character)

Figure 97 (Page 1 of 2). Attributes Unique to Load Modules (PDS only). If PMAR_LFMT=OFF then this section follows the PMAR basic section.

Offset	Length or Bit Pattern	Name	Description
00 (X'00')	23	PMARR	Load module (PDS) attributes section (structure)
00 (X'00')	2	PMARR_SLEN	Section length (unsigned)
2 (X'02')	21	PMARR_DATA	Section data (character)
2 (X'02')	8	PMARR_TTRS	TTR fields (character)
2 (X'02')	3	PMARR_TTRT	TTR of first block of text (character)
5 (X'05')	1	PMARR_ZERO	Zero (character)
6 (X'06')	3	PMARR_TTRN	TTR of note list or scatter translation table. Used for modules in scatter load format or overlay structure only. (character)
9 (X'09')	1	PMARR_NL	Number of entries in note list for scatter format modules and modules in overlay structure, otherwise zero. (address)
10 (X'0A')	2	PMARR_FTBL	Length of first block of text (signed)
12 (X'0C')	3	PMARR_ORG	Load module origin if 0 (unsigned)
12 (X'0C')	2	-	Reserved
14 (X'0E')	1	PMARR_RLDS	Number of RLD/CTL records that follow the first text record
15 (X'F')	8	PMARR_SCAT	Scatter load information (character)
15 (X'F')	2	PMARR_SLSZ	Scatter list length (unsigned)
17 (X'11')	2	PMARR_TTSZ	Translation table length (unsigned)
19 (X'13')	2	PMARR_ESDT	ESDID of first text block (character)
21 (X'15')	2	PMARR_ESDC	ESDID of EP control section (character)

Figure 97 (Page 2 of 2). Attributes Unique to Load Modules (PDS only). If PMAR_LFMT=OFF then this section follows the PMAR basic section.

Offset	Length or Bit Pattern	Name	Description
23 (X'17')	variable	PMARR_END	End of load module attributes (character)

Figure 98. Alias in Unformatted Form. Used only as input to the PUT function.

Offset	Length or Bit Pattern	Name	Description
00 (X'00')	7	PMARA	PMAR alias entry section (structure)
00 (X'00')	2	PMARA_LEN	Section length (unsigned)
2 (X'02')	5	PMARA_DATA	Section data (character)
2 (X'02')	4	PMARA_EPA	Entry point offset (unsigned)
6 (X'06')	1	PMARA_ATR	Attribute bytes (character)
6 (X'06')	1	PMARA_ATR1	First attribute byte (bitstring)
6 (X'06')	1	PMARA_FTB2	Alternative name for flags byte. These flags must be at the same offsets as the corresponding flags in PDS2FTB2 declared by macro IHAPDS. (bitstring)
	xxxx	-	Reserved
	.... 11..	PMARA_AMD	Alias entry addressing mode. If B'00', AMODE is 24. If B'10', AMODE is 31. If B'11', AMODE is ANY.
	xx	-	Reserved
7 (X'07')	variable	PMARA_END	End of alias entry section (character)

Parameters Related to the PUT Function

If the DESERV exit gets control for a DESERV put function invocation, DESX_DESP_PTR points to the DESERV parameter list. If the DESP field DESP_FUNC=X'04' (DESP_FUNC_PUT), this indicates a PUT function parameter list. Figure 99 shows the fields of the DESP that pertain to the DESERV PUT invocation.

Figure 99 (Page 1 of 3). DESERV PUT DESP Fields

Offset	Length or Bit Pattern	Name	Description
00 (X'00')	104	DESP	DE Services parameter list (structure)
00 (X'00')	16	DESP_HEADER	Standard header (character)

Figure 99 (Page 2 of 3). DESERV PUT DESP Fields

Offset	Length or Bit Pattern	Name	Description
00 (X'00')	8	DESP_ID	Eyecatcher 'IGWDESP' (character)
08 (X'08')	4	DESP_LEN	Length of DESP (signed)
	4	DESP_LEN_IV	Constant to be used for DESP_LEN
12 (X'0C')	1	DESP_LEV	Control block level (character)
	4	DESP_LEV_IV	Constant to be used for DESP_LEV
13 (X'0D')	3	-	Reserved
16 (X'10')	1	DESP_FUNC	Function type (GET=X'01', PUT=X'04', DELETE=X'07', RENAME=X'08', UPDATE=X'09') (unsigned)
	X'07'	DESP_FUNC_DELETE	Constant to be used for DESP_FUNC
	X'08'	DESP_FUNC_RENAME	Constant to be used for DESP_FUNC
	X'09'	DESP_FUNC_UPDATE	Constant to be used for DESP_FUNC
	X'04'	DESP_FUNC_PUT	Constant to be used for DESP_FUNC
	X'01'	DESP_FUNC_GET	Constant to be used for DESP_FUNC
	X'00'	DESP_FUNC_OMITTED	Constant to be used for DESP_FUNC
17 (X'11')	3	-	Reserved
20 (X'14')	4	-	Reserved
24 (X'18')	12	DESP_DATA	Function data (character)
24 (X'18')	2	-	Reserved
26 (X'1A')	1	-	Reserved
27 (X'1B')	1	-	Reserved
28 (X'1C')	1	DESP_LIBTYPE	Indicates whether a DCB or a DEB is input. (unsigned)
	X'02'	DESP_LIBTYPE_DCB	DCB input
	X'01'	DESP_LIBTYPE_DEB	DEB input
	X'00'	DESP_LIBTYPE_OMITTED	Omitted
29 (X'1D')	1	-	Reserved
30 (X'1E')	1	-	Reserved
31 (X'1F')	1	-	Reserved

Figure 99 (Page 3 of 3). DESERV PUT DESP Fields

Offset	Length or Bit Pattern	Name	Description
32 (X'20')	1	DESP_OPTION	REPLACE option (REPLACE=X'01', NOREPLACE=X'00') (unsigned)
	X'02'	DESP_OPTION_REPLACE_ALIAS	Constant to be used with DESP_OPTION
	X'01'	DESP_OPTION_REPLACE	Constant to be used with DESP_OPTION
	X'00'	DESP_OPTION_NOREPLACE	Constant to be used with DESP_OPTION
33 (X'21')	1	-	Reserved
34 (X'22')	1	-	Reserved
35 (X'23')	1	-	Reserved
36 (X'24')	4	DESP_DCB_PTR	DCB address
40 (X'28')	4	DESP_DEB_PTR	DEB address
44 (X'2C')	4	-	Reserved
48 (X'30')	4	-	Reserved
52 (X'34')	4	-	Reserved
56 (X'38')	4	-	Reserved
60 (X'3C')	4	-	Reserved
64 (X'40')	4	-	Reserved
68 (X'44')	4	-	Reserved
72 (X'48')	4	DESP_MEM_DATA_PTR	MEM_DATA_ADDRESS
76 (X'4C')	4	DESP_MEM_DATA2	MEM_DATA entry count (unsigned)
80 (X'50')	4	-	Reserved
84 (X'54')	4	-	Reserved
88 (X'58')	4	-	Reserved
92 (X'5C')	4	-	Reserved
96 (X'60')	4	-	Reserved
100 (X'64')	4	-	Reserved

PUT Return and Reason Codes

Figure 100 on page 275 describes the return and reason codes for the PUT function.

Return Code	Description	
DESRC_SUCC 0	Successful	
	Reason Code	**Description**
	DESRS_SUCC	X'00' Successful
DESRC_INFO 4	Not completely successful.	
	Reason Code	**Description**
	DESRS_NAME_NOT_DEFINED	X'400' Name to be replaced did not previously exist.
DESRC_WARN 8	Results questionable	
DESRC_PARM 12	Missing/invalid parameters	
	Reason Code	**Description**
	DESRS_PDS_NOT_SUPPORTED	X'415' This function requires a PDSE data set
	DESRS_INVALID_PUT_OPTION	X'429' The PUT function requires that the OPTION field be specified.
	DESRS_INVALID_MEM_DATA_PTR	X'428' The address of the MEM_DATA block is 0
	DESRS_INVALID_MEM_DATA_CNT	X'427' The count of entries in the MEM_DATA block is 0
	DESRS_DCB_NOT_OPEN_OUTPUT	X'41F' With function PUT the DCB must have been opened for output
	DESRS_INVALID_PARM_LIST_HEADER	X'411' The id, length, or level of the DESP is not valid
	DESRS_UNSUPPORTED_FUNC	X'424' The FUNC value is incorrect
	DESRS_DEB_REQUIRES_AUTH	X'423' To pass the DEB the caller must be supervisor state or a privileged key
	DESRS_INVALID_DCB_PTR	X'422' The address of the DCB is 0
	DESRS_DCB_NOT_OPEN	X'421' The passed DCB is not opened
	DESRS_INVALID_DEB_PTR	X'41E' Address of the DEB is 0 or DEB was input but the DCB pointed to by the DEB did not point back to the DEB
DESRC_CALR 16	Caller has a problem	
	Reason Code	**Description**
	DESRS_INVALID_CT	X'414' Connect token not valid.
	DESRS_INVALID_MLT	X'413' MLT is not valid.
	DESRS_MORE_THAN_1_PRIMARY	X'412' The MEM_DATA must have only one member designated as primary
	DESRS_INVALID_NAME_LENGTH	X'425' The length of an alias name was either 0 or
	DESRS_INVALID_NAME_PREFIX	X'410' The first 8 bytes of a name were all X'FF'
	DESRS_DATA_LENGTH_ERROR	X'43A' The DESD data length is not valid, data length must be greater than 0 and less than 108 bytes.
	DESRS_NO_PRIMARY_NAME	X'40F' The MEM_DATA must have one member designated as primary

	DESRS_NAME_ALREADY_EXISTS	X'40E' The PUT failed; name conflict
	DESRS_NAME_NOT_DEFINED	X'400' Name to be replaced did not previously exist
	DESRS_NAME_IS_PRIMARY_NAME	X'43C' The alias name specified is a primary name and the options did not allow for deleting primary name
	DESRS_PRI_NM_THIS_FILE	X'3FD' Alias name is same name as primary name for this member
	DESRS_DEBCHK_FAILED	X'41D' The DEBCHK macro failed, the DCB or DEB was not valid.
DESRC_ENVR 20	Resources unavailable	
DESRC_IOER 24	I/O error	
DESRC_MEDE 28	Media error	
DESRC_DSLE 32	Data Set logical error	
DESRC_SEVE 36	Severe error	
	Reason Code	**Description**
	DESRS_UNKNOWN	X'447' Issued by the DESERV recovery routine when entered for an unknown reason (i.e., prog chk).
	DESRS_ADD_STACK_FAILED	X'437' Non-zero return code from an IGWFESTK request
	DESRS_SETLOCK_ERR	X'407' Bad return code from SETLOCK

Figure 100. PUT Return and Reason Codes

The DESD (DESERV member data descriptor) is the input to the DESERV PUT function. The DESD is an array consisting of the number of entries defined by the DESP field DESP_MEM_DATA2. The DESP_MEM_DATA_PTR points to the DESD and the DESD CSECT of the IGWDES macro maps it. The DESD structure is shown in Figure 101.

Figure 101 (Page 1 of 2). DESD Parameter List

Offset	Length or Bit Pattern	Name	Description
00 (X'00')	16	DESD	Member data descriptor (structure)
00 (X'00')	16	DESD_ENTRY	Entry descriptor (character)
00 (X'00')	1	DESD_FLAG	Flags (bitstring)
	1...	DESD_FLAG_ALIAS	Alias entry

Figure 101 (Page 2 of 2). DESD Parameter List

Offset	Length or Bit Pattern	Name	Description
1 (X'01')	1	DESD_CODE	Processing code (error=X'02', not processed =X'01', successful =X'00')(unsigned)
	X'02'	DESD_CODE_ERROR	Constant to be used with DESD_CODE
	X'01'	DESD_CODE_NOGO	Constant to be used with DESD_CODE
	X'00'	DESD_CODE_SUCC	Constant to be used with DESD_CODE
2 (X'02')	2	DESD_ERRCODE	Error code (low order halfword of DESERV reason code if error) (unsigned)
4 (X'04')	2	-	Reserved
6 (X'06')	2	DESD_DATA_LEN	Length of data area (unsigned)
08 (X'08')	4	DESD_DATA_PTR	Address of data (address)
12 (X'0C')	4	DESD_NAME_PTR	Address of varying length name (address)

The DESD_NAME_PTR points to the DESN structure as defined above. The DESD_DATA_PTR points to the directory entry for the program object being saved in the PDSE directory. The format of the directory entry is different depending on whether the DESD entry represents the primary name or an alias name. For the primary name, the DESD_DATA_PTR points to the CSECT PMAR mapped by IEWPMAR. For an alias name, the DESD_DATA_PTR points to the CSECT PMARA mapped by the macro IEWPMAR. Figure 98 on page 272 shows the PMARA structure. The DESD_FLAG_ALIAS identifies the entry in the DESD as a primary or an alias.

The DESERV exit is passed to the current return and reason code that is to be passed back to the DESERV caller. The exit (either the pre-processing or the post-processing) can cause DESERV to return a different return and reason code to the DESERV caller by returning with a return code of 4 in register 15. If the exit returns control to DESERV with a return code of 4 in register 15, the (possibly modified) values of DESX_RETURN_CODE and DESX_REASON_CODE are returned to the caller of DESERV. If the pre-processing exit returns a return code of 0 in register 15, processing continues. Whereas if the post-processing exit returns with a return code of 0 in register 15, the original values of DESX_RETURN_CODE and DESX_REASON_CODE (i.e. those that were passed as input to the exit) are returned to the DESERV caller.

The exit processing can include interrogation of the DESERV parameter list (DESP) or interrogation or modification of the other interface structures and buffers. The DESP will not be modified by the DESERV exit. The exit is passed the caller key DESP, but DESERV has already made a key 5 copy of the DESP that is used for DESERV processing. Therefore modifications to the caller key DESP would not influence the DESERV processing. The DESERV interface structures are defined in

the macro IGWDES. The macros IGWSMDE and IEWPMAR define the directory entry format.

Parameters Related to the DELETE Function

If the DESERV exit gets control for a DESERV DELETE function invocation, DESX_DESP_PTR points to the DESERV parameter list. If the DESP field DESP_FUNC=X'07' (DESP_FUNC_DELETE), this indicates a DELETE function parameter list. Figure 102 shows the fields of the DESP that pertain to the DESERV DELETE invocation.

Figure 102 (Page 1 of 2). DESERV DELETE DESP Fields

Offset	Length or Bit Pattern	Name	Description
00 (X'00')	104	DESP	DE Services parameter list (structure)
00 (X'00')	16	DESP_HEADER	Standard header (character)
00 (X'00')	8	DESP_ID	Eyecatcher IGWDESP (character)
08 (X'08')	4	DESP_LEN	Length of DESP (signed)
	X'04'	DESP_LEN_IV	Constant to be used with DESP_LEN
12 (X'0C')	1	DESP_LEV	Control block level (character)
	X'04'	DESP_LEV_IV	Constant to be used with DESP_LEV
13 (X'0D')	3	-	Reserved
16 (X'10')	1	DESP_FUNC	Function type (GET=X'01', PUT=X'04', DELETE=X'07', RENAME=X'08', UPDATE=X'09')
	X'09'	DESP_FUNC_UPDATE	Constant to be used with DESP_FUNC
	X'08'	DESP_FUNC_RENAME	Constant to be used with DESP_FUNC
	X'07'	DESP_FUNC_DELETE	Constant to be used with DESP_FUNC
	X'04'	DESP_FUNC_PUT	Constant to be used with DESP_FUNC
	X'01'	DESP_FUNC_GET	Constant to be used with DESP_FUNC
	X'00'	DESP_FUNC_OMITTED	Constant to be used with DESP_FUNC
17 (X'11')	3	-	Reserved
20 (X'14')	4	-	Reserved
24 (X'18')	12	DESP_DATA	Function data (character)
24 (X'18')	2	-	Reserved

Figure 102 (Page 2 of 2). DESERV DELETE DESP Fields

Offset	Length or Bit Pattern	Name	Description
26 (X'1A')	1	-	Reserved
27 (X'1B')	1	-	Reserved
28 (X'1C')	1	DESP_LIBTYPE	Indicates whether a DCB or a DEB in input. (DCB input X'02', DEB input X'01') (unsigned)
	X'02'	DESP_LIBTYPE_DCB	Constant to be used with DESP_LIBTYPE
	X'01'	DESP_LIBTYPE_DEB	Constant to be used with DESP_LIBTYPE
	X'00'	DESP_LIBTYPE_OMITTED	Constant to be used with DESP_LIBTYPE
29 (X'1D')	1	-	Reserved
30 (X'1E')	1	-	Reserved
31 (X'1F')	1	-	Reserved
32 (X'20')	1	-	Reserved
33 (X'21')	1	-	Reserved
34 (X'22')	1	-	Reserved
35 (X'23')	1	-	Reserved
36 (X'24')	4	DESP_DCB_PTR	DCB address
40 (X'28')	4	DESP_DEB_PTR	DEB address
44 (X'2C')	4	-	Reserved
48 (X'30')	4	-	Reserved
52 (X'34')	4	-	Reserved
56 (X'38')	4	-	Reserved
60 (X'3C')	4	-	Reserved
64 (X'40')	4	-	Reserved
68 (X'44')	4	-	Reserved
72 (X'48')	4	-	Reserved
76 (X'4C')	4	-	Reserved
80 (X'50')	4	DESP_NAME_LIST	List of names to be deleted (DESL) (address)
84 (X'54')	4	DESP_NAME_LIST2	Number of entries in DESL name list (unsigned)
88 (X'58')	4	-	Reserved
92 (X'5C')	4	-	Reserved
96 (X'60')	4	-	Reserved
100 (X'64')	4	-	Reserved

The DESL points to the names to be deleted. The DESP_NAME_LIST_PTR points to the DESL and the DESL DSECT in the IGWDES macro maps it. A DESL is an

array consisting of the number of entries the DESP field DESP_NAME_LIST2 defines. The DESL structure is shown in Figure 87 on page 260.

Parameters Related to the RENAME Function

If the DESERV exit gets control for a DESERV RENAME function invocation, DESX_DESP_PTR points to the DESERV parameter list. If the DESP field DESP_FUNC=X'08' (DESP_FUNC_RENAME), this indicates a RENAME function parameter list. Figure 103 shows the fields of the DESP that pertain to the DESERV RENAME invocation.

Figure 103 (Page 1 of 2). DESERV RENAME DESP Fields

Offset	Length or Bit Pattern	Name	Description
00 (X'00')	104	DESP	DE Services parameter list (structure)
00 (X'00')	16	DESP_HEADER	Standard header (character)
00 (X'00')	8	DESP_ID	Eyecatcher IGWDESP (character)
08 (X'08')	4	DESP_LEN	Length of DESP (signed)
	X'04'	DESP_LEN_IV	Constant to be used with DESP_LEN
12 (X'0C')	1	DESP_LEV	Control block level (character)
	X'04'	DESP_LEV_IV	Constant to be used with DESP_LEV
13 (X'0D')	3	-	Reserved
16 (X'10')	1	DESP_FUNC	Function type (GET=X'01', PUT=X'04', DELETE=X'07', RENAME=X'08', UPDATE=X'09')
	X'09'	DESP_FUNC_UPDATE	Constant to be used with DESP_FUNC
	X'08'	DESP_FUNC_RENAME	Constant to be used with DESP_FUNC
	X'07'	DESP_FUNC_DELETE	Constant to be used with DESP_FUNC
	X'04'	DESP_FUNC_PUT	Constant to be used with DESP_FUNC
	X'01'	DESP_FUNC_GET	Constant to be used with DESP_FUNC
	X'00'	DESP_FUNC_OMITTED	Constant to be used with DESP_FUNC
17 (X'11')	3	-	Reserved
20 (X'14')	4	-	Reserved
24 (X'18')	12	DESP_DATA	Function data (character)
24 (X'18')	2	-	Reserved

Figure 103 (Page 2 of 2). DESERV RENAME DESP Fields

Offset	Length or Bit Pattern	Name	Description
26 (X'1A')	1	-	Reserved
27 (X'1B')	1	-	Reserved
28 (X'1C')	1	DESP_LIBTYPE	Indicates whether a DCB or a DEB is input. (DCB,DEB) (DCB input X'02', DEB input X'01') (unsigned)
	X'02'	DESP_LIBTYPE_DCB	Constant to be used with DESP_LIBTYPE
	X'01'	DESP_LIBTYPE_DEB	Constant to be used with DESP_LIBTYPE
	X'00'	DESP_LIBTYPE_OMITTED	Constant to be used with DESP_LIBTYPE
29 (X'1D')	1	-	Reserved
30 (X'1E')	1	-	Reserved
31 (X'1F')	1	-	Reserved
32 (X'20')	1	-	Reserved
33 (X'21')	1	-	Reserved
34 (X'22')	1	-	Reserved
35 (X'23')	1	-	Reserved
36 (X'24')	4	DESP_DCB_PTR	DCB address
40 (X'28')	4	DESP_DEB_PTR	DEB address
44 (X'2C')	4	-	Reserved
48 (X'30')	4	-	Reserved
52 (X'34')	4	-	Reserved
56 (X'38')	4	-	Reserved
60 (X'3C')	4	-	Reserved
64 (X'40')	4	-	Reserved
68 (X'44')	4	-	Reserved
72 (X'48')	4	-	Reserved
76 (X'4C')	4	-	Reserved
80 (X'50')	4	DESP_NAME_LIST	List of pairs of names for rename operations (DESL) (address)
84 (X'54')	4	DESP_NAME_LIST2	Number of entries in DESL name list (unsigned)
88 (X'58')	4	-	Reserved
92 (X'5C')	4	-	Reserved
96 (X'60')	4	-	Reserved
100 (X'64')	4	-	Reserved

The DESL points to the names that are to be renamed and to the new names. The DESP_NAME_LIST_PTR points to the DESL and the DESL DSECT in the IGWDES macro maps it. A DESL is an array consisting of the number of entries the DESP field DESP_NAME_LIST2 defines. The DESL structure is shown in Figure 87 on page 260.

Parameters Related to the UPDATE Function

If the DESERV exit gets control for a DESERV UPDATE function invocation, DESX_DESP_PTR points to the DESERV parameter list. If the DESP field DESP_FUNC=X'09' (DESP_FUNC_UPDATE), this indicates an UPDATE function parameter list. Figure 104 shows the fields of the DESP that pertain to the DESERV UPDATE invocation.

Figure 104 (Page 1 of 3). DESERV UPDATE DESP Fields

Offset	Length or Bit Pattern	Name	Description
00 (X'00')	104	DESP	DE Services parameter list (structure)
00 (X'00')	16	DESP_HEADER	Standard header (character)
00 (X'00')	8	DESP_ID	Eyecatcher 'IGWDESP' (character)
08 (X'08')	4	DESP_LEN	Length of DESP (signed)
	X'04'	DESP_LEN_IV	Constant to be used with DESP_LEN
12 (X'0C')	1	DESP_LEV	Control block level (character)
	X'04'	DESP_LEV_IV	Constant to be used with DESP_LEV
13 (X'0D')	3	-	Reserved
16 (X'10')	1	DESP_FUNC	Function type (GET=X'01', PUT=X'04', DELETE=X'07', RENAME=X'08', UPDATE=X'09')
	X'09'	DESP_FUNC_UPDATE	Constant to be used with DESP_FUNC
	X'08'	DESP_FUNC_RENAME	Constant to be used with DESP_FUNC
	X'07'	DESP_FUNC_DELETE	Constant to be used with DESP_FUNC
	X'04'	DESP_FUNC_PUT	Constant to be used with DESP_FUNC
	X'01'	DESP_FUNC_GET	Constant to be used with DESP_FUNC
	X'00'	DESP_FUNC_OMITTED	Constant to be used with DESP_FUNC
17 (X'11')	3	-	Reserved
20 (X'14')	4	-	Reserved

Figure 104 (Page 2 of 3). DESERV UPDATE DESP Fields

Offset	Length or Bit Pattern	Name	Description
24 (X'18')	12	DESP_DATA	Function data (character)
24 (X'18')	2	-	Reserved
26 (X'1A')	1	-	Reserved
27 (X'1B')	1	-	Reserved
28 (X'1C')	1	DESP_LIBTYPE	Function subtype (DCB,DEB) (DCB input X'02', DEB input X'01') (unsigned)
	X'02'	DESP_LIBTYPE_DCB	Constant to be used with DESP_LIBTYPE
	X'01'	DESP_LIBTYPE_DEB	Constant to be used with DESP_LIBTYPE
	X'00'	DESP_LIBTYPE_OMITTED	Constant to be used with DESP_LIBTYPE
29 (X'1D')	1	-	Reserved
30 (X'1E')	1	-	Reserved
31 (X'1F')	1	-	Reserved
32 (X'20')	1	-	Reserved
33 (X'21')	1	-	Reserved
34 (X'22')	1	-	Reserved
35 (X'23')	1	-	Reserved
36 (X'24')	4	DESP_DCB_PTR	DCB address
40 (X'28')	4	DESP_DEB_PTR	DEB address
44 (X'2C')	4	-	Reserved
48 (X'30')	4	-	Reserved
52 (X'34')	4	-	Reserved
56 (X'38')	4	-	Reserved
60 (X'3C')	4	-	Reserved
64 (X'40')	4	-	Reserved
68 (X'44')	4	-	Reserved
72 (X'48')	4	-	Reserved
76 (X'4C')	4	-	Reserved
80 (X'50')	4	DESP_NAME_LIST	List of SMDEs with imbedded PMAR (and PMARL) to update the directory information (DESL) (address)
84 (X'54')	4	DESP_NAME_LIST2	Number of entries in DESL name list (unsigned)
88 (X'58')	4	-	Reserved
92 (X'5C')	4	-	Reserved
96 (X'60')	4	-	Reserved

Figure 104 (Page 3 of 3). DESERV UPDATE DESP Fields

Offset	Length or Bit Pattern	Name	Description
100 (X'64')	4	-	Reserved

The DESL points to the SMDEs which are to be updated. The DESP_NAME_LIST_PTR points to the DESL and the DESL DSECT in the IGWDES macro maps it. A DESL is an array consisting of the number of entries the DESP field DESP_NAME_LIST2 defines. The DESL structure is shown in Figure 87 on page 260.

Entry Environment for Exit Routine

Interrupts: enabled

State and Key: supervisor state and key 0

ASC Mode: P=H=S

AMODE, RMODE: AMODE=31, RMODE=ANY

LOCKS: no locks held

Registers:

0	unpredictable
1	address of DESERV EXIT parameter list mapped by DSECT DESX in IGWDES.
2-12	unpredictable
13	eighteen word save area (key 0).
14	return address
15	entry point address of exit routine

The exit is entered in TASK mode and the DESERV recovery environment is an ESTAE, therefore the exit routine can issue SVCs if required.

Exit Environment for Exit routine

Interrupts: enabled

State and Key: supervisor state and key 0

ASC Mode: P=H=S

AMODE, RMODE: AMODE=31, RMODE=ANY

LOCKS: no locks held

Registers:

Unless otherwise specified, all registers must be restored to their contents on entry.

15 return code

For pre-processing exit

R15 = 0 Continue processing of this DESERV call.

R15 = 4 Discontinue processing of this DESERV call. Control is immediately returned to the caller of DESERV with the return and reason codes as set by the exit in the DESX fields DESX_RETURN_CODE and DESX_REASON_CODE.

Additional Return and Reason Codes: For the GET, RENAME, DELETE, and UPDATE return and reason codes, refer to *DFSMS/MVS Macro Instructions for Data Sets* .

For post-processing exit

R15=0 Continue processing of this DESERV call, i.e. return to the caller of DESERV.

R15 = 4 Control is returned to the caller of DESERV with the return and reason codes as set by the exit in the DESX fields DESX_RETURN_CODE and DESX_REASON_CODE. Values for DESX_RETURN_CODE and DESX_REASON_CODE are described in the macro IGWDES. The reason code structure is such that the first two bytes are system component id and module id (i.e. system diagnostic information). The low order two bytes contain the real reason code as indicated in the macro IGWDES.

Additional Return and Reason Codes: For the GET, RENAME, DELETE, and UPDATE return and reason codes, refer to *DFSMS/MVS Macro Instructions for Data Sets* .

Note: Any other return code from the exits causes DESERV code to take an SVC dump.

Registers on Entry to the DESERV Exit

When your DESERV exit gets control, the general-purpose registers have the following content:

Register	Contents
0	not applicable
1	address of DESX
2-12	not applicable
13	address of register save area
14	return address
15	address of DESERV exit entry point

Registers on Return from the DESERV Exit

When you return control to DESERV, the register contents must be set up as follows:

Register	Contents
0-14	restored to contents at entry
15	return code

DESERV Exit Return and Reason Codes

Return Code	Description
00 (X'00')	Continue with DESERV function
04 (X'04')	Return immediately to the DESERV caller and return the return and reason codes as defined by DESX_RETURN_CODE and DESX_REASON_CODE respectively.

DESERV FUNC=EXIT Return and Reason Codes

The formats of the return and reason codes are:

Offset/length	Description
00 (X'00') 1 byte	SMS Component code (X'27') indicates Common Adaptor (of which DESERV is a part)
01 (X'01') 1 byte	Module ID - used for problem diagnosis
02 (X'02') 2 bytes	Reason code - identifies the error. A program testing the DESERV reason code should only look at these last two bytes. The component id and module id should not be tested. They are reported for diagnostic purposes only.

The following are the two low order byte values for the reason codes that DESERV FUNC=EXIT might return (sorted by return code).

Return Code	Description	
DESRC_SUCC	X'00' Successful processing	
	Reason Code	**Description**
	DESRS_SUCC	X'00' Successful processing
DESRC_INFO 4	Not completely successful	
DESRC_WARN 8	Results questionable	
	Reason Code	**Description**
	DESRS_DST_ALREADY_EXISTS	X'44B' An EXIT exists and DESERV FUNC EXIT with NOREPLACE specified was issued. The current exit is not replaced
	DESRS_DST_COMP_SWAP_FAILED	X'451' An EXIT_OPTION=DELETE specified a DST address that was not current. The compare and swap failed.
DESRC_PARM 12	Missing or invalid parameters	
	Reason Code	**Description**
	DESRS_EXIT_OPTION_INVALID	X'44F' The EXIT_OPTION specified is not supported
	DESRS_EXIT_SCOPT_INVALID	X'45' The EXIT_SCOPE specified is not supported
	DESRS_EXIT_DST_PTR_ZERO	X'44C' The caller of DESERV FUNC=EXIT supplied a DST address of zero via the EXIT_DST parameter
	DESRS_INVAL_DST_HEADER	X'44D' The DST header is not correct
	DESRS_INVAL_PREVDST_HEADER	X'453' The DST header is not valid for the DST pointed to by DESP_PREV_DSTPTR_PTR. This is checked for EXIT_OPTION=DELETE.
	DESRS_PREV_DSTPTR_PTR_ZERO	X'452' The pointer to the previous DST is zero. This is checked for EXIT_OPTION=DELETE.
	DESRS_INVALID_PARM_LIST_HEADER	X'411' The id, length, or level of the DESP is not valid
	DESRS_UNSUPPORTED_FUNC	X'424' The FUNC value is incorrect
	DESRS_DEB_REQUIRES_AUTH	X'423' To pass the DEB the caller must be in supervisor state or a privileged key
	DESRS_INVALID_DCB_PTR	X'422' The address of the DCB is 0
	DESRS_DCB_NOT_OPEN	X'421' The passed DCB is not opened
	DESRS_INVALID_DEB_PTR	X'41E' Address of the DEB is 0 or DEB was input but the DCB pointed to by the DEB did not point back to the DEB
DESRC_CALR 16	Caller has a problem	
	Reason Code	**Description**
	DESRS_DEBCHK_FAILED	X'41D' The DEBCHK macro failed. The DCB or DEB was not valid

	DESRS_AUTH_ERROR	X'449' Caller not supervisor state or system key
DESRC_ENVR 20	Resources unavailable	
DESRC_IOER 24	I/O error	
DESRC_MEDE 28	Media error	
DESRC_DSLE 32	Data set logical error	
DESRC_SEVE 36	Severe error	
	Reason Code	**Description**
	DESRS_UNKNOWN	X'447' Issued by the DESERV recovery routine when entered for an unknown reason (i.e., prog chk).
	DESRS_ADD_STACK_FAILED	X'437' Non-zero return code from an IGWFESTK request
	DESRS_SETLOCK_ERR	X'407' Bad return code from SETLOCK
	DESRS_EXTRACT_ERROR	X'406' IGWFTOKM EXTRACT failed
	DESRS_SET_ERROR	X'405' IGWFTOKM SET failed.

Figure 105. Return and Reason Codes for the Exit DESERV Function

Additional Return and Reason Codes: For the GET, RENAME, DELETE, and UPDATE return and reason codes, refer to *DFSMS/MVS Macro Instructions for Data Sets* .

Example of the DESERV Exit

The following program segment establishes and deletes the task mode DESERV exit and thereby restores the previous task. The sample is generic enough to apply to either the global or the task level exit but shows here the task level support. When establishing a global exit, obtain the DST storage in common storage.

```
SAMPLE   CSECT
       USING  *,12
       STM    14,12,12(13)      SAVE REGISTERS
       LR     12,15             ESTABLISH BASE REGISTER
       LA     2,SAVE            ADDRESS REGISTER SAVE AREA
       ST     2,8(13)           FORWARD CHAIN SAVE AREA
       ST     13,SAVE+4         BACKWARD CHAIN SAVE AREA
       LR     13,2              ESTABLISH SAVE AREA
       .
       .
       .
*
*
* ESTABLISH THE TASK LEVEL EXIT
*
*     BUILD THE DST TO REPRESENT MY TASK LEVEL EXIT
*
       LA     3,MY_DST             ADDRESSABILITY TO MY DST
       USING  DST,3                MAP DST
       XC     DST,DST              CLEAR MY DST STORAGE
       MVC    DST_ID,DST_ID_CONST  SET EYECATCHER IN DST
       LA     15,DST_LEN_IV(,0)    GET LENGTH OF DST
       ST     15,DST_LEN           SET LENGTH OF DST
       MVI    DST_LEV,DST_LEV_IV   SET LEVEL OF DST USED
       OI     DST_FLAGS,DST_FLAGS_PROP  SET FLAG TO PROPAGATE THUS      EXIT
*                                  TO ATTACHED TASKS.
       MVC    DST_EXIT_PTR,ADDR_DESEXIT  SET ADDRESS OF TASK EXIT
*
*     CALL DESERV TO ENABLE MY TASK LEVEL EXIT
*        THIS MODULE ASSUMES THAT IT IS RUNNING EITHER SUPERVISOR S     TATE
*        OR SYSTEM KEY.
*
       DESERV FUNC=EXIT,                                                 *
              EXIT_SCOPE=TASK,                                           *
              EXIT_OPTION=REPLACE,                                       *
              EXIT_DST=MY_DST,                                           *
              EXIT_PREV_DSTPTR=MY_PREV_DSTPTR,                           *
              MF=S
       .
       .
       .
*
* QUERY THE CURRENT DST ADDRESS, RETURNED IN CURRENT_DSTPTR.
* THERE IS NO NEED TO QUERY PRIOR TO DOING THE DELETE, THIS IS
* HERE JUST TO SHOW THE INVOCATION SYNTAX.
*
       DESERV FUNC=EXIT,                                             *
              EXIT_SCOPE=TASK,                                       *
              EXIT_OPTION=QUERY,                                     *
              EXIT_PREV_DSTPTR=CURRENT_DSTPTR,                       *
              MF=S
```

Figure 106 (Part 1 of 2). Establishing and Deleting a Task Level DESERV Exit

```
*
*    HERE THE APPLICATION WOULD DO SOMETHING TO
*    CAUSE A DESERV CALL TO BE DONE. AN EXAMPLE OF THIS TYPE OF THING
*    WOULD BE ATTACHING THE BINDER AS A SUBTASK (THE BINDER USES
*    DESERV.)
*
       .
       .
       .
*
* DELETE THE TASK LEVEL EXIT
*
       DESERV FUNC=EXIT,                                              *
              EXIT_SCOPE=TASK,                                        *
              EXIT_OPTION=DELETE,                                     *
              EXIT_DST=MY_DST,                                        *
              EXIT_PREV_DSTPTR=MY_PREV_DSTPTR,                        *
              MF=S
       .
       .
       .
*
* RESTORE REGISTERS AND RETURN
       L      13,SAVE+4
       LM     14,12,12(13)
       SR     15,15          SET RETURN CODE
       BR     14             RETURN TO CALLER
* THE FOLLOWING IS A BLOCK USED BY THIS APPLICATION. THIS BLOCK CAN
* BE USED BY THE EXIT. THE EXIT ALSO (SMARTLY) FINDS THE
* PREV_DST_PTR FIELD IN MY_BLOCK WHEN IT GIVES CONTROL TO THE
* PREVIOUSLY ESTABLISHED TASK LEVEL EXIT.
MY_BLOCK        DS  0D             MY APPLICATION BLOCK
MY_APPLICATION_STUFF DS  CL8       MY APPLICATION STUFF
MY_DST          DS  CL(DST_LEN_IV) MY DST IMBEDDED IN MY BLOCK
MY_PREV_DSTPTR  DS  F              ADDRESS OF PREVIOUS TASK LEVEL
                EXTRN   DESEXIT
ADDR_DESEXIT    DC  A(DESEXIT)     ADDRESS OF MY TASK LEVEL EXIT
DST_ID_CONST    DC  CL8'IGWDST  '  DST EYECATCHER
*                                   DST
CURRENT_DSTPTR  DS  F              ADDRESS OF CURRENT TASK LEVEL
*                                   DST
SAVE            DS  18F            REGISTER SAVE AREA
                END
```

Figure 106 (Part 2 of 2). Establishing and Deleting a Task Level DESERV Exit

The following program segment shows a sample DESERV EXIT. This is the exit established by the previous segment of code. This sample shows how to pass control to a previous exit.

```
DESEXIT CSECT
DESEXIT AMODE  31                Must be AMODE 31
DESEXIT RMODE  ANY               Could be RMODE 24 if required
*
*  entry code to save registers and establish base register.
        USING  *,12
        STM    14,12,12(13)      SAVE REGISTERS
        LR     12,15             ESTABLISH BASE REGISTER
        LA     2,SAVE            ADDRESS REGISTER SAVE AREA
        ST     2,8(13)           FORWARD CHAIN SAVE AREA
        ST     13,SAVE+4         BACKWARD CHAIN SAVE AREA
        LR     13,2              ESTABLISH SAVE AREA
*
* Assume this exit is only interested in the output from GET functions.
* Therefore ignore entry for all other functions and only
* process the get function invocations where get processing is complete
*
        SLR    2,2               clear reg
        ST     2,EXIT_RC         initialize return code
        LR     2,1               get the DESX address in reg 2
        USING  DESX,2            map the DESX
        L      3,DESX_DST_PTR    get address of DST
        LR     4,3               get address of DST
        LA     5,MY_DST-MY_BLOCK (,0) get offset to DST within MY_BLOCK
        SR     4,5               get the address of MY_BLOCK
        USING  MY_BLOCK,4        map MY_BLOCK
*
* First give control to any previously established DESERV
* exits. If the value in PREV_DST_PTR is zero, then there was no
* previous DST (i.e. no previous DESERV exit). PREV_DST_PTR was
* saved in MY_BLOCK by the SAMPLE CSECT that enabled this
* exit.
        SR     15,15             simulate previous exit's return code
        ICM    5,15,PREV_DST_PTR get previous DST address
        BZ     NOPREVDST         branch if zero, no previous DST
* There was a previous exit, to which you transfer control.
* First build a DESX then branch to the previous exit.
*
*
* Getmain dynamic storage for interfacing with the other exit
*
*
        GETMAIN RU,LV=DESX_LEN_IV,SP=230,KEY=0,LOC=(ANY)
        ST     1,MY_DESX_STG_PTR
        MVC    0(L'DESX,1),DESX  copy the DESX that was input to this
*                                 routine.
        ST     5,DESX_DST_PTR-DESX(,1) Set previous DST address in DESX
        L      15,DST_EXIT_PTR-DST(,5)  get address of previous exit
        BALR   14,15             call the previous exit in 31 bit amode
```

Figure 107 (Part 1 of 3). Sample DESERV Exit Routine

```
NOPREVDST EQU  *
        ST     15,EXIT_RC         save previous exit's return code
* If this invocation of the exit was the pre-processing exit and the
* exit we called just returned with a return code of 4 then they
* might have returned the data in question.
        TM     DESX_FLAGS,DESX_POST_BIT is this post-processing call
        BO     POST               yes, branch
* this is a pre-processing call, did the exit just called request an
* immediate return, and therefore maybe fill in the output fields
* if the return code was zero, the called routine wants deserv to
* perform the function.
        CLC    EXIT_RC,ZERO       was return code zero
        BE     RETURN             yes, branch to return to deserv
POST    EQU    *                  either called exit performed the
*                                  deserv function, or this is a post
*                                  processing exit call.
        L      6,DESX_DESP_PTR    get the DESERV parameter list (DESP)
        USING  DESP,6             map DESP
        CLI    DESP_FUNC,DESP_FUNC_GET is this a function GET call
        BNE    RETURN             no, branch
        CLC    DESX_REASON_CODE+2(2),=AL2(DESRS_SUCC) check low order
*                                  halfword of reason code for success
        BE     GOODDATA           branch if DESERV GET was successful
        CLC    DESX_REASON_CODE+2(2),=AL2(DESRS_NOTFOUND) check for
*                                  some members not found
        BNE    BADDATA            branch if some other error
GOODDATA EQU   *
*
* Process the entries in the names list that were found
*
        .
        .
        .
        B      RETURN             return to the caller (DESERV)
BADDATA EQU    *
        .
        .
        .
*
RETURN  EQU    *                   Return to caller (DESERV)
*
*  restore registers, FREEMAIN storage, etc.
*
        L      15,EXIT_RC          set return code
        L      13,SAVE+4
        L      14,12(13)           restore return address
        LM     0,12,20(13)         restore callers other registers
        BR     14                  return to deserv
```

Figure 107 (Part 2 of 3). Sample DESERV Exit Routine

```
*
* CONTROL INFORMATION
*
SAVE            DS  18F            REGISTER SAVE AREA
ZERO            DC  F'0'           constant of zero
EXIT_RC         DC  F'0'           Return code to pass back to DESERV
DESX_ID_CONST   DC  CL8'IGWDESX '  EYECATCHER FOR DESX
MY_DESX_STG_PTR DC  F'0'           ADDRESS OF GETMAINED STORAGE FOR
*                                   DESX TO PASS TO PREVIOUS EXIT.
MY_BLOCK        DSECT              MY APPLICATION BLOCK
MY_APPLICATION_STUFF DS CL8        MY APPLICATION STUFF
MY_DST          DS  CL(DST_LEN_IV) MY DST IMBEDED IN MY BLOCK
PREV_DST_PTR    DS  F              ADDRESS OF PREVIOUS TASK LEVEL EXIT
                IGWDES             DESERV MAPPINGS
                IGWSMDE            DIRECTORY ENTRY
                IEWPMAR            PROGRAM MANAGEMENT ATTRIBUTE RECORD
*                                  THE PMAR IS A SUB RECORD OF THE SMDE
                END
```

Figure 107 (Part 3 of 3). Sample DESERV Exit Routine

Chapter 12. Maintaining the System Image Library

This chapter describes how to maintain the system image library, SYS1.IMAGELIB. SYS1.IMAGELIB is a partitioned data set (a PDSE is not supported) containing universal character set (UCS), forms control buffer (FCB), and printer control information for DFSMSdfp–supported IBM printers in the following forms, depending on the type of printer:

- UCS images (also called modules)
- UCS image tables
- FCB images (also called modules)
- Control modules.

The system uses UCS images and image tables to relate a user-requested UCS to the corresponding print band/train. Most IBM standard UCS images are included in SYS1.IMAGELIB during system installation. Note that when you install a new release of DFSMS/MVS it replaces images that have the IBM-supplied names. If you modified or replaced any of these you might want to maintain a separate copy of them. The following table shows the standard character set images for the IBM 1403, 3203, and 3211 printers:

Printer	Images
1403 or 3203	AN, HN, PCAN, PCHN, PN, QN, QNC, RN, SN, TN, XN, YN
3211	A11, G11, H11, P11, T11

For detailed, printer-specific information, or to determine which print bands/trains are available, see the publications listed in the following table:

Publication Title	Contents
IBM 2821 Control Unit Component Description	Information on creating a user-designed chain/train for the 1403 Printer.
IBM 3203 Printer Component Description and Operator's Guide	Information on creating a user-designed train for the 3203 Printer.
IBM 3211 Printer, 3216 Interchangeable Train Cartridge, and 3811 Printer Control Unit Component Description and Operator's Guide	Information on creating a user-designed train for the 3211 Printer.
IBM 3262 Model 5 Printer Product Description	Information on band IDs for the 3262 Model 5 Printer.
IBM 3800 Printing Subsystem Programmer's Guide	Information on planning for, conversion to, and use of the IBM 3800 Model 1 Printing Subsystem.
IBM 3800 Printing Subsystem Model 3 Programmer's Guide: Compatibility	Information on planning for, conversion to, and use of the IBM 3800 Model 3, 6, and 8 Printing Subsystem.
IBM 4245 Printer Model 1 Component Description and Operator's Guide	Information on band IDs for the 4245 Printer.
IBM 4248 Printer Model 1 Description	Information on band IDs for the 4248 Printer.

© Copyright IBM Corp. 1979, 1999

Publication Title	Contents
IBM 6262 Printer Print Band Manual	Information on band IDs for the 6262 Printer.
IBM 6262 Printer Model 014 User's Guide	Information on operating and maintaining the 6262 Model 14 Printer.

To use the information in this chapter, you should be familiar with the publications listed in the following table:

Publication Title	Contents
DFSMS/MVS Macro Instructions for Data Sets	Describes the SETPRT macro. You can use it to specify the images (modules) that you want.
DFSMS/MVS Utilities	Describes the IEBIMAGE utility program.
OS/390 MVS JCL Reference	Describes the CHARS, MODIFY, UCB, and FCB parameters of the DD statement, that are processed at OPEN.
OS/390 JES2 Initialization and Tuning Guide	Contains JES2 reference information.
OS/390 JES3 Initialization and Tuning Guide	Contains JES3 reference information.

Figure 108 provides information about the printer-specific contents of SYS1.IMAGELIB.

Figure 108. SYS1.IMAGELIB Contents

Printer Type	UCS Image	UCS Image Table [8]	FCB Image	Control Module
1403	X			
3203 [1]	X		X	
3211	X		X	
3262 Model 5 [2,4,5]		X	X	
4245 [3]		X[6]	X	
4248 [3]		X[6]	X	
6262 Model 14 [2,4,5]		X	X	
3800				X [7]

Notes to Figure 108:

1. The IBM 3203 Model 5 Printer is treated as a 3211 Printer by JES, except that the 3203 Model 5 does not support the 3211 indexing feature and ignores any indexing commands from JES. The 3203 Model 5 uses its own unique UCS images, but uses 3211 FCB images.
2. You can operate this printer as if it were a 4248 operating in 4248 native mode. In this case, use the 4248-related UCS sections of this chapter. The UCS information is contained in image tables.

3. You can operate this printer in native mode, in which case the UCS information is contained in image tables. You can also operate these printers in 3211 compatibility mode, in which case the UCS information is contained in UCS images; you should then use the 3211-related UCS sections of this chapter.
4. This printer uses the same image table as the 4248 printer. However, it does not support variable printer speeds or the horizontal copy feature and host stacker controls of the 4248 printer.
5. This printer uses the same FCB image as the 4248 Printer.
6. The image table is supplied by IBM. The contents of the 4245 image table are shown in Figure 113 on page 303. The contents of the 4248 image table are shown in Figure 114 on page 304.
7. 3800 Printing Subsystem control modules exist for:
   - Character arrangement tables
   - Graphics character modification tables
   - Copy modification tables
   - Library character sets
   - FCB images.

   You can use the IEBIMAGE utility program to create and maintain these control modules. See *DFSMS/MVS Utilities* for details about IEBIMAGE. IEBIMAGE and SYS1.IMAGELIB are not used when the printer is running in page mode. In that case, PSF for OS/390 uses other types of modules from other libraries. See *3900 Product Description* for more information.
8. Figure 112 on page 303 defines and describes the structure of a UCS image table entry.

UCS Images in SYS1.IMAGELIB

This section applies to the IBM 1403, 3203, and 3211 printers. SYS1.IMAGELIB contains UCS images for these printers. You can use the assembler and linkage editor to add a UCS image to those that reside in SYS1.IMAGELIB. The assembler does not generate executable code. It merely prepares DC statements, and the linkage editor puts them into SYS1.IMAGELIB. Observe the following rules when creating a new UCS image:

1. The member name must be 5 to 8 characters long; the first 4 characters must be the appropriate UCS prefix, as follows:

Prefix	Meaning
UCS1	1403 printer
UCS2	3211 printer (or 3211-compatible printer)
UCS3	3203 printer

   These first four characters must be followed by a character set code, one to four characters long. Any valid combination of letters and numbers under assembler language rules is acceptable. However, do not use the single letters U or C, because they are symbols for special conditions recognized by the system. Specify the assigned character set code on the DD statement or SETPRT macro to load the image into the UCS buffer.

You can supply an alias name for a new image with the ALIAS statement. (For more information on the ALIAS statement, see the publication *DFSMS/MVS Program Management*.)

2. The first byte of the character set image load module specifies whether the image is a default. If a program issues an OPEN macro to a printer in which the UCS buffer has been loaded with a default image and the JCL does not specify a UCS name, the system uses the image in the buffer. If the buffer has not been loaded with a default image the system directs the operator to take action. Specify the following in the first byte:

   For JES2:

Value	Meaning
X'00'	Indicates that the image is not to be used as a default.
X'40'	Indicates that the output is to be folded.
X'80'	Indicates a default image.
X'C0'	Indicates default image and folding.

   For non-JES2:

Value	Meaning
X'00'	Indicates that the image is not to be used as a default.
X'80'	Indicates a default image.

3. The second byte of the load module indicates the number of lines (n) to be printed for image verification. See “Verifying the UCS Image” on page 310 for more information on image verification.

4. Each byte of the next n bytes indicates the number of characters to be printed on each verification line. For the 3211 Printer, the maximum number of characters printed per line is 48; the bytes of associative bits (see Rule 5, which follows) do not print during verification.

5. The UCS image itself must follow the previously described fields. The image must fill the number of bytes required by the printer; see the following table for image lengths. Note that, because of Assembler language syntax, you must code two apostrophes or two ampersands to represent a single apostrophe or a single ampersand, respectively, within a character set image.

Printer	Image Length
1403	240 bytes
3203	304 bytes (240 characters followed by 64 bytes of associative bits)
3211	512 bytes (432 characters followed by 15 bytes of X'00', 64 bytes of associative bits, and 1 reserved byte of X'00')

You must code associative bits to prevent data checks when adding a UCS image to SYS1.IMAGELIB. See the appropriate printer manual for more information on coding associative bits.

Examples of UCS Image Coding

Figure 109 shows an example of the JCL to add a 1403 UCS image, YN, to SYS1.IMAGELIB. Notes, which apply to all examples, follow Figure 111 on page 301.

```
//ADDYN     JOB  MSGLEVEL=1
//STEP      EXEC PROC=ASMHCL,PARM.ASM='NODECK,LOAD',
//             PARM.LKED='LIST,OL,REFR,RENT,XREF'
//ASM.SYSIN DD *
UCS1YN   CSECT
         DC   X'80'        (THIS IS A DEFAULT IMAGE)
         DC   AL1(6)       (NUMBER OF LINES TO BE PRINTED)
         DC   AL1(39)      (39 CHARACTERS TO BE PRINTED ON LINE 1)
         DC   AL1(42)      (42 CHARACTERS TO BE PRINTED ON LINE 2)
         DC   AL1(39)      (39 CHARACTERS TO BE PRINTED ON LINE 3)
         DC   AL1(39)      (39 CHARACTERS TO BE PRINTED ON LINE 4)
         DC   AL1(42)      (42 CHARACTERS TO BE PRINTED ON LINE 5)
         DC   AL1(39)      (39 CHARACTERS TO BE PRINTED ON LINE 6)
*    THE FOLLOWING SIX LINES REPRESENT THE TRAIN IMAGE
         DC   C'1234567890STABCDEFGHIJKLMNOPQRUVWXYZ*,.'
         DC   C'1234567890STABCDEFGHIJKLMNOPQRUVWXYZ*,.#-$'
         DC   C'1234567890STABCDEFGHIJKLMNOPQRUVWXYZ*,.'
         DC   C'1234567890STABCDEFGHIJKLMNOPQRUVWXYZ*,.'
         DC   C'1234567890STABCDEFGHIJKLMNOPQRUVWXYZ*,.#-$'
         DC   C'1234567890STABCDEFGHIJKLMNOPQRUVWXYZ*,.'
         END
/*
//LKED.SYSLMOD DD DSNAME=SYS1.IMAGELIB(UCS1YN),DISP=OLD,
//             SPACE=        (OVERRIDE SECONDARY ALLOCATION)
```

Figure 109. Code to Add a 1403 UCS Image to SYS1.IMAGELIB

Notes

See Figure 111 on page 301 for the notes to this figure.

Figure 110 on page 300 shows an example of the JCL to add a 3203 UCS image, YN, to SYS1.IMAGELIB. Notes to this figure follow Figure 111 on page 301.

```
//ADYN3203  JOB  MSGLEVEL=1
//STEP      EXEC PROC=ASMHCL,PARM.ASM='NODECK,LOAD',
//               PARM.LKED='LIST,OL,REFR,RENT,XREF'
//ASM.SYSIN DD *
UCS3YN   CSECT
         DC   X'80'        (THIS IS A DEFAULT IMAGE)
         DC   AL1(6)       (NUMBER OF LINES TO BE PRINTED)
         DC   AL1(39)      (39 CHARACTERS TO BE PRINTED ON LINE 1)
         DC   AL1(42)      (42 CHARACTERS TO BE PRINTED ON LINE 2)
         DC   AL1(39)      (39 CHARACTERS TO BE PRINTED ON LINE 3)
         DC   AL1(39)      (39 CHARACTERS TO BE PRINTED ON LINE 4)
         DC   AL1(42)      (42 CHARACTERS TO BE PRINTED ON LINE 5)
         DC   AL1(39)      (39 CHARACTERS TO BE PRINTED ON LINE 6)
*    THE FOLLOWING SIX LINES REPRESENT THE TRAIN IMAGE
         DC   C'1234567890STABCDEFGHIJKLMNOPQRUVWXYZ*,.'
         DC   C'1234567890STABCDEFGHIJKLMNOPQRUVWXYZ*,.#-$'
         DC   C'1234567890STABCDEFGHIJKLMNOPQRUVWXYZ*,.'
         DC   C'1234567890STABCDEFGHIJKLMNOPQRUVWXYZ*,.'
         DC   C'1234567890STABCDEFGHIJKLMNOPQRUVWXYZ*,.#-$'
         DC   C'1234567890STABCDEFGHIJKLMNOPQRUVWXYZ*,.'
*    THE FOLLOWING FOUR DC INSTRUCTIONS DEFINE THE ASSOCIATIVE BITS,
*    UCSB BYTE POSITIONS 241-304
         DC   X'C0101010101010101010004000000000010'
         DC   X'1010101010101010004040000000040001010'
         DC   X'1010101010100040000000001010101010'
         DC   X'10101010004000000000'
         END
/*
//LKED.SYSLMOD DD DSNAME=SYS1.IMAGELIB(UCS3YN),DISP=OLD,
//             SPACE=         (OVERRIDE SECONDARY ALLOCATION)
```

Figure 110. Code to Add a 3203 UCS Image to SYS1.IMAGELIB

Notes

See Figure 111 on page 301 for the notes to this figure.

Figure 111 on page 301 shows an example of the JCL to add a 3211 UCS image, A11, to SYS1.IMAGELIB.

```
//ADDA11    JOB  MSGLEVEL=1
//STEP      EXEC PROC=ASMHCL,PARM.ASM='NODECK,LOAD',
//               PARM.LKED='LIST,OL,REFR,RENT,XREF'
//ASM.SYSIN DD *
UCS2A11  CSECT
         DC   X'80'        (THIS IS A DEFAULT IMAGE)
         DC   AL1(9)       (NUMBER OF LINES TO BE PRINTED)
         DC   AL1(48)      (48 CHARACTERS TO BE PRINTED ON LINE 1)
         DC   AL1(48)      (48 CHARACTERS TO BE PRINTED ON LINE 2)
         DC   AL1(48)      (48 CHARACTERS TO BE PRINTED ON LINE 3)
         DC   AL1(48)      (48 CHARACTERS TO BE PRINTED ON LINE 4)
         DC   AL1(48)      (48 CHARACTERS TO BE PRINTED ON LINE 5)
         DC   AL1(48)      (48 CHARACTERS TO BE PRINTED ON LINE 6)
         DC   AL1(48)      (48 CHARACTERS TO BE PRINTED ON LINE 7)
         DC   AL1(48)      (48 CHARACTERS TO BE PRINTED ON LINE 8)
         DC   AL1(48)      (48 CHARACTERS TO BE PRINTED ON LINE 9)
*    THE FOLLOWING NINE LINES REPRESENT THE TRAIN IMAGE
*    NOTE 2 AMPERSANDS MUST BE CODED TO GET 1 IN ASSEMBLER SYNTAX
         DC   C'1&lt;.+IHGFEDCBA*$-RQPONMLKJ%,&amp.&amp.ZYXWVUTS/@#098765432'
         DC   C'1&lt;.+IHGFEDCBA*$-RQPONMLKJ%,&amp.&amp.ZYXWVUTS/@#098765432'
         DC   C'1&lt;.+IHGFEDCBA*$-RQPONMLKJ%,&amp.&amp.ZYXWVUTS/@#098765432'
         DC   C'1&lt;.+IHGFEDCBA*$-RQPONMLKJ%,&amp.&amp.ZYXWVUTS/@#098765432'
         DC   C'1&lt;.+IHGFEDCBA*$-RQPONMLKJ%,&amp.&amp.ZYXWVUTS/@#098765432'
         DC   C'1&lt;.+IHGFEDCBA*$-RQPONMLKJ%,&amp.&amp.ZYXWVUTS/@#098765432'
         DC   C'1&lt;.+IHGFEDCBA*$-RQPONMLKJ%,&amp.&amp.ZYXWVUTS/@#098765432'
         DC   C'1&lt;.+IHGFEDCBA*$-RQPONMLKJ%,&amp.&amp.ZYXWVUTS/@#098765432'
         DC   C'1&lt;.+IHGFEDCBA*$-RQPONMLKJ%,&amp.&amp.ZYXWVUTS/@#098765432'
         DC   15X'00'      (RESERVED FIELD, BYTES 433-447)
*    THE FOLLOWING FOUR DC INSTRUCTIONS DEFINE THE ASSOCIATIVE BITS,
*    UCSB BYTE POSITIONS 448-511
         DC   X'C0101010101010101010004040424000 4010'
         DC   X'1010101010101010004040410000404010 10'
         DC   X'1010101010100040400000001010101010'
         DC   X'10101010004040444800'
         DC   X'00'        (RESERVED FIELD, BYTE 512)
         END
/*
//LKED.SYSLMOD DD DSNAME=SYS1.IMAGELIB(UCS2A11),DISP=OLD,
//             SPACE=         (OVERRIDE SECONDARY ALLOCATION)
```

Figure 111. Sample Code to Add a 3211 UCS Image to SYS1.IMAGELIB

Notes to Figure 109 on page 299, Figure 110 on page 300, and Figure 111:

1. The RENT linkage editor attribute is required.
2. For the 3203 and 3211 printers, to avoid data checks code the 64 bytes of associative bits. To determine how to code these bits for a particular image, see the publication *IBM 3203 Printer Component Description and Operator's Guide* or *IBM 3211 Printer, 3216 Interchangeable Train Cartridge, and 3811 Printer Control Unit Component Description and Operator's Guide.*
3. Executing the ASMHCL procedure does not generate executable code. The assembler/linkage editor merely places the UCS image into SYS1.IMAGELIB.
4. The SPACE parameter is overridden here because the ASMHCL cataloged procedure has secondary allocation specified. You can specify use of the original secondary allocation amount by deleting the override.

UCS Image Alias Names

Alias names are provided for many of the IBM-supplied print bands and trains. For example, if the data set were printed on a 3211, a request for the 1403 TN train would assign the T11 train. The assigned alias names that follow the naming conventions currently used in SYS1.IMAGELIB are:

Image	Alias
UCS1AN	UCS1A11
UCS1HN	UCS1H11
UCS1PN	UCS1P11
UCS1TN	UCS1T11
UCS2A11	UCS2AN
UCS2H11	UCS2HN
UCS2P11	UCS2PN, UCS2RN, UCS2QN
UCS2T11	UCS2TN

The image and alias names are included in SYS1.IMAGELIB at system installation.

Some bands/trains, such as SN and G11, do not have aliases because neither has an equivalent band/train on the other printer. An installation can assign an alias, if it chooses. (For details about the ALIAS statement, see *DFSMS/MVS Program Management*.) If you do not specify a name (or alias), you must specify an installation-defined SYSOUT class or a printer routing code to assign the data set to the correct printer. If JES is directed to print a data set on a printer for which the specified image does not exist, JES notifies the operator. The operator can then cause the data set to be printed with a valid band/train or redirect the data set to the proper printer. If an installation defines a new band/train, it can supply an alias name for it through the ALIAS statement when including the image in SYS1.IMAGELIB.

UCS Image Tables in SYS1.IMAGELIB

This section applies only to the IBM 3262 Model 5, 4245[8] , 4248[8] , and 6262 Model 14 printers. SYS1.IMAGELIB does not contain UCS images for these printers, but, instead, contains image tables. The UCS image for each band is stored within the printer and is automatically loaded into the UCS buffer when you turn on machine power or install a new band. See Figure 112 on page 303 for the format of image table entries, and “Adding or Modifying a UCS Image Table Entry” on page 307 for information on how to add or modify an image table entry.

SYS1.IMAGELIB contains one UCS image table for each type of printer that supports image tables. An image table contains an entry for most installation-standard IBM-supplied bands. The 4245 image table is named UCS5. The shared 4248, 3262 Model 5, and 6262 Model 14 image table is named UCS6.

[8] If you are running the printer in 3211 compatibility mode, UCS information is contained in image tables. Use the 3211-related sections of this chapter.

Alias Names in UCS Image Tables

The image tables also define alias names for most installation-standard print bands used on the IBM 4245 and 4248 printers. The IBM-supplied image tables do not provide alias names for the IBM 3262 Model 5 or 6262 Model 14 printers.

Some print bands, such as SN and KA22, do not have alias names because there is no equivalent band on other printers. You can add an alias name by adding or modifying an entry in the appropriate UCS image table. See “Adding or Modifying a UCS Image Table Entry” on page 307. A typical UCS image table entry is shown in Figure 112.

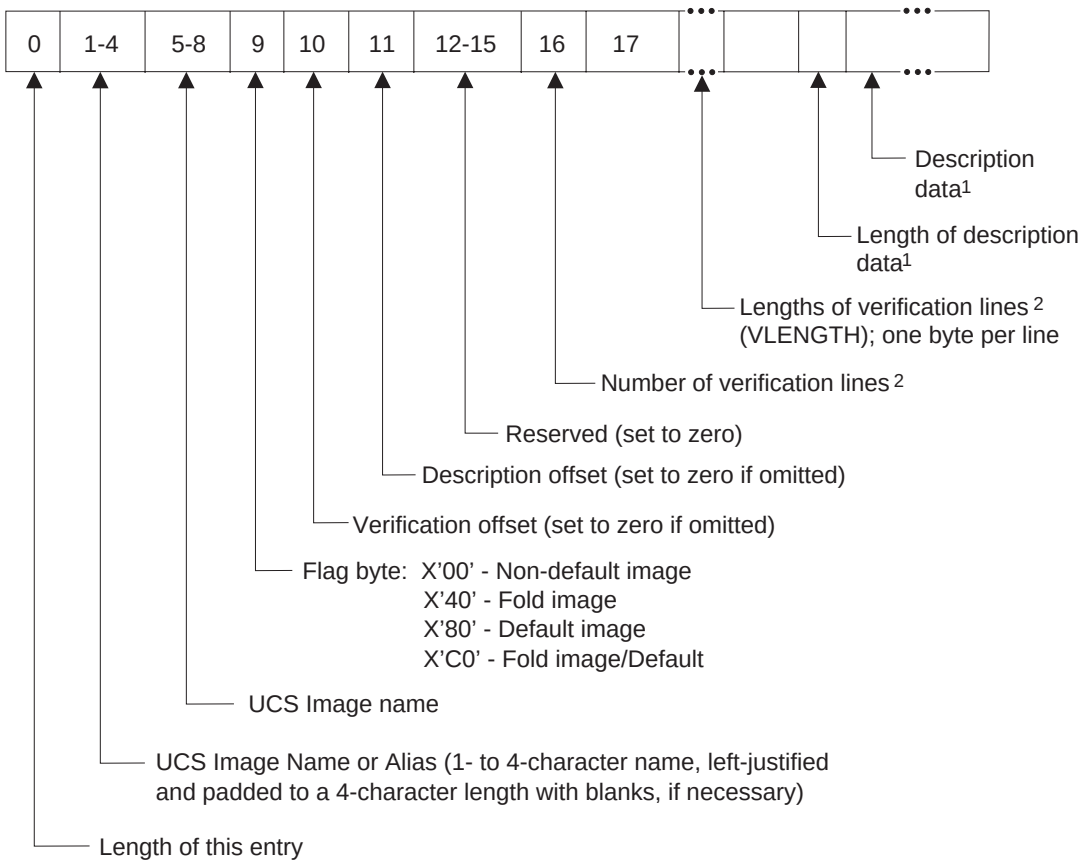


Figure 112. UCS Image Table Entry Format

Notes to Figure 112:

1. This field is optional. The description data field is of variable length, up to a maximum of 32 bytes.
2. This field is optional for the 4245 Printer. For the 3262 Model 5, 4248, and 6262 Model 14, this field does not apply and is set to `X'00'`.

The contents of the UCS image table UCS5 (IGGUCS5 macro) for the 4245 Printer are shown in Figure 113.

Figure 113. UCS5 Image Table Contents

Name	Alias	Default	Description
AN21	AN21	YES	Default UCS image
AN21	AN	NO	1403/3203 AN image
AN21	A11	NO	3211 A11 image
AN21	40E1	NO	4248 40E1 image
HN21	HN21	NO	Nondefault UCS image
HN21	HN	NO	1403/3203 HN image
HN21	H11	NO	3211 H11 image
HN21	4101	NO	4248 4101 image
PL21	PL21	NO	Nondefault UCS image
PL21	PN	NO	1403/3203 PN image
PL21	P11	NO	3211 P11 image
PL21	4121	NO	4248 4121 image
SN21	SN21	NO	Nondefault UCS image
SN21	4201	NO	4248 4201 image
TN21	TN21	NO	Nondefault UCS image
TN21	TN	NO	1403/3203 TN image
TN21	T11	NO	3211 T11 image
TN21	4181	NO	4248 4181 image
GN21	GN21	NO	Nondefault UCS image
GN21	G11	NO	3211 G11 image
GN21	41C1	NO	4248 41C1 image
RN21	RN21	NO	Nondefault UCS image
RN21	RN	NO	1403/3203 RN image
KA21	KA21	NO	Nondefault UCS image
KA21	4041	NO	4248 4041 image
KA22	KA22	NO	Nondefault UCS image
FC21	FC21	NO	Nondefault UCS image
FC21	4161	NO	4248 4161 image

The contents of the UCS image table UCS6 (IGGUCS6 macro), for the 4248 printer, are shown in Figure 114.

Figure 114. UCS6 Image Table Contents

Name	Alias	Default	Description
40E1	40E1	YES	Default UCS image
40E1	AN21	NO	4245 AN21 image
40E1	AN	NO	1403/3203 AN image
40E1	A11	NO	3211 A11 image
4101	4101	NO	Nondefault UCS image
4101	HN21	NO	4245 HN21 image
4101	HN	NO	1403/3203 HN image
4101	H11	NO	3211 H11 image
41C1	41C1	NO	Nondefault UCS image
41C1	GN21	NO	4245 GN21 image
41C1	G11	NO	3211 G11 image
4121	4121	NO	Nondefault UCS image
4121	PL21	NO	4245 PL21 image
4121	PN	NO	1403/3203 PN image
4121	P11	NO	3211 P11 image
4181	4181	NO	Nondefault UCS image
4181	TN21	NO	4245 TN21 image
4181	TN	NO	1403/3203 TN image
4181	T11	NO	3211 T11 image
4061	4061	NO	Nondefault UCS image
40C1	40C1	NO	Nondefault UCS image
4161	4161	NO	Nondefault UCS image
4161	FC21	NO	4245 FC21 image
4201	4201	NO	Nondefault UCS image
4201	SN21	NO	4245 SN21 image
4041	4041	NO	Nondefault UCS image
4041	KA21	NO	4245 KA21 image

Note: The image tables for the 4245 and 4248 printers include USA and Canada band IDs only. To support other national band IDs, modify the UCS image table. See Figure 115 on page 306 and “Adding or Modifying a UCS Image Table Entry” on page 307.

The 3262 Model 5 and 6262 Model 14 printers use the 4248 UCS image table, UCS6. However, IBM does not provide band names or aliases for either the 3262 Model 5 or 6262 Model 14 printer. To use 3262 Model 5 or 6262 Model 14 UCS images, add the names and aliases to UCS6. “Adding or Modifying a UCS Image Table Entry” on page 307 describes how to add entries to the UCS image table. For a list of the bands available for the 3262 Model 5, see Figure 115. For a list of the bands available for the 6262 Model 14, see the publications *IBM 6262 Printer Print Band Manual* and *IBM 6262 Printer Model 014 User’s Guide.*

Figure 115. 3262 Model 5 Print Bands

Character Set Name	UCS6 Band Name [1]	3262 Model 5 Band Image Select Switch Settings (positions 1-8[2])
U.S./International		
48 char EBCDIC	00*bb*[3]	*xx*00 0000
63 char EBCDIC	01*bb*	*xx*00 0001
64 char EBCDIC	02*bb*	*xx*00 0010
96 char EBCDIC	03*bb*	*xx*00 0011
48 char AON OCR	04*bb*	*xx*00 0100
48 char BON OCR	05*bb*	*xx*00 0101
Austria/Germany		
52 char EBCDIC	06*bb*	*xx*00 0110
63 char EBCDIC	07*bb*	*xx*00 0111
64 char EBCDIC	08*bb*	*xx*00 1000
96 char EBCDIC	09*bb*	*xx*00 1001
52 char AON OCR	0A*bb*	*xx*00 1010
52 char BON OCR	0B*bb*	*xx*00 1011
Canada/French		
116 char EBCDIC	0C*bb*	*xx*00 1100
Katakana		
96 char EBCDIC	0D*bb*	*xx*00 1101
128 char EBCDIC	0E*bb*	*xx*00 1110
UINN		
128 char U.S. text	0F*bb*	*xx*00 1111
WTNN		
128 char World Trade text	10*bb*	*xx*01 0000

Notes

1. You can define any 1- to 4-character name as aliases of these names.
2. *xx* designates switch settings that are ignored. Switch number 3 must be on if you are using a special order (RPQ) band.
3. Please note that *bb* here represents a space.

Adding or Modifying a UCS Image Table Entry

To use a new UCS image name/alias with the 3262 Model 5, 4245, 4248, or 6262 Model 14 printers, add an entry for that image name/alias to the UCS image table. Use the assembler to create the image table object module, then link-edit the object module into SYS1.IMAGELIB, as shown in the following procedure. Similarly, to specify other images as defaults or change the description on an old image, change the image table.

To build new UCS table entries, or to change the format of old entries, use the following procedure. For examples of coding the IGGUCSIT macro, see Figure 116 on page 309 and Figure 117 on page 310.

1. To build a new UCS image table entry issue the IGGUCSIT macro, as described in the following text. If you are updating the image table as shown in the following examples, the linkage editor builds a new entry at the start of the table, even if you intended to replace an existing entry. When the system subsequently uses the table, it encounters the new entry first, thus the old one is effectively replaced.
2. Include the UCS image table source, using the IGGUCS5 or IGGUCS6 macro, both of which reside in SYS1.MODGEN.
3. Assemble the image table module (UCS5 or UCS6).
4. Link-edit the assembled module into SYS1.IMAGELIB.

   Note: The RENT linkage editor attribute is required.

The format of the IGGUCSIT macro is:

```
                   ┌LIST┐
►►─IGGUCSIT─MF=─┤    ├──,NAME=image_name──────────────────────►
                   └DSECT┘
►──┬──────────────────────┬──┬──────────────────────────────┬──►
   └─,ALIAS=image_alias───┘  │              ┌NO┐            │
                             └─,DEFAULT=─┤    ├──┘            │
                                          └YES┘
►──┬──────────────────────┬──┬──────────────────────────────┬──►
   └─,DESCR=description───┘  │              ┌4245┐          │
                             └─,DEVICE=─┤    ├──┘            │
                                          └4248┘
►──┬──────────────────────────┬──┬──────────────────────────┬──►◄
   └─,VLENGTH=(n1,n2,...n)────┘  │              ┌NO┐        │
                                 └─,FOLD=─┤    ├──┘          │
                                          └YES┘
```

MF=LIST or **DSECT**
specifies the form of the macro instruction.

LIST
produces a UCS image table entry based on the information supplied in other IGGUCSIT parameters. If LIST is selected or allowed to default, the NAME parameter must also be coded.

DSECT
produces a DSECT for a single UCS image table entry, similar to the sample entry shown in Figure 112 on page 303. If you code DSECT, all other parameters of IGGUCSIT are ignored.

LIST is the default.

NAME=*image_name*
specifies the one to four character UCS image name.

ALIAS=*image_alias*
specifies a one to four character alias name for the UCS image. If ALIAS is not specified, the image name coded in the NAME parameter will be entered in the UCS image table.

Note: The 3262, 4248, and 6262 printers have no band IDs in common. Because these devices all use the UCS6 image table, select a unique alias name for each of these printer types. The alias must not appear in the image table more than once.

DEFAULT=YES or **NO**
indicates whether the new UCS image is to be used as a default value.

YES
indicates that this UCS image is a default. Default images are used by the system for jobs that do not request a specific image.

NO
indicates that this UCS image should not be used as a default.

If the DEFAULT parameter is not specified, the new UCS image is not used as a default.

DESCR=*description*
specifies descriptive information about the new UCS image. *description* can be up to 32 EBCDIC or hexadecimal characters long. You cannot use EBCDIC and hexadecimal characters in combination.

Descriptive information is placed in the header line of the verification display, following the real UCS image name. If you omit the DESCR parameter, no description appears in the display. For more information on the verification display, see “Verifying the UCS Image” on page 310.

If VLENGTH is not specified for the 4245 Printer, the DESCR parameter is ignored.

DEVICE=4245 or **4248**
specifies the type of device for which an image table entry is to be created.

If you specify MF=LIST on the first invocation of the IGGUCSIT macro, DEVICE defaults to 4245. The default for subsequent invocations is the printer type that you specified (or the default) on the first invocation. Table entries with different DEVICE specifications are not allowed.

For the 3262 Model 5 or 6262 Model 14 printers, DEVICE=4248 should be specified to create the appropriate form of the image table entry.

VLENGTH=(*n1*,*n2*,**. . .** *n***)**
specifies the lengths of each line in the UCS verification display. The length of each line must be specified separately, even if all lines are of the same length.

n1 is the length of print line 1; *n2* is the length of print line 2; *n* is the length of the last print line. To display the complete image, the sum of the verification line lengths should equal 350.

For details on the verification report, see “Verifying the UCS Image” on page 310.

The VLENGTH parameter is not valid for the 3262 Model 5, 4248, or 6262 Model 14 printers.

FOLD=YES or **NO**
indicates whether the UCS image is to be folded.

YES
indicates that the UCS image is to be folded. Allows printing only uppercase characters from either upper- or lowercase data codes. Folding continues until an UNFOLD command is received.

NO
indicates that the UCS image is not to be folded. This is the default.

Adding to the UCS Image Table

In Figure 116, the band name RPQ1 with description “RPQ BAND” is added to UCS5. In the UCS verification display, 7 lines of 50 characters each are printed. Macro IGGUCS5 causes the UCS image table source (as distributed by IBM) to be included in the table entry.

```
                                                                    72
//UCS5   JOB  . . .
//       EXEC ASMHCL,
//             PARM.ASM='NODECK,LOAD',
//             PARM.LKED='OL,RENT,REUS'
//SYSPRINT  DD SYSOUT=A
//ASM.SYSIN DD *
         TITLE 'UPDATED UCS5 IMAGE TABLE'
UCS5     CSECT
         IGGUCSIT NAME=RPQ1,                                        X
               VLENGTH=(50,50,50,50,50,50,50),                      X
               DESCR='RPQ BAND'
         IGGUCS5
         END
/*
//LKED.SYSLMOD DD DSN=SYS1.IMAGELIB(UCS5),DISP=OLD,
//             SPACE=  (OVERRIDE SECONDARY ALLOCATION)
```

Figure 116. Adding a New Band ID to the 4245 UCS Image Table (UCS5)

Notes to Figure 116:

1. The RENT linkage editor attribute is required.
2. Executing the ASMHCL procedure does not generate executable code. The assembler/linkage editor places the updated UCS image table into SYS1.IMAGELIB.
3. The SPACE parameter is overridden here because the ASMHCL cataloged procedure has a secondary allocation specified. Eliminating the override causes the original secondary allocation amount to be used.

In Figure 117 on page 310 the band name 40E1 DEFAULT BAND has been added to UCS6 and defined as a default band. An alias name, HN21, is also defined for band 40E1. Macro IGGUCS6 causes the UCS image table source (as distributed by IBM) to be included in the table entry.

```
                                                                  72
//UCS6   JOB  . . .
//       EXEC ASMHCL,
//             PARM.ASM='NODECK,LOAD',
//             PARM.LKED='OL,RENT,REUS'
//SYSLIB    DD
//          DD DSN=SYS1.AMODGEN,DISP=SHR
//SYSPRINT  DD SYSOUT=A
//ASM.SYSIN DD *
         TITLE 'UPDATED UCS6 IMAGE TABLE'
UCS6     CSECT
         IGGUCSIT NAME=40E1,                                      X
               DEVICE=4248,                                       X
               ALIAS=HN21,                                        X
               DEFAULT=YES,                                       X
               DESCR='40E1 DEFAULT BAND'
         IGGUCS6
         END
/*
//LKED.SYSLMOD DD DSN=SYS1.IMAGELIB(UCS6),DISP=OLD,
//             SPACE=  (OVERRIDE SECONDARY ALLOCATION)
```

Figure 117. Adding a New Default Entry to the 4248 UCS Image Table (UCS6).

Notes to Figure 117:

1. This method creates a duplicate entry for 40E1 that becomes the first entry in the table. Because the table is searched sequentially, the new entry is always found before the old entry, thus replacing the old entry.
2. The RENT linkage editor attributes is required.
3. Executing the ASMHCL procedure does not generate executable code. The assembler/linkage editor places the updated UCS image table into SYS1.IMAGELIB.
4. The SPACE parameter is overridden because the ASMHCL cataloged procedure has a secondary allocation specified. Eliminating the override causes the original secondary allocation amount to be used.

Verifying the UCS Image

For the 1403 (with the UCS feature), 3203, 3211, 3262 Model 5, 4245, 4248, and 6262 Model 14 printers, you can print the UCS image for visual verification using either of the following parameters:

- In JCL: **UCS=(***character set code***,,VERIFY)**
- In the SETPRT macro: **UCS=(***character set code***,,V)**.

These parameters have no effect for SYSOUT data sets.

You can also use these parameters for the 3262 Model 5, 4248, and 6262 Model 14 printers. However, because the UCS image cannot be read directly from the 3262 Model 5, 4248, or 6262 Model 14, only the header information is printed. The verification display header appears on the printer as follows:

`UCS IMAGE VERIFICATION` *`image_id`* `[,FOLD] [`*`description`*`]`

image_id
A one to four character name of the UCS image.

description
The descriptive information supplied for this UCS image in the UCS image table.

For more information about the UCS VERIFY parameters, see *OS/390 MVS JCL Reference* and *DFSMS/MVS Macro Instructions for Data Sets*.

FCB Images in SYS1.IMAGELIB

Two standard FCB images, STD1 and STD2, are included in SYS1.IMAGELIB during system installation. The names of these standard images begin with the characters “FCB2”. You can define FCB images whose names begin with “FCB4” for the 4248, 3262 Model 5, and 6262 Model 14 Printers. These printers can use both the “FCB2”-prefixed images (referred to as 3211 format FCBs) and the “FCB4”-prefixed images. All other printers, except the 3800, can use only “FCB2”-prefixed images.

This section describes how you can create or replace “FCB2”-prefixed images. *DFSMS/MVS Utilities* describes how you can use the IEBIMAGE utility program to create, update, or replace an “FCB4”-prefixed image for the 3262 Model 5, 4248, or 6262 Model 14 Printers. For details about the “FCB4”-prefixed images, see either *IBM 6262 Printer Model 014 Product Description* or *IBM 4248 Printer Model 1 Description.*

The 3262 Model 5, 4245, 4248, and 6262 Model 14 printers each load a default FCB image into the buffer when they are powered on. The 3262 Model 5 default FCB image is an 11-inch form with 6 lines per inch, a Channel 1 on the third print line, and a Channel 12 on line 64. The 4245 default FCB image is an 11-inch form with 6 lines per inch and a Channel 1 on the first print line. The 4248 default FCB image is the last FCB image loaded. The 6262 Model 14 default FCB image is either the last FCB image loaded and saved or the default shipped with the printer.

STD1 sets line spacing at 6 lines per inch for an 8½ inch form; STD2 is a default FCB image that sets line spacing at 6 lines per inch for an 11-inch form. Channels for both images are evenly spaced, with Channel 1 on the fourth line and Channel 9 on the last line. See Figure 118 on page 312 and Figure 119 on page 313 for the format of the standard STD1 and STD2 images.

The standard FCB image for the 3800 Printing Subsystem, STD3, is included in SYS1.IMAGELIB during system installation. All models of the 3800 use FCB images whose names begin with “FCB3.” Use the IEBIMAGE utility to create and modify FCB modules for the 3800 Printing Subsystem.

```
FCB2STD1  CSECT
          DC    X'80'             DEFAULT
          DC    AL1(48)           FCB IMAGE LENGTH = 48
          DC    X'000000'         LINE 1, 2, 3
          DC    X'01'             LINE 4, CHANNEL 1
          DC    X'000000'         LINE 5, 6, 7
          DC    X'02'             LINE 8, CHANNEL 2
          DC    X'000000'         LINE 9, 10, 11
          DC    X'03'             LINE 12, CHANNEL 3
          DC    X'000000'         LINE 13, 14, 15
          DC    X'04'             LINE 16, CHANNEL 4
          DC    X'000000'         LINE 17, 18, 19
          DC    X'05'             LINE 20, CHANNEL 5
          DC    X'000000'         LINE 21, 22, 23
          DC    X'06'             LINE 24, CHANNEL 6
          DC    X'000000'         LINE 25, 26, 27
          DC    X'07'             LINE 28, CHANNEL 7
          DC    X'000000'         LINE 29, 30, 31
          DC    X'08'             LINE 32, CHANNEL 8
          DC    X'000000'         LINE 33, 34, 35
          DC    X'0A'             LINE 36, CHANNEL 10
          DC    X'000000'         LINE 37, 38, 39
          DC    X'0B'             LINE 40, CHANNEL 11
          DC    X'000000'         LINE 41, 42, 43
          DC    X'0C'             LINE 44, CHANNEL 12
          DC    X'000000'         LINE 45, 46, 47
          DC    X'19'             LINE 48, CHANNEL 9-END OF FCB IMAGE
          END
```

Figure 118. Format of the Standard STD1 FCB Image

```
FCB2STD2  CSECT
          DC    X'80'             DEFAULT
          DC    AL1(66)           FCB IMAGE LENGTH = 66
          DC    X'000000'         LINE 1, 2, 3
          DC    X'01'             LINE 4, CHANNEL 1
          DC    X'0000000000'     LINE 5, 6, 7, 8, 9
          DC    X'02'             LINE 10, CHANNEL 2
          DC    X'0000000000'     LINE 11, 12, 13, 14, 15
          DC    X'03'             LINE 16, CHANNEL 3
          DC    X'0000000000'     LINE 17, 18, 19, 20, 21
          DC    X'04'             LINE 22, CHANNEL 4
          DC    X'0000000000'     LINE 23, 24, 25, 26, 27
          DC    X'05'             LINE 28, CHANNEL 5
          DC    X'0000000000'     LINE 29, 30, 31, 32, 33
          DC    X'06'             LINE 34, CHANNEL 6
          DC    X'0000000000'     LINE 35, 36, 37, 38, 39
          DC    X'07'             LINE 40, CHANNEL 7
          DC    X'0000000000'     LINE 41, 42, 43, 44, 45
          DC    X'08'             LINE 46, CHANNEL 8
          DC    X'0000000000'     LINE 47, 48, 49, 50, 51
          DC    X'0A'             LINE 52, CHANNEL 10
          DC    X'0000000000'     LINE 53, 54, 55, 56, 57
          DC    X'0B'             LINE 58, CHANNEL 11
          DC    X'0000000000'     LINE 59, 60, 61, 62, 63
          DC    X'0C'             LINE 64, CHANNEL 12
          DC    X'00'             LINE 65
          DC    X'19'             LINE 66, CHANNEL 9-END OF FCB IMAGE
          END
```

Figure 119. Format of the Standard STD2 FCB Image

Adding an FCB Image to the Image Library

You can add a 3211-format FCB image to those that reside in SYS1.IMAGELIB, using the assembler and linkage editor. No executable code is generated; the assembler prepares DCs, and the linkage editor links them into SYS1.IMAGELIB. The new FCB image must be structured according to the following rules:

1. The member name cannot exceed 8 bytes and must begin with the prefix FCB2. The characters that follow identify the FCB image and are referred to as the image identifier (ID). Any combination of valid assembler language characters can be used, with the exception of a single `'C'` or `'U'`, because these are used by the system to recognize special conditions. To load the image into the FCB buffer, the image identifier must be specified in the FCB keyword of a DD statement or in the SETPRT macro.

2. The first byte of the FCB load module specifies whether the image is the default. (Default images are used by the system for jobs that do not request a specific image.) Specify the following in the first byte:

Value	Meaning
X'80'	Indicates a default image
X'00'	Indicates a nondefault image

3. The second byte of the load module indicates the number of bytes to be transferred to the control unit to load the FCB image. This count includes the byte, if used, for the print position indexing feature.

4. The third byte of the load module (the first byte of the FCB image) is either the print position indexing byte, or the lines-per-inch byte. The print position indexing byte is optional and, when used, precedes the lines-per-inch byte. The 3203 Model 5, 3262 Model 5, 4245, 4248, and 6262 Model 14 printers accept and discard the index byte if it is present, because they do not support the indexing feature. A description of the print position indexing feature and its use can be found in the publication *IBM 3211 Printer, 3216 Interchangeable Train Cartridge, and 3811 Printer Control Unit Component Description and Operator's Guide.*

   The special index flag in the third byte contains `X'80'` plus a binary index value, from 1 to 32 (the default is 1). This index value sets the left margin: 1 indicates flush-left; any other value indicates a line indented the specified number of spaces.

   The form image begins with the lines-per-inch (LPI) byte. The LPI byte defines the number of lines per inch (6 or 8) and also represents the first line of the page.

   Note: Printers controlled by JES2 require a channel 1 identifier here.

   Typically, the length of an FCB image is consistent with the length of the form it represents. For example, an 8½ inch form to be printed at 6 LPI has an FCB image that is 51 bytes long (8½ inches times 6 LPI).

   The LPI byte appears as follows:

Value	Meaning
`X'1n'`	Sets 8 LPI
`X'0n'`	Sets 6 LPI

5. All remaining bytes (lines) must contain `X'0n'`, except the last byte, which must be `X'1n'`. The letter n can be a hexadecimal value from 1 to C, representing a channel (one to 12), or it can be 0, which means no channel is indicated.

In Figure 120 on page 315, an FCB load module is assembled and added to SYS1.IMAGELIB. The image defines a print density of 8 lines per inch on an 11-inch form, with a right shift of 15 line character positions (1½ inches).

```
//ADDFCB  JOB   MSGLEVEL=1
//STEP    EXEC  PROC=ASMHCL,PARM.ASM='NODECK,LOAD',
//              PARM.LKED='LIST,OL,REFR,RENT,XREF'
//ASM.SYSIN DD  *
FCB2ID1   CSECT
*THIS EXAMPLE IS FOR A FORM LENGTH OF 11 INCHES WITH 8 LPI (88 LINES)
          DC    X'80'     THIS IS A DEFAULT IMAGE
          DC    AL1(89)   LENGTH OF FCB IMAGE AND INDEXING BYTE
          DC    X'8F'     OFFSET 15 CHARACTERS TO THE RIGHT
          DC    X'10'     8 LINES PER INCH-NO CHANNEL FOR LINE 1
          DC    XL4'0'    4 LINES NO CHANNEL
          DC    X'01'     CHANNEL 1 IN LINE 6
          DC    XL6'0'    6 LINES NO CHANNEL
          DC    X'02'     CHANNEL 2 IN LINE 13
          DC    XL6'0'    6 LINES NO CHANNEL
          DC    X'03'     CHANNEL 3 IN LINE 20
          DC    XL6'0'    6 LINES NO CHANNEL
          DC    X'04'     CHANNEL 4 IN LINE 27
          DC    XL6'0'    6 LINES NO CHANNEL
          DC    X'05'     CHANNEL 5 IN LINE 34
          DC    XL6'0'    6 LINES NO CHANNEL
          DC    X'06'     CHANNEL 6 IN LINE 41
          DC    XL6'0'    6 LINES NO CHANNEL
          DC    X'07'     CHANNEL 7 IN LINE 48
          DC    XL6'0'    6 LINES NO CHANNEL
          DC    X'08'     CHANNEL 8 IN LINE 55
          DC    XL6'0'    6 LINES NO CHANNEL
          DC    X'09'     CHANNEL 9 IN LINE 62
          DC    XL6'0'    6 LINES NO CHANNEL
          DC    X'0A'     CHANNEL 10 IN LINE 69
          DC    XL6'0'    6 LINES NO CHANNEL
          DC    X'0B'     CHANNEL 11 IN LINE 76
          DC    XL6'0'    6 LINES NO CHANNEL
          DC    X'0C'     CHANNEL 12 IN LINE 83
          DC    XL4'0'    4 LINES NO CHANNEL
          DC    X'10'     POSITION 88 LAST LINE IN IMAGE
          END
/*
//LKED.SYSLMOD DD DSNAME=SYS1.IMAGELIB(FCB2ID1),DISP=OLD,
//              SPACE=    (OVERRIDE SECONDARY ALLOCATION)
```

Figure 120. Sample Code to Assemble and Add an FCB Load Module to SYS1.IMAGELIB

Notes to Figure 120:

1. The RENT linkage editor attribute is required.
2. Executing the ASMHCL procedure does not generate executable code. The assembler/linkage editor is used to place the FCB image into SYS1.IMAGELIB.
3. The SPACE parameter is overridden here because the ASMHCL cataloged procedure has a secondary allocation specified. Eliminating the override causes the original secondary allocation amount to be used.

Modifying an FCB Image

To modify an FCB image in virtual storage before loading it into a forms control buffer, use the following sequence of macro instructions to read the FCB image into virtual storage.

1. An IMGLIB macro instruction, along with the OPEN parameter
2. A BLDL macro instruction to determine if the FCB image is in the image library
3. A LOAD macro instruction to load the image into virtual storage
4. After the image has been read in, issue the IMGLIB macro instruction with the CLOSE parameter and the address of the DCB built by the first IMGLIB macro.

A SETPRT macro instruction can be used to load the forms control buffer with the modified image. Printers other than the 3800 require the use of an FCB entry in an exit list, as described in *DFSMS/MVS Using Data Sets*.

The formats of the BLDL and SETPRT macros are given in *DFSMS/MVS Macro Instructions for Data Sets*; the format of the LOAD macro is given in the publications *OS/390 MVS Assembler Services Guide* and *OS/390 MVS Assembler Services Reference*.

The format of the IMGLIB macro is:

```
►►──┬──────────┬──IMGLIB──┬─OPEN──────────────────────────────────►◄
    └─label────┘          └─CLOSE,addr────┘
```

OPEN
opens SYS1.IMAGELIB and specifies that a DCB is to be built for it. The address of the DCB is returned in register 1.

CLOSE
closes SYS1.IMAGELIB.

addr
specifies the RX-type address of a word that points to the DCB. If coded in the form (*reg*), the register in parentheses contains the address of the DCB, not the address of the word.

Return Codes from IMGLIB

Return codes from the IMGLIB macro with the OPEN parameter are as follows:

Return Code	Meaning
0 (X'00')	Operation successful.
4 (X'04')	Either the volume containing SYS1.IMAGELIB is not mounted or a required catalog volume is not mounted.
8 (X'08')	Either SYS1.IMAGELIB does not exist on the volume to which the catalog points, or SYS1.IMAGELIB is not cataloged.
12 (X'0C')	An error occurred in reading the catalog or VTOC.

BLDL and LOAD are the only macros that can refer to the DCB built by the IMGLIB macro.

JES Support for the 3211 Indexing Feature

JES2 supports the 3211 indexing feature in two ways:

1. Specification of the INDEX parameter on the /*OUTPUT card
2. The extended FCB image.

   The system supplies two special FCBs: FCB26 for 6 lines per inch and FCB28 for 8 lines per inch (specified as FCB=6 and FCB=8, respectively). These FCBs contain a channel 1 indication in position 1, a special index flag in the third byte, and the number of lines per inch in the fourth byte of the image.

   The special index flag in the third byte of FCB26 and FCB28 contains X'80' plus a binary index value, in the range 1 to 32 (default=1). The index value sets the left margin (1 indicates flush-left; other values cause indentation of the print line by N-1 positions).

   If JES2 is to use any other FCB images, the images must specify channel 1 in position 1; otherwise, JES2 positions the forms in the printer incorrectly. (Because STD1 and STD2 do not specify channel 1 in position 1, they must be modified for JES2.)

   If the third byte of any other FCB image (specifying the number of lines per inch) contains a data character other than X'80', JES2 uses that specification and supplies an index value of 1.

JES3 does not support the 3211 indexing feature, and any indexing commands from JES3 are ignored by the 3203 Model 5.

Appendix A. CVAF VTOC Access Macros

This appendix includes VTOC index information that depends on internal system logic. It is intended to help you to use CVAF macro instructions to modify the VTOC.

Coding CVAF Macro Instructions

This section shows you how to code CVAF macro instructions and specify the services you want by selecting from the available parameter options. It also provides coding examples that can help you implement the macros.

CVAFDIR Macro Overview and Specification

For an indexed or nonindexed VTOC, you can use the CVAFDIR macro to:

- Read or write a DSCB by specifying the name of the data set it represents.
- Read or write a DSCB by specifying its address.

In addition, for an indexed VTOC, the macro can be used to:

- Read or write VTOC index records. (This allows calling programs to modify the VTOC index.)
- Read and retain in virtual storage the first high-level VIER, and VIERs used during an index search.
- Read and retain in virtual storage the space map VIRs.
- Free VIRs retained in virtual storage.

See “Accessing the DSCB Directly” on page 46 for additional information.

The format of the CVAFDIR macro is:

© Copyright IBM Corp. 1979, 1999

```
►►──┬──────────┬──CVADIR──ACCESS=──┬─READ──┬──┬──────────────┬──►
    └─label────┘                   ├─WRITE─┤  └─,DSN=addr────┘
                                   └─RLSE──┘

►──┬──────────────┬──┬──────────────────────┬──┬──────────────┬──►
   └─,BUFLIST=addr─┘  └─,VERIFY=──┬─NO──┬──┘  ├─,UCB=(ucbaddr)─┤
                                  └─YES─┘     └─,DEB=addr──────┘

►──┬──────────────────────────┬──┬──────────────────────────┬──►
   └─,IOAREA=──┬─NOKEEP──────┬──┘  └─,MAPRCDS=──┬─NO──────────┬──┘
               ├─KEEP────────┤                  ├─YES─────────┤
               ├─(KEEP,addr)─┤                  ├─(YES,addr)──┤
               └─(NOKEEP,addr)┘                 └─(NO,addr)───┘

►──┬──────────────────────────┬──┬──────────────────────────┬──►
   └─,IXRCDS=──┬─NOKEEP──────┬──┘  └─,BRANCH=──┬─NO──────────┬──┘
               ├─KEEP────────┤                 ├─YES─(1)─────┤
               ├─(KEEP,addr)─┤                 ├─(YES,SUP)───┤
               └─(NOKEEP,addr)┘                └─(YES,PGM)───┘

►──┬──────────────────┬──►◄
   └─,MF=──┬─I───────┬──┘
           ├─L───────┤
           └─(E,addr)┘
```

Note:
[1] If YES is coded, the default is SUP.

ACCESS: Read or Write a DSCB or VIRs, or Release Buffer Lists

When ACCESS is READ or WRITE, a single DSCB is accessed for an indexed or nonindexed VTOC, or one or more VIRs are accessed for an indexed VTOC.

ACCESS=READ

specifies that a single DSCB or one or more VIRs are to be read into a buffer whose address is in a buffer list.

If the buffer list is for a DSCB, only one entry is used in the buffer list. The first entry with the skip bit set to zero and a nonzero buffer address is used.

All VIRs whose buffer list entry has the skip bit off are read into a buffer.

DSN and BUFLIST are required if ACCESS=READ for a DSCB buffer list.

ACCESS=WRITE

specifies that a single DSCB or one or more VIRs are to be written from buffers whose address is in a buffer list.

ACCESS=WRITE is permitted with BRANCH=NO only if the caller is authorized by APF.

DSN and BUFLIST are required if ACCESS=WRITE for a DSCB buffer list.

If any buffer list entry has its modified bit set, only those entries with the modified bit set are written. If no modify bits are on, all VIRs are written.

ACCESS=RLSE

applies only to VIR buffer lists. It requests the release of one or more buffers in the VIR buffer list chain identified in the BUFLIST keyword, and the release of each buffer list for which all buffers are released.

DSN and BUFLIST are not required if ACCESS=RLSE.

Only buffers in the buffer list with the skip bit set to zero and with a nonzero buffer address are released. The buffer list is not released if any entry has the skip bit set to one.

For an indexed VTOC, if ACCESS=RLSE is coded, buffer lists and buffers pointed to by the BUFLIST keyword are released, along with buffer lists supplied in the CVAF parameter list CVMRCDS and CVIRCDS fields. If the CVMRCDS or the CVIRCDS buffers are supplied in the BUFLIST field, either directly or indirectly through chaining, the keyword MAPRCDS=YES, IXRCDS=KEEP, or MAPRCDS=(NO,0), IXRCDS=(NOKEEP,0) must be coded to prevent CVAF from freeing the buffers more than once. If buffers are released, the CVAF parameter list field pointing to the buffer list is updated.

DSN: Specify the Name of the DSCB

DSN=*addr*

supplies the address of a 44-byte data set name of the DSCB to be accessed.

DSN is required if ACCESS=READ or WRITE and the request is to read or write a DSCB. If a 140-byte DSCB is specified:

- CVAF validity checks the storage location, but ignores the contents of the location.
- Specify an argument that points to an extent within the VTOC.

BUFLIST: Specify One or More Buffer Lists

BUFLIST=*addr*

supplies the address of a buffer list used to read or write a DSCB or VIRs.

VERIFY: Verify that a DSCB is a Format-0 DSCB

VERIFY=YES

CVAF verifies that the DSCB is a format-0 DSCB before writing the DSCB. The first four bytes of the key are compared with binary zeros. If the key does not start with 4 bytes of zeros, the DSCB is not written and an error code is returned. This is the default.

VERIFY=NO

CVAF does not test the key of the DSCB.

Note: VERIFY applies only when writing a 140-byte DSCB. VERIFY is ignored when a VIR is written.

UCB or DEB: Specify the VTOC to Be Accessed

UCB= *rs-type or (2-12)* *standard form* **UCB**= *rx-type or (2-12)* *execute form*

specifies the address of the UCB for the VTOC to be accessed. The UCB address can be for a captured UCB, or for an actual UCB above or below the 16MB line. Use the address of a UCB, not a UCB copy. An unauthorized caller must not use this parameter. If your program is in 31-bit mode, this address must be in 31-bit address; the high order byte is part of the address. You should not code the UCB parameter with MF=L.

Note: Code the address of the UCB parameter as register (2-12). Coding an RX-Type address gives unpredictable results.

DEB=*addr*
supplies the address of a DEB opened to the VTOC you want to access. CVAF does not allow output requests to the VTOC or VTOC index if you specify the DEB subparameter. If you are not authorized, you cannot perform any asynchronous activity (such as EXCP, CLOSE, EOV) against the data set represented by the DEB because CVAF removes the DEB from the DEB table for the duration of the CVAF call. If you are not authorized (neither APF authorized nor in a system key), specify a DEB address, not a UCB, to CVAFDIR. See “Identifying the Volume” on page 41 for further details.

If you supply a previously obtained I/O area through the IOAREA keyword, neither UCB nor DEB need be supplied. Otherwise, supply either a UCB or DEB. If you supply a UCB address, it is overlaid in the CVPL by the UCB address in the I/O area. If you supply both the DEB and UCB addresses in the CVPL, the DEB address is used and the UCB address in the CVPL is overlaid by the UCB address in the DEB.

IOAREA: Keep or Free the I/O Work Area

IOAREA=KEEP
specifies that the CVAF I/O area associated with the CVAF parameter list is to be kept upon completion of the CVAF request. IOAREA=KEEP can be coded with BRANCH=NO only if the caller is authorized (APF or system key).

If IOAREA=KEEP is coded, the caller must call CVAF with IOAREA=NOKEEP specified at some future time, whether or not any further VTOC access is required. An example of such a caller is the recovery routine of the caller of CVAF.

Coding IOAREA=KEEP allows subsequent CVAF requests to be more efficient, as certain initialization functions can be bypassed. Neither DEB nor UCB need be specified when a previously obtained CVAF I/O area is supplied; nor can they be changed.

When IOAREA=KEEP is first issued, CVAF returns the CVAF I/O area in the CVAF parameter list (CVIOAR). Subsequent calls of CVAF can use that same parameter list, and CVAF obtains its I/O area from the CVIOAR.

When processing on the current volume is finished, release all areas that were kept.

IOAREA=(KEEP,*addr*)
supplies the address of a previously obtained I/O area. If a different CVAF parameter list is used, the previously obtained I/O area can be passed to CVAF by coding its address as the second parameter of the IOAREA keyword.

IOAREA=NOKEEP
causes the work area to be freed upon completion of the CVAF request.

IOAREA=(NOKEEP,*addr*)
causes a previously obtained work area to be freed upon completion of the CVAF request.

MAPRCDS: Keep or Free MAPRCDS Buffer List and Buffers

This keyword applies to an indexed VTOC only and specifies the disposition of the MAPRCDS buffer list and buffers.

MAPRCDS=YES

specifies that the buffer list and buffers are to be retained at the end of processing.

If no buffer list address is in the CVAF parameter list, CVAF reads the MAP VIRs into buffers it obtains. The buffer list that contains the address and RBAs of the VIRs can be accessed after processing from the CVAF parameter list field, CVMRCDS. The buffer list and VIR buffers are in your protect key: subpool 0 if you are not authorized; subpool 229 if you are.

When processing on the current volume is finished, release all areas that were kept.

MAPRCDS=(YES,*addr*)

If MAPRCDS=YES is coded and the buffer list address (CVMRCDS in CVAF parameter list) is supplied, VIRs are not read.

The CVMRCDS buffer list used in CVAFDIR macro can be passed to another CVAF macro call through the MAPRCDS keyword.

If MAPRCDS=YES is coded for a nonindexed VTOC, the function is performed, but an error code is returned.

MAPRCDS=NO

If MAPRCDS=NO is coded, all the buffers without the skip bit on in the buffer list whose address is in the CVMRCDS field of the CVPL are freed. If all the buffers are freed, the buffer list is also freed.

MAPRCDS=(NO,*addr*)

frees buffer lists and buffers previously obtained by CVAF.

You must free buffer lists and buffers obtained by CVAF. This can be done in one of three ways:

- By coding MAPRCDS=NO on the CVAFDIR macro that obtained the buffers
- By coding MAPRCDS=NO on a subsequent CVAF macro
- By coding CVAFDIR ACCESS=RLSE and providing the address of the buffer list in the BUFLIST keyword.

Note: You must enqueue the VTOC and reserve the unit to maintain the integrity of MAP records read.

IXRCDS: Retain VIERS in Virtual Storage

This keyword applies to indexed VTOCs only.

IXRCDS=KEEP

specifies that VIERs read into storage are to be kept in virtual storage. The VIERs are retained even if processing cannot complete successfully. The CVAF parameter list in field CVIRCDS contains the address of a buffer list with the VIR buffer addresses and RBAs of the VIERs read.

The index search function dynamically updates the buffer list and, when necessary, obtains additional buffer lists and chains them together.

If IXRCDS=KEEP is specified and no buffer list is supplied to CVAF in the CVPL, CVAF obtains a buffer list and buffers and reads the first high-level VIER. The address of the buffer list is placed in the CVMICDS field of the CVPL.

The buffer list and VIR buffers are in your protect key. The subpool is 0 if you are not authorized; it is subpool 229 if you are.

If IXRCDS=KEEP is coded for a nonindexed VTOC, a request to read or write a DSCB is performed, but an error code is returned.

When processing on the current volume is finished, release all areas that were kept.

IXRCDS=(KEEP,*addr*)
The index records buffer list address from one CVAF request is being passed to this CVAF parameter list by specifying its address as the second parameter in the IXRCDS keyword.

IXRCDS=NOKEEP
If IXRCDS=NOKEEP is coded, the VIERs that are accessed (if any) are not retained. Furthermore, the buffer list supplied in the CVIRCDS field in the CVAF parameter list is released, as are all buffers found in the buffer list. If the skip bit is set in any entry in the buffer list, the buffer and buffer list are not freed. This is the default.

IXRCDS=(NOKEEP,*addr*)
specifies that previously accessed VIERs are not to be retained.

You must free buffer lists and buffers obtained by CVAF. This can be done in one of three ways:

- By coding IXRCDS=NOKEEP on the CVAFDIR macro that obtained the buffers
- By coding IXRCDS=NOKEEP on a subsequent CVAF macro
- By coding CVAFDIR ACCESS=RLSE and providing the address of the buffer list in the BUFLIST keyword.

Note: You must enqueue the VTOC and reserve the unit to maintain the integrity of the VIERs read.

BRANCH: Specify the Entry to the Macro

BRANCH=(YES,SUP)
requests the branch entry. Your program be in supervisor state. Protect key checking is bypassed.

If BRANCH=YES is coded, an 18-word save area must be supplied. No lock can be held on entry to CVAF. SRB mode is not allowed.

BRANCH=YES
Equivalent to BRANCH=(YES,SUP), because SUP is the default when YES is coded. Protect key checking is bypassed.

BRANCH=(YES,PGM)
requests the branch entry. Your program be authorized by APF and be in problem state. Protect key checking is bypassed.

BRANCH=NO
requests the SVC entry. If any output operations are requested, your program must be authorized by APF. Protect key checking is performed. This is the default.

MF: Specify the Form of the Macro

This keyword specifies whether the list, execute, or normal form of the macro is requested.

MF=I
If I is coded or if neither L nor E is coded, the CVAF parameter list is generated and CVAF is called. This is the normal form of the macro.

MF=L
indicates the list form of the macro. A parameter list is generated, but CVAF is not called.

MF=(E,*addr*)
indicates the execute form of the macro. The CVAF parameter list whose address is in *addr* can be modified by this form of the macro.

Return Codes from CVAFDIR

On return from CVAF, register 1 contains the address of the CVPL (CVAF parameter list), and register 15 contains one of the following return codes:

Return Code	Meaning
0 (X'00')	The request was successful. However, if the CVAFDIR request is to read or write a DSCB and a VTOC index structure error is encountered, the CVSTAT field indicates the structure error that was encountered. (CVSTAT code descriptions are in “VTOC Index Error Message and Associated Codes” on page 360.)
4 (X'04')	An error occurred. The CVSTAT field in the CVPL contains an indication of the cause of the error. (CVSTAT code descriptions are in “VTOC Index Error Message and Associated Codes” on page 360.)
8 (X'08')	Invalid VTOC index structure while processing a request to read or write a VTOC index record. The CVSTAT field in the CVPL contains an indication of the cause of the error. (CVSTAT code descriptions are in “VTOC Index Error Message and Associated Codes” on page 360.)
12 (X'0C')	The CVAF parameter list is not in your protect key or is not valid (the ID is not valid, or the length field is incorrect, or the CVFCTN (function code) field is not valid or is not supported in this release). The CVPL has not been modified.
16 (X'10')	An I/O error was encountered.

Example of Using the CVAFDIR Macro with a VTOC

This example uses the CVAFDIR macro to read a DSCB of a given data set name and determines whether the DSCB is for a partitioned data set. The address of the 44-byte data set name is supplied to the program in register 5 (labeled RDSN in the example). The address of a DEB open to the VTOC is supplied to the program in register 4 (labeled RDEB in the example).

The buffer list is in the program and is generated by the ICVAFBFL macro. The DSCB buffer is in the program and is generated by the IECSDSL1 macro.

```
DIRXMP1  CSECT
         STM   14,12,12(RSAVE)
         BALR  12,0
         USING *,12
         ST    RSAVE,SAVEAREA+4
         LA    RWORK,SAVEAREA
         ST    RWORK,8(,RSAVE)
         LR    RSAVE,RWORK
***********************************************************
*
*        REGISTERS
*
***********************************************************
REG1     EQU   1                  REGISTER 1
RWORK    EQU   3                  WORK REGISTER
RDEB     EQU   4                  DEB ADDRESS
RDSN     EQU   5                  ADDRESS OF DATA SET NAME
RSAVE    EQU   13                 SAVE AREA ADDRESS
REG15    EQU   15                 RETURN CODE REGISTER 15
***********************************************************
*
*        RETURN CODES
*
***********************************************************
PDSRTN   EQU   0                  DATA SET A PDS RETURN CODE
NOTFND   EQU   4                  DATA SET NOT FOUND RETURN CODE
NOTPDS   EQU   8                  DATA SET NOT A PDS RETURN CODE
UNEXPECD EQU   12                 UNEXPECTED ERROR RETURN CODE
***********************************************************
*
*        READ DSCB INTO DS1FMTID.
*        DATA SET NAME ADDRESS SUPPLIED IN RDSN.
*        ADDRESS OF DEB OPEN TO VTOC SUPPLIED IN RDEB.
*        DETERMINE IF DATA SET IS A PARTITIONED DATA SET.
*        THIS PROGRAM IS NEITHER REENTRANT NOR REUSABLE.
*
***********************************************************
         XC   BUFLIST(BFLHLN+BFLELN),BUFLIST ZERO BUFFER LIST
         OI   BFLHFL,BFLHDSCB      DSCBS TO BE READ WITH BUFFER LIST
         MVI  BFLHNOE,1            ONE BUFFER LIST ENTRY
         LA   RWORK,DS1FMTID       ADDRESS OF DSCB BUFFER
         ST   RWORK,BFLEBUF        PLACE IN BUFFER LIST
         OI   BFLEFL,BFLECHR       CCHHR OF DSCB RETURNED BY CVAF
         MVI  BFLELTH,DSCBLTH      DATA PORTION OF DSCB READ - DSN     *
                                   SUPPLIED IN CVPL
         MVC  DS1DSNAM,0(RDSN)     MOVE IN DATA SET NAME TO WORKAREA
         CVAFDIR ACCESS=READ,DSN=DS1DSNAM,BUFLIST=BUFLIST,DEB=(RDEB)
         USING CVPL,REG1           ADDRESSABILITY TO CVPL
         LTR  REG15,REG15          ANY ERROR
         BZ   NOERROR              BRANCH IF NOT
```

Figure 121 (Part 1 of 2). Example of CVAFDIR Macro with VTOC

```
*********************************************************
*
*        DETERMINE WHAT ERROR IS
*
*********************************************************
         C    REG15,ERROR4         IS RETURN CODE 4
         BNE  OTHERERR             BRANCH IF NOT 4
         CLI  CVSTAT,STAT001       IS IT DATA SET NAME NOT FOUND?
         BNE  OTHERERR             BRANCH IF NOT
         DROP REG1                 ADDRESSABILITY TO CVPL NOT NEEDED
*********************************************************
*
*        DATA SET NAME NOT FOUND
*
*********************************************************
         L    RSAVE,4(,RSAVE)
         RETURN (14,12),RC=NOTFND  SET UP DATA SET NOT FOUND ERROR
NOERROR  EQU  *                    DSCB READ
         MVC  F1CCHHR,BFLEARG      MOVE CCHHR OF FORMAT 1/4 DSCB TO    *
                                   WORKAREA
         CLI  DS1FMTID,C'4'        IS DSCB A FORMAT 4 DSCB
         BE   NOTF1                BRANCH IF YES. NOT A FORMAT 1
         TM   DS1DSORG,DS1DSGPO    IS FORMAT 1 DSCB FOR PARTITIONED    *
                                   DATA SET
         BO   PDS                  BRANCH IF PDS
NOTF1    EQU  *                    DSCB IS NOT A PDS
         L    RSAVE,4(,RSAVE)
         RETURN (14,12),RC=NOTPDS  SET UP NOT PDS RETURN CODE
PDS      EQU  *                    DATA SET IS PARTITIONED
         L    RSAVE,4(,RSAVE)
         RETURN (14,12),RC=PDSRTN  SET UP PDS RETURN CODE
OTHERERR EQU  *                    UNEXPECTED ERROR
         L    RSAVE,4(,RSAVE)
         RETURN (14,12),RC=UNEXPECD

ERROR4   DC   F'4'                 ERROR RETURN CODE 4
BUFLIST  ICVAFBFL DSECT=NO         BUFFER LIST
         IECSDSL1 (1)              FORMAT 1 DSCB DATA SET NAME AND     *
                                   BUFFER
DSCBLTH  EQU  *-IECSDSL1-L'DS1DSNAM LENGTH OF DATA PORTION OF DSCB
F1CCHHR  DS   XL5                  CCHHR OF DSCB
SAVEAREA DS   18F                  SAVE AREA
CVPL     ICVAFPL ,                 CVPL MAPPING MACRO

         END
```

Figure 121 (Part 2 of 2). Example of CVAFDIR Macro with VTOC

Example of Using the CVAFDIR Macro with an Indexed VTOC

This example uses the CVAFDIR macro to read one or more DSCBs from a VTOC. The UCB is supplied to the program in register 4 (labeled RUCB). The TTR of each DSCB read is to be returned to the caller. This program must be APF authorized.

The address of a parameter list is supplied to the program in register 5 (labeled RLIST). The parameter list contains one or more 3-word entries. The format of each 3-word entry is mapped by the LISTMAP DSECT. The first word contains the

address of the data set name of the DSCB to be read. The second word contains the address of the 96-byte buffer into which the DSCB is to be read. The third word contains the address of the 3-byte TTR of the DSCB read.

The CVPL is generated by a list form of the CVAFDIR macro at label CVPL. The BUFLIST, IXRCDS, IOAREA, and BRANCH keywords are coded on the list form of the macro. IXRCDS=KEEP and IOAREA=KEEP are coded to avoid overhead if two or more DSCBs are to be read. BRANCH=(YES,PGM) is coded in the list form of the CVAFDIR macro to cause the CVPL to have the CV1PGM bit set to one; this indicates to CVAF that the caller is authorized by APF and not in supervisor state. The execute forms of the CVAFDIR macro then specify BRANCH=YES, and not BRANCH=(YES,PGM), because the CV1PGM bit is set in the list form of the macro.

The CVAFDIR macro with ACCESS=RLSE is coded before the program exits to release the CVAF I/O area and the index records buffer list. BUFLIST=0 is coded because no user-supplied buffer list is to be released; BUFLIST was coded on the list form of the CVAFDIR macro and, therefore, is in the CVBUFL field of the CVPL. This field must be set to zero for the release function.

```
DIRXMP2  CSECT
         STM   14,12,12(13)
         BALR  12,0
         USING *,12
         ST    13,SAVEAREA+4
         LA    RWORK,SAVEAREA
         ST    RWORK,8(,13)
         LR    13,RWORK
************************************************************
*
*        REGISTERS
*
************************************************************
RWORK    EQU   3                  WORK REGISTER
RUCB     EQU   4                  UCB ADDRESS SUPPLIED BY CALLER
RLIST    EQU   5                  ADDRESS OF PARAMETER LIST
RDSN     EQU   6                  ADDRESS OF DATA SET NAME
RTTR     EQU   7                  ADDRESS OF TTR
REG15    EQU   15                 RETURN CODE REGISTER 15
************************************************************
*
*        READ DSCB OF DATA SET NAME SUPPLIED. RETURN TTR OF DSCB.
*        UCB ADDRESS SUPPLIED IN RUCB.
*        ADDRESS OF PARAMETER LIST IN RLIST.
*          WORD 1 OF PARAMETER LIST = ADDRESS OF DATA SET NAME
*          WORD 2 OF PARAMETER LIST = ADDRESS OF DSCB TO BE RETURNED
*          WORD 3 OF PARAMETER LIST = ADDRESS OF TTR TO BE RETURNED
*                    WORDS 1-3 DUPLICATED WITH THE HIGH ORDER BIT OF
*                    WORD 3 SET TO ONE FOR LAST ENTRY.
*
************************************************************
         USING LISTMAP,RLIST       ADDRESSABILITY TO PARMLIST
TOPLOOP  EQU  *                    LOOP FOR EACH DSCB
         XC   BUFLIST(BFLHLN+BFLELN),BUFLIST ZERO BUFFER LIST
         OI   BFLHFL,BFLHDSCB      DSCBS TO BE READ WITH BUFFER LIST
         MVI  BFLHNOE,1            ONE BUFFER LIST ENTRY
         LA   RWORK,LISTDSCB       ADDRESS OF DSCB BUFFER
         ST   RWORK,BFLEBUF        PLACE IN BUFFER LIST
         OI   BFLEFL,BFLETTR       TTR OF DSCB RETURNED BY CVAF
         MVI  BFLELTH,DSCBLTH      DATA PORTION OF DSCB READ - DSN     *
                                   SUPPLIED IN CVPL
         L    RDSN,LISTDSN         ADDRESS OF DATA SET NAME
         CVAFDIR DSN=(RDSN),UCB=(RUCB),MF=(E,CVPL),BRANCH=YES
         L    RTTR,LISTTTR         ADDRESS OF TTR TO BE RETURNED
         USING TTRMAP,RTTR         MAP OF TTR
         LTR  REG15,REG15          ANY ERROR
         BZ   NOERROR              BRANCH IF NOT
         XC   TTR,TTR              ZERO TTR INDICATING NO DSCB
         B    RELOOP               GET NEXT ENTRY
NOERROR  EQU  *                    DSCB READ
         MVC  TTR,BFLEARG          RETURN TTR OF DSCB
```

Figure 122 (Part 1 of 2). Example of Using the CVAFDIR Macro with an Indexed VTOC

```
RELOOP   EQU  *                    GET NEXT ENTRY
         TM   LASTLIST,LASTBIT     IS IT LAST ENTRY IN LIST?
         LA   RLIST,NEXTLIST       GET NEXT ENTRY
         BZ   TOPLOOP              PROCESS NEXT LIST
         CVAFDIR ACCESS=RLSE,      RELEASE CVAF OBTAINED AREAS         *
               IOAREA=NOKEEP,      RELEASE IOAREA                      *
               IXRCDS=NOKEEP,      RELEASE VIER BUFFER LIST            *
               BUFLIST=0,          NO USER BUFFER LIST SUPPLIED TO RLSE*
               BRANCH=YES,         BRANCH ENTER CVAF                   *
               MF=(E,CVPL)
         L     13,SAVEAREA+4
         RETURN (14,12)

BUFLIST  ICVAFBFL DSECT=NO         BUFFER LIST

SAVEAREA DS    18F                 REGISTER SAVE AREA
LISTMAP  DSECT
LISTDSN  DS    F                   ADDRESS OF DATA SET NAME
LISTDSCB DS    F                   ADDRESS OF BUFFER FOR DSCB TO BE    *
                                   RETURNED
LISTTTR  DS    0F                  ADDRESS OF TTR OF DSCB TO BE        *
                                   RETURNED
LASTLIST DS    X                   FIRST BYTE
LASTBIT  EQU   X'80'               LAST ENTRY IN LIST
         DS    XL3                 REMAINDER OF TTR ADDRESS
NEXTLIST EQU   *                   NEXT LIST
DSCB     DSECT
         IECSDSL1 (1)
DSCBLTH  EQU   *-DSCB-L'DS1DSNAM   LENGTH OF DATA PORTION OF DSCB
TTRMAP   DSECT
TTR      DS    XL3                 TTR TO BE RETURNED
DIRXMP2  CSECT
CVPL     CVAFDIR ACCESS=READ,BUFLIST=BUFLIST,MF=L,                     *
               IOAREA=KEEP,        KEEP IOAREA TO AVOID OVERHEAD       *
               IXRCDS=KEEP      KEEP VIERS FOR 2ND AND SUBSEQUENT CALLS*
                                   CALLED IN PROGRAM STATE BUT APF     *
                                   AUTHORIZED SO UCB IS SUPPLIED
         ORG  CVPL                 OVERLAY CVPL WITH EXPANSION OF MAP
CVPLMAP  ICVAFPL DSECT=NO

         END
```

Figure 122 (Part 2 of 2). Example of Using the CVAFDIR Macro with an Indexed VTOC

CVAFDSM Macro Overview and Specification

The CVAFDSM macro is used to obtain volume information for an indexed or non-indexed VTOC.

The CVAFDSM macro can be used for an indexed VTOC to obtain:

- One or more extents that describe unallocated space on the volume
- A count of free DSCBs on the VTOC
- A count of free VTOC index records in the VTOC index.

The CVAFDSM macro can be used for an nonindexed VTOC to obtain:

- One or more extents that describe unallocated space on the volume

The format of the CVAFDSM macro is:

```
►►──┬──────────┬──CVAFDSM──ACCESS=MAPDATA──,MAP=──┬─INDEX──┬──────────────►
    └─label────┘                                  ├─VOLUME─┤
                                                  └─VTOC───┘

►──┬─────────────────┬──┬──────────────────────────────────────────────────►
   └─,EXTENTS=addr───┘  └─,RTA4BYTE=──┬─NO──┬──┘
                                      └─YES─┘

►──┬──────────────────────────────────┬──┬──────────────────────────────────►
   └─,MAPRCDS=──┬─YES──(1)──────────┬──┘  ├─,UCB=(ucbaddr)──┬──┘
                ├─(YES,addr)────────┤     └─,DEB=addr─────────┘
                ├─NO──(2)───────────┤
                └─(NO,addr)─────────┘

►──┬──────────────────────────┬──┬──────────────────────────────────────────►
   └─,COUNT=──┬─NO──┬──┘  └─,CTAREA=addr──┘
              └─YES─┘

►──┬──────────────────────────────────┬──┬──────────────────────────────────►
   └─,IOAREA=──┬─NOKEEP──────────────┬──┘  └─,BRANCH,──┬─NO──────────────┬──┘
               ├─KEEP────────────────┤                 ├─YES──(3)────────┤
               ├─(KEEP,addr)─────────┤                 ├─(YES,SUP)───────┤
               └─(NOKEEP,addr)───────┘                 └─(YES,PGM)───────┘

►──┬──────────────────────┬──►◄
   └─,MF=──┬─I──────────┬──┘
           ├─L──────────┤
           └─(E,addr)───┘
```

Notes:

1. Default if MF=I.
2. Default if MF=L or MF=(E, addr).
3. If YES is coded, the default is SUP.

ACCESS: Request Information from Index Space Maps or the VTOC

ACCESS=MAPDATA

obtains data from index space maps or free space DSCBs.

The following data is available from the index space maps:

- The number of format-0 DSCBs (the data is obtained from the VTOC map of DSCBs)
- The number of unallocated VIRs in the index (the data is obtained from the VTOC index map)
- The number (and location) of extents of unallocated pack space (the data is obtained from the VTOC pack space map).

The following data is available from nonindexed VTOCs.

- The number (and location) of extents of unallocated pack space.

MAP: Identify the Map to Be Accessed

MAP=INDEX

specifies that the VTOC index map (VIXM) is to be accessed and a count of unallocated VIRs returned. COUNT=YES must also be coded.

MAP=VOLUME

for indexed VTOCs, specifies that the VTOC pack space map (VPSM) is to be accessed and information on unallocated extents of pack space returned.

For nonindexed VTOCs, specifies that the information from the free space DSCBs on unallocated extents is to be returned.

For indexed and nonindexed VTOCs, EXTENTS=*addr* and COUNT=NO must also be coded.

MAP=VTOC

specifies that the VTOC map of DSCBs (VMDS) is to be accessed and a count of format-0 DSCBs returned. COUNT=YES must also be coded.

EXTENTS: Storage Area Where Extents Are Returned

EXTENTS=*addr*

specifies the address where extent information is to be returned. You specify this parameter only when you also specify MAP=VOLUME to request that unallocated space information from the volume is to be returned.

When RTA4BYTE=YES is specified:

- Information about free extents uses mapping macro ICVEDT02 (see “Using Macro ICVEDT02 to Map the Extents Area” on page 45 for ICVEDT02 format).
- EXTENTS= specifies the address of a control block (ICVEDT02) used to pass 4 byte relative track addresses of the unallocated space. You provide this storage, and you must use this mapping macro to initialize it.

  Prior to calling CVAF, initialize the ICVEDT02 control block as follows:

  – Place “ICVEDT02” in the first 8 bytes (DT2X7EYE).

  – Place the total area length, in bytes, in DT2X7LEN. This value is 36 + (8 * the value in DT2X7ENT).

  – Set the control block level number to “1” (DT2X7LEV). (Note that this value can be different in a possible future release, if IBM makes changes that effect the control block.)

  – Set the remaining fields in the control block static area DT2X7FLG through DT2X7RE2 to zero prior to calling CVAF for the first time. Leave these fields unchanged from the way the previous call returned them, when calling for additional extents.

  – Set DT2X7ENT to the total number of extent descriptor entries that will fit in the storage you provide.

  – Place the relative track address (RTA) at which CVAF should start the search into the first four bytes of the first extent area DT2RTAST(1).

    CVAF updates the first extent entry with information about the next free extent found that has a higher starting RTA than that provided. Each subsequent extent entry is filled in with information about additional free space extents (in ascending RTA order).

For all calls, if all the unallocated extents from the volume are returned before the provided storage area is filled, the remaining entries are set to zero. CVAF will now set return code 4 in register 15, and will set the CVSTAT field to X'20' to indicate end of data.

If the provided storage area is not large enough to hold all the unallocated extents, CVAF will not set return code 4 in register 15. In this case, you can call CVAF again to get the remaining unallocated space information. Do NOT modify ANY header information in ICVEDT02, as CVAF can have saved internal use restart information there. Instead, copy the last ending RTA+1 (DT2RTAED) returned from the previous CVAF search into DT2RTAST.

When RTA4BYTE=NO is specified or defaulted:

- Information about free extents has the format of *XXYYZ* (see “RTA4BYTE: Specify the Type of Extent Area Used” for *XXYYZ* format).
- EXTENTS= is the address of a 1-byte count field containing the number of 5-byte entries that follow. You provide this storage area. The length of the area, in bytes, is 1 + (*count* * 5), where *count* is the value of the first byte of the area. The first two bytes (“XX”) of the first 5-byte extent area entry, is the relative track address (RTA) at which CVAF will start the search. CVAF updates the first entry with information about the next free extent found that has a higher starting RTA than that supplied. Each subsequent entry is filled in with information about additional free extents (in ascending track address order).
- CVAF can be called multiple times, as needed, to retrieve more extents than the area can hold in a single call. The first extent returned is the first free extent after the relative track address (“XX”) recorded in the first extent (*XXYYZ*) in the area.
- To retrieve the first free extent on the volume, set “XX” in the first entry to zero. When calling additional times, set “XX” in the first entry to the LAST relative track address returned by the previous call.

RTA4BYTE: Specify the Type of Extent Area Used

RTA4BYTE=YES

specifies that the extents area contains pairs of addresses in the format RTA RTA+1, where RTA is the four byte relative address of the first track of the extent, and RTA+1 is the four byte relative address of the last track of the extent plus 1.

You must use the macro ICVEDT02 to map the extent area if you specify RTA4BYTE=YES. See Figure 22 on page 45 for a description.

RTA4BYTE=YES is a required parameter for an nonindexed VTOC, and optional for an indexed VTOC.

RTA4BYTE=NO

specifies that the extents area is in the format *XXYYZ* where:

XX	is the relative track address of the first track of the extent
YY	is the number of whole cylinders in the extent
Z	is the number of additional tracks in the extent.

If you do not specify the RTA4BYTE parameter, the default is RTA4BYTE=NO.

MAPRCDS: Keep or Free MAPRCDS Buffer List and Buffers

MAPRCDS=YES

specifies that the buffer list and buffers are to be retained at the end of the function.

If MAPRCDS=YES is specified and no buffer list is supplied through the CVAF parameter list, CVAF reads the VIRs into buffers obtained by CVAF. The buffer list that contains the address and RBAs of the VIRs can be accessed after the CVAF call from the CVAF parameter list field, CVMRCDS. The buffer list and VIR buffers are in the caller's protect key: subpool 0 if the caller is not authorized; subpool 229 if the caller is authorized.

MAPRCDS=YES is the default if MF=I is specified or defaulted.

When processing on the current volume is finished, release all areas that were kept.

MAPRCDS=(YES,*addr*)

If MAPRCDS=YES is coded, but the buffer list address (CVMRCDS in CVAF parameter list) is supplied, the VIRs are not read.

The CVMRCDS buffer list from one CVAF call can be passed to another CVAF macro call through the MAPRCDS keyword.

MAPRCDS=NO

specifies that the MAP records buffers and buffer list are freed upon completion of the CVAFDSM function.

NO is the default if MF=L is specified.

MAPRCDS=(NO,*addr*)

frees buffer lists and buffers previously obtained by CVAF.

Buffer lists and buffers obtained by CVAF must be freed by the caller. This can be done in one of the following ways:

- By coding MAPRCDS=NO on the call that obtained the buffers
- By coding MAPRCDS=NO on a subsequent CVAF call
- By coding CVAFDIR ACCESS=RLSE and providing the buffer list in the BUFLIST keyword.

  If MF=(E,*addr*) is coded and MAPRCDS is not coded, the parameter list value of MAPRCDS is not changed.

Note: You must enqueue the VTOC and reserve the unit to maintain the integrity of the MAP records read.

UCB or DEB: Specify the VTOC to Be Accessed

UCB= *rs-type or (2-12)* *standard form* **UCB**= *rx-type or (2-12)* *execute form*

specifies the address of the UCB for the VTOC to be accessed. The UCB address can be for a captured UCB, or for an actual UCB above or below the 16MB line. Use the address of a UCB, not a UCB copy. An unauthorized caller can not use this parameter. If your program is in 31-bit mode, this address must be in 31-bit address; the high order byte is part of the address. You should not code the UCB parameter with MF=L.

DEB=*addr*

supplies the address of a DEB opened to the VTOC you want to access. CVAF does not allow output requests to the VTOC or VTOC index if you specify the

DEB subparameter. Without authorization, you cannot perform any asynchronous activity (such as EXCP, CLOSE, EOV) against the data set represented by the DEB because CVAF removes the DEB from the DEB table for the duration of the CVAF call. If you are not authorized (neither APF authorized nor in a system key), specify a DEB address, not a UCB, to CVAFDSM. See “Identifying the Volume” on page 41 for further details.

If you supply a previously obtained I/O area through the IOAREA keyword, neither UCB nor DEB need be supplied. Otherwise, supply either a UCB or DEB. If you supply a UCB address, it is overlaid in the CVPL by the UCB address in the I/O area. If you supply both the UCB and the DEB addresses in the CVPL, the DEB address is used and the UCB address in the CVPL is overlaid by the UCB address in the DEB.

COUNT: Obtain a Count of Unallocated DSCBs or VIRs

COUNT=YES

indicates that a count of unallocated DSCBs or VIRs in the designated space map is requested. MAP=VTOC or MAP=INDEX must be specified if COUNT=YES is coded.

COUNT=NO

indicates that a count of unallocated DSCBs or VIRs is not desired but, rather, information on free space on the pack is desired. MAP=VOLUME must be coded if COUNT=NO is coded or the default.

CTAREA: Supply a Field to Contain the Number of Format-0 DSCBs

CTAREA=*addr*

supplies the address of a 4-byte field to contain the number of format-0 DSCBs when COUNT=YES, MAP=VTOC is specified; or the number of unallocated VIRs in the VTOC index when COUNT=YES, MAP=INDEX is specified.

IOAREA: Keep or Free the I/O Work Area

IOAREA=KEEP

specifies that the CVAF I/O area associated with the CVAF parameter list is to be kept upon completion of the CVAF request. IOAREA=KEEP can be coded with BRANCH=NO only if the caller is authorized (APF or system key).

If IOAREA=KEEP is coded, the caller must call CVAF with IOAREA=NOKEEP specified at some future time, whether or not any further VTOC access is required. An example of such a caller is the recovery routine of the caller of CVAF.

Coding IOAREA=KEEP allows subsequent CVAF requests to be more efficient, as certain initialization functions can be bypassed. Neither DEB nor UCB need be specified when a previously obtained CVAF I/O area is supplied; neither can they be changed.

When IOAREA=KEEP is first issued, CVAF returns the CVAF I/O area in the CVAF parameter list (CVIOAR). Subsequent calls of CVAF can use that same parameter list, and CVAF obtains its I/O area from the CVIOAR.

When processing on the current volume is finished, release all areas that were kept.

IOAREA=(KEEP,*addr*)
supplies the address of a previously obtained I/O area. If a different CVAF parameter list is used, the previously obtained CVAF I/O area can be passed to CVAF by coding its address as the second parameter of the IOAREA keyword.

IOAREA=NOKEEP
causes the work area to be freed upon completion of the CVAF request. This is the default.

IOAREA=(NOKEEP,*addr*)
causes a previously obtained work area to be freed upon completion of the CVAF request.

BRANCH: Specify the Entry to the Macro

BRANCH=(YES,SUP)
requests the branch entry. The caller must be in supervisor state. Protect key checking is bypassed.

If BRANCH=YES is coded, an 18-word save area must be supplied. No lock can be held on entry to CVAF. SRB mode is not allowed.

BRANCH=YES
Equivalent to BRANCH=(YES,SUP), because SUP is the default when YES is coded. Protect key checking is bypassed.

BRANCH=(YES,PGM)
requests the branch entry. The caller must be APF authorized and in problem state. Protect key checking is bypassed.

BRANCH=NO
requests the SVC entry. The caller must be APF authorized if any output operations are requested. Protect key checking is performed. This is the default.

MF: Specify the Form of the Macro

This keyword specifies whether the list, execute, or normal form of the macro is requested.

MF=I
If I is coded or if neither L nor E is coded, the CVAF parameter list is generated, as is code, to call CVAF. This is the default.

MF=L
indicates the list form of the macro. A parameter list is generated, but code to call CVAF is not generated.

MF=(E,*addr*)
indicates the execute form of the macro. The remote CVAF parameter list supplied as *addr* is used in, and can be modified by, the execute form of the macro.

Return Codes from CVAFDSM

On return from CVAF, register 1 contains the address of the CVPL (CVAF parameter list), and register 15 contains one of the following return codes:

Return Code	Meaning
0 (X'00')	The request was successful.

Return Code	Meaning
4 (X'04')	End of data (CVSTAT is set to decimal 32), or an error was encountered. The CVSTAT field in the CVPL contains an indication of the cause of the error. (CVSTAT code descriptions are in "VTOC Index Error Message and Associated Codes" on page 360)
8 (X'08')	Invalid VTOC index structure. CVSTAT contains an indication of the cause of the error. (CVSTAT code descriptions are in "VTOC Index Error Message and Associated Codes" on page 360)
12 (X'0C')	One of the following • The CVAF parameter list is not in your protect key. • The CVAF parameter list protect key is invalid. • The CVAF parameter list ID is invalid. • The CVAF parameter list length is incorrect. • The CVAF parameter list has not been modified. • The function code (CVFCTN) field is not valid. • The function code (CVFCTN) field is not supported by this release.
16 (X'10')	An I/O error was encountered.

CVAFFILT Macro Overview and Specification

You can use the CVAFFILT macro to invoke the CVAF filter service. You can also use it to map or initialize the CVAF parameter list (CVPL). CVAF filter retrieves data set DSCB chains from an indexed or nonindexed VTOC and places them in buffers you provide. You can request the DSCBs for a single partially-qualified data set name or for a list of fully-qualified data set names.

Identify a specific DASD device and provide both a filter criteria list (FCL) defining the request, and a CVAF buffer list (with buffers) for DSCB return. The format of the two elements of the FCL is shown in Figure 24 on page 50 and Figure 25 on page 52. The format of the buffer list is shown in "Using Buffer Lists" on page 43. CVAFFILT returns a complete set of DSCBs in the order that they are chained in the VTOC (format-1, format-2, then format-3).

Note: Keywords coded on the list form of the macro need not be coded on the execute form. Keywords coded on one CVAFFILT call remain in effect for subsequent calls unless overridden, if you use the same CVAFFILT parameter list.

See "Reading Sets of DSCBs with CVAF Filter" on page 49 for additional information.

The format of the CVAFFILT macro is:

```
►►──┬──────────┬──CVAFFILT──┬──────────────────────────────────────────►
    └─label────┘            └─,ACCESS=─┬─READ──────┬─┘
                                       ├─RESUME────┤
                                       └─RLSE──────┘

►──┬──────────────────────────────┬──┬──────────────────────────────►
   └─,BUFLIST=─┬─addr──────┬─┘      ├─,UCB=(ucbaddr)──┤
               └─(reg)─────┘        └─,DEB=addr───────┘

►──┬──────────────────────────┬──┬──────────────────────────────────────────►
   └─,FCL=─┬─addr──────┬─┘      └─,FLTAREA=─┬─NOKEEP──────────────────────┬─┘
           └─(reg)─────┘                    ├─KEEP────────────────────────┤
                                            ├─(KEEP,─┬─addr──────┬─)─────┤
                                            │        └─(reg)─────┘       │
                                            └─(NOKEEP,─┬─addr──────┬─)──┘
                                                       └─(reg)─────┘

►──┬──────────────────────────────────────────────────────────────────────►
   └─,IOAREA=─┬─NOKEEP──────────────────────┬─┘
              ├─KEEP────────────────────────┤
              ├─(KEEP,─┬─addr──────┬─)─────┤
              │        └─(reg)─────┘       │
              └─(NOKEEP,─┬─addr──────┬─)──┘
                         └─(reg)─────┘

►──┬──────────────────────────────────────────────────────────────────────┬──┬──────────────────────────────────────────────────────────────────────┬──►◄
   └─,BRANCH=─┬─NO────────────────────────┬─┘  └─,MF=─┬─I──────────────────────────┬─┘
              ├─YES───────────────────────┤            ├─D──────────────────────────┤
              └─(YES,─┬─SUP─┬─)──────────┤            ├─L──────────────────────────┤
                      └─PGM─┘            │            └─(E,─┬─addr──────┬─)─────────┤
                                         └──────────────────└─(reg)─────┘
```

Note: For the first operand following CVAFFILT, do not code the leading comma.

Control Block Address Resolution:Keyword=*addr or (reg)*

You, as the caller, either define or reference the control blocks needed by CVAF filter. Caller-defined control blocks are: BUFLIST, CVPL, and FCL. Caller-referenced control blocks are: DEB, FLTAREA, IOAREA, and UCB. The CVAFFILT macro generates different instructions for **keyword**=*addr* and **keyword**=(*reg*) depending upon whether you are specifying a defined or referenced control block.

1. When you specify any control block's address as (*reg*), the CVAFFILT macro assumes that the register specified contains that address.
2. When you specify a “defined” control block's address as *addr*, the CVAFFILT macro assumes that the specified location is that of the control block itself. The macro generates a load address instruction (LA) to obtain the control block's address.
3. When you specify a “referenced” control block's address as *addr*, the CVAFFILT macro assumes that the specified location is that of a word containing the address of the control block. The macro generates a load instruction (L) to obtain the control block's address.

ACCESS: Retrieve a DSCB, or Release FLTAREA and/or IOAREA

ACCESS=READ

retrieves all DSCBs associated with the data set names specified in the filter criteria list (FCL), placing them in your buffers. You can select (filter) the retrieved DSCBs by providing either a list of one or more fully-qualified names, or a single partially-qualified name, using single or double asterisk notation. (See the example of partially-qualified names in "Partially-Qualified Names for CVAFFILT" on page 342.)

If the number of buffers is not large enough to hold all the requested DSCBs, CVAFFILT indicates this in the CVSTAT status byte of the CVAF parameter list (CVPL). You can resume the READ function by issuing a call with ACCESS=RESUME. See "Codes Put in the CVSTAT Field" on page 361.

When selecting DSCBs by partially-qualified name, CVAFFILT uses only the first data set name in the FCL list. Set the FCLCOUNT count field in the FCL to 1 or CVAFFILT returns error code 63 in the CVSTAT status byte of the CVPL. The DSCBs returned by CVAFFILT might not be in sequence by data set name; however, the DSCBs for each data set are always in order (format-1, format-2, format-3).

When selecting DSCBs by fully-qualified names, you can request that CVAF filter return the DSCBs for the selected data set names in the data set name order implied by the FCL. See the FCL1ORDR flag in Figure 24 on page 50.

Always test the status byte of each data set name in the FCL list to ensure successful completion. (Some error conditions result in failure to return a data set's DSCBs.) See the FCLDSNST byte in Figure 25 on page 52.

ACCESS=RESUME

resumes a previously initiated READ or RESUME function that was terminated because you did not provide enough buffers to contain all the requested DSCBs. For the RESUME function to execute correctly, the keyword FLTAREA=KEEP must be coded in each of the previous READ and RESUME function calls.

ACCESS=RLSE

releases the previously kept filter save area (FLTAREA) and/or CVAF I/O work area (IOAREA).

BUFLIST: Specify a Buffer List

BUFLIST=*addr* or (*reg*)

supplies the address of a buffer list used to read DSCBs. When you specify ACCESS=RLSE, the BUFLIST keyword is required for the standard form of the macro. See the format of the buffer list header and buffer list entry in Figure 20 on page 44 and Figure 21 on page 45, respectively.

UCB or DEB: Specify the VTOC to Be Accessed

UCB= *rs-type or (2-12)* *standard form* **UCB=** *rx-type or (2-12)* *execute form*

specifies the address of the UCB for the VTOC to be accessed. The UCB address might be for a captured UCB, or for an actual UCB above or below the 16MB line. Use the address of a UCB, not a UCB copy. An unauthorized caller must not use this parameter. If your program is in 31-bit mode, this address must be in 31-bit address; the high order byte is part of the address. You should not code the UCB parameter with MF=L.

DEB=*addr or (reg)*
supplies the address of a DEB opened to the VTOC you want to access. If you are not authorized, specify a DEB address, not a UCB, to CVAFFILT; also, without authorization, you cannot perform any asynchronous activity against the data set represented by the DEB (such as EXCP, CLOSE, EOV), because CVAF removes the DEB from the DEB table for the duration of the CVAF call. See “Identifying the Volume” on page 41 for further details.

If you supply a previously obtained I/O area through the IOAREA keyword, neither UCB nor DEB is needed. Otherwise, supply either a UCB or DEB. If you supply a UCB address, it is overlaid in the CVPL by the UCB address in the I/O area. If you supply both the UCB and the DEB addresses in the CVPL, the DEB address is used and the UCB address in the CVPL is overlaid by the UCB address in the DEB.

FCL: Specify a Filter Criteria List

FCL=*addr or (reg)*
supplies the address of a filter criteria list. It is required when ACCESS=READ is specified on the standard form of the macro. The format of the two elements of the filter criteria list is shown in Figure 24 on page 50 and Figure 25 on page 52.

FLTAREA: Keep or Free the Filter Save Area

FLTAREA=KEEP
specifies keeping the filter save area. Code this operand if the RESUME function might be called later (to resume processing prematurely terminated because the number of caller-supplied buffers is not enough to contain all the returned DSCBs).

CVAFFILT returns the address of the kept filter save area in the CVAFFILT parameter list (CVFSA field). If you specify the same parameter list in subsequent RESUME calls, CVAFFILT reuses the same filter save area.

Note: If you code this operand, you must subsequently issue CVAFFILT with ACCESS=RLSE to release the filter save area.

FLTAREA=(KEEP,*addr or (reg)***)**
supplies the address of a previously obtained filter save area. See the description of FLTAREA=KEEP operand for additional concerns.

FLTAREA=NOKEEP
frees the filter save area upon completion of the CVAF request.

FLTAREA=(NOKEEP,*addr or (reg)***)**
frees a previously obtained filter save area upon completion of the CVAF request.

IOAREA: Keep or Free the I/O Work Area

IOAREA=KEEP
specifies keeping the CVAF I/O work area. For authorized callers, CVAFFILT returns the address of the kept I/O work area in the CVAFFILT parameter list (CVIOAR). If you specify the same parameter list in subsequent calls, CVAFFILT reuses the same I/O work area.

Note: If you code this operand, you must subsequently issue CVAFFILT with ACCESS=RLSE to release the I/O work area.

IOAREA=(KEEP,*addr or (reg)*)
supplies the address of a previously obtained filter save area. See the description of IOAREA=KEEP operand for additional concerns.

IOAREA=NOKEEP
frees the filter save area upon completion of the CVAF request.

IOAREA=(NOKEEP,*addr or (reg)*)
frees a previously obtained CVAF I/O work area upon completion of the CVAF request.

BRANCH: Specify the Entry to the Macro

BRANCH=NO
requests the SVC (default) entry. Protect key checking is performed.

BRANCH=YES
Equivalent to BRANCH=(YES,SUP), because SUP is the default when you code YES. You must be in supervisor state. Protect key checking is bypassed.

BRANCH=(YES,SUP)
requests the branch entry. You must be in supervisor state. Protect key checking is bypassed. If you specify BRANCH=YES, supply an 18-word save area. You cannot hold a lock at entry to CVAF. You cannot be in SRB mode.

BRANCH=(YES,PGM)
requests the branch entry. You must be APF authorized and be in problem state. Protect key checking is bypassed.

MF: Specify the Form of the Macro

Specifies whether the DSECT, list, execute, or normal form of the macro is requested. You can be in either 24-bit or 31-bit addressing mode. If you are not authorized, you must pass the address of a DEB built by OPEN. If you are authorized, you can pass either the DEB address or the UCB address. You must ensure that the volume is allocated and will remain mounted (for example, by dynamic allocation).

MF=I
specifies the standard form of the macro. The CVAF parameter list is generated and CVAF is called. The default is MF=I.

MF=D
specifies the DSECT form of the macro. The macro generates a request for the ICVAFPL macro to map the unique CVAF filter CVPL (4-bytes longer than standard CVPL).

MF=L
specifies the list form of the macro. The CVAF parameter list is generated, but CVAF is not called.

MF=(E,*addr or (reg)*)
specifies the execute form of the macro. The CVAF parameter list whose address is in *addr* or *reg* is used. You can modify the parameter list with this form of the macro.

Return Codes from CVAFFILT

CVAF filter service does not issue any messages. Upon return from CVAF, register 1 contains the address of the CVAF parameter list and register 15 contains one of the following return codes:

Return Code	Meaning
0 (X'00')	The request was successful.
4 (X'04')	Logical error; status information in CVSTAT.
8 (X'08')	Invalid VTOC structure.
12 (X'0C')	CVAFFILT parameter list in wrong key, or not valid.
16 (X'10')	I/O error.

Note: CVSTAT in the CVAF parameter list contains the status code. See “Codes Put in the CVSTAT Field” on page 361 for a list of the status codes.

Partially-Qualified Names for CVAFFILT

CVAFFILT supports partially-qualified data set names using single or double asterisk notation and the percent sign as shown in the following text.

- You can use a single asterisk to represent a single qualifier. For example, SYS1.*.LOAD designates any data set with three qualifiers, the first being SYS1, the second being any qualifier, and the third being LOAD.
- You can also use a single asterisk to represent zero or more unspecified characters. For example, LOAD.*LIB designates any data set having only two qualifiers, with LOAD being the first, and the second qualifier ending with the character string LIB (for example, LINKLIB). The asterisk can appear anywhere within the qualifier. You can use two single asterisks in the following way: LOAD.A*B*.LIB. CVAFFILT does not support the use of two or more single asterisks with any other character within a single qualifier (for example, LOAD.B**.LIB is not valid).
- A double asterisk represents a place holder for zero or more qualifiers. For example, SYS1.** designates any data set having SYS1 as its first or only qualifier.
- You can specify a percent sign (%) as part of a partially-qualified name. The percent sign designates any data set whose name matches the partially-qualified name, except for the single character in the position indicated by the percent sign. For example, SYS%.*.LOAD% designates any data set with three qualifiers, the first being any 4-character qualifier beginning with SYS, the second being any qualifier, and the third being any five character qualifier beginning with LOAD.

Example of Using the CVAFFILT Macro

This example uses the CVAFFILT macro to read all format-1, format-2, and format-3 DSCBs from a given VTOC, calculates the total number of DSCBs by format type, and returns the totals to the calling program (the caller of this example program, not the caller of CVAF). The address of a DEB opened to the VTOC is passed to the example program in register 1 (labeled R1 in the example).

The buffer list and filter criteria list are defined in the program. The ICVAFBFL macro generates the buffer list, and the ICVFCL macro generates the filter criteria list.

```
FLTXMP   CSECT
*
* INPUT ... REGISTER 1  - ADDRESS OF A DEB OPENED TO THE VTOC.
*           REGISTER 13 - ADDRESS OF A STANDARD REGISTER SAVE AREA
*           REGISTER 14 - ADDRESS OF THE RETURN POINT WITHIN CALLER
*
* PROCESS . USE THE CVAFFILT MACRO (ACCESS=READ, ACCESS=RESUME, AN D
*           ACCESS=RLSE) TO READ ALL FORMAT 1, 2, AND 3 DSCBS FROM A
*           GIVEN VTOC. IF FILTER SERVICE DETECTS AN ERROR CONDITION,
*           IT RETURNS DIAGNOSTIC INFORMATION FOR DEBUGGING ANALYSIS.
*
* OUTPUT .. REGISTER 1  = ADDRESS OF THE DATA RETURN AREA (SEE LABEL
*                         RET_AREA AT THE END OF THIS LISTING).
*           REGISTER 15 = ZERO IF NO ERRORS WERE ENCOUNTERED.
*                         OTHERWISE, ERROR INFORMATION IS PROVIDED
*                         IN THE DATA RETURN AREA (SEE LABEL RET_ERR).
*
* THE CVAF PARAMETER LIST (CVPL), FILTER CRITERIA LIST (FCL), BUFFER
* LIST, AND DSCB BUFFERS ARE DEFINED WITHIN THIS CSECT.
*
*******************************************************************
*        EQUATES FOR ASSEMBLY CONSTANTS AND REGISTERS
*******************************************************************
BFLE_N   EQU   11  NUMBER OF BUFFER LIST ELEMENTS AND BUFFERS DESIRED
R0       EQU    0
R1       EQU    1
R2       EQU    2
R3       EQU    3
R4       EQU    4
R5       EQU    5
R11      EQU   11
R12      EQU   12
R13      EQU   13
R14      EQU   14
R15      EQU   15
*
*******************************************************************
*    SAVE CALLER'S REGISTERS AND ESTABLISH A NEW REGISTER SAVE AREA
*******************************************************************
         STM   R14,R12,12(R13)       SAVE CALLER'S REGISTERS
         BALR  R12,0                 ESTABLISH THIS PROGRAM'S
         USING *,R12                    BASE REGISTER
         ST    R13,SAVEAREA+4        SAVE ADDRESS OF CALLER'S SAVE AREA
         LA    R15,SAVEAREA          GET ADDRESS OF THE NEW SAVE AREA
         ST    R15,8(,R13)           CHAIN CALLER'S AREA TO OURS
         LR    R13,R15               ESTABLISH THE NEW SAVE AREA
*
```

Figure 123 (Part 1 of 5). Example of the CVAFFILT Macro

```
*******************************************************************
* ESTABLISH ADDRESSABILITY TO THE CVPL. PLACE GIVEN DEB ADDRESS
* IN THE CVPL, INITIALIZE THE FLAG BYTE AND THE RETURN DATA AREA.
*******************************************************************
         LA    R11,CVPL_DEF          ESTABLISH ADDRESSABILITY
         USING CVPL_MAP,R11             TO THE CVPL
         ST    R1,CVDEB              PLACE GIVEN ADDR(DEB) IN CVPL
         MVI   FLAGS,F_RSET          RESET THE LOCAL FLAG BYTE
         XC    RET_AREA,RET_AREA     INIT. DATA RETURN AREA TO ZERO
*
*******************************************************************
* INITIALIZE THE BUFFER LIST HEADER (BFLH) AND ELEMENTS (BFLE)
*******************************************************************
         XC    BFLH_DEF(BFL_SIZE),BFLH_DEF  SET BUFR LIST AREA TO ZERO
         LA    R1,BFLH_DEF           R1 -> BUFFER LIST HEADER
         USING BFL_MAP,R1            ESTABLISH ADDRESSABILITY
         MVI   BFLHNOE,BFLE_N        SET NUMBER OF BUFFER ELEMENTS
         OI    BFLHFL,BFLHDSCB       IDENTIFY AS DSCB BUFR ELEMNT LIST
         LA    R2,BFLH_DEF+BFLHLN    R2 -> FIRST BUFFER LIST ELEMENT
         USING BFLE,R2               ESTABLISH ADDRESSABILITY
         LA    R3,DSCB_DEF           R3 -> FIRST DSCB BUFFER
         LA    R4,BFLE_N             R4 = NUMBER OF ELEMENTS AND BUFRS
BFLE_INT OI    BFLEFL,BFLECHR        REQUEST CCHHR ON RETURN
         MVI   BFLELTH,DSCB_SIZ      SET BUFR LNGTH TO FULL DSCB SIZE
         ST    R3,BFLEBUF            SET ADDR(DSCB BUFFER)
         LA    R2,BFLELN(R2)         R2 -> NEXT BUFFER LIST ELEMENT
         LA    R3,DSCB_SIZ(R3)       R3 -> NEXT DSCB BUFFER
         BCT   R4,BFLE_INT           LOOP THROUGH ALL ELEMENTS
         DROP  R1,R2                 DROP TEMP USING
*
*******************************************************************
* INITIALIZE THE FILTER CRITERIA LIST (FCL) HEADER AND ELEMENT
*******************************************************************
         XC    FCL_DEF(FCL_SIZE),FCL_DEF  SET FCL AREA TO ZERO
         LA    R1,FCL_DEF            R1 -> FCL HEADER
         USING FCL_MAP,R1            ESTABLISH ADDRESSABILITY
         MVC   FCLID,FCL_ID          SET THE EYECATCHER 'FCL '
         MVC   FCLCOUNT,=H'1'        SET NUMBER OF FCL ELEMENTS
         LA    R2,FCLHDEND           R2 -> FIRST (ONLY) FCL ELEMENT
         USING FCLDSN,R2             ESTABLISH ADDRESSABILITY
         MVI   FCLDSNLG,X'02'        SET LENGTH(DSN PATTERN)
         LA    R3,=C'**'             R3 -> C'**'
         ST    R3,FCLDSNA            SET ADDR(DSN PATTERN)
         DROP  R1,R2                 DROP TEMP USING
*
```

Figure 123 (Part 2 of 5). Example of the CVAFFILT Macro

```
*******************************************************************
* ISSUE CVAFFILT ACCESS=READ REQUEST
*******************************************************************
         MVI  RET_FTN,RET_READ       IDENTIFY THE CURRENT FUNCTION
         CVAFFILT ACCESS=READ,FCL=FCL_DEF,BUFLIST=BFLH_DEF,            X
               MF=(E,CVPL_DEF)
*
*******************************************************************
* TEST THE RETURN FROM CVAFFILT ACCESS=READ OR ACCESS=RESUME
*******************************************************************
TEST_RET LTR   R15,R15               IF DSCB RETURN IS COMPLETE
         BZ    COUNTLST                GO COUNT LAST SET OF DSCBS
         CH    R15,=H'4'             IF RETURN CODE OTHER THAN FOUR
         BNE   ERR_RET                 GO RETURN THE ERROR CONDITION
         CLI   CVSTAT,STAT064        IF OTHER THAN RESUME RECOMMENDED
         BNE   ERR_RET                 GO RETURN THE ERROR CONDITION
         B     COUNTCUR              ELSE GO COUNT CURRENT DSCB SET
*
*******************************************************************
* COUNT THE NUMBER OF FORMAT 1, 2, AND 3 DSCBS RETURNED
*******************************************************************
COUNTLST OI    FLAGS,F_LAST          INDICATE LAST SET OF DSCBS
COUNTCUR LM    R1,R3,RET_F1          GET PRIOR DSCB COUNTS
         LA    R4,FCL_DEF            R4 -> FCL HEADER
         USING FCL_MAP,R4            ESTABLISH ADDRESSABILITY
         SLR   R5,R5                 R5 = ZERO FOR FOLLOWING ICM
         ICM   R5,B'0011',FCLDSCBR   R5 = NUMBER OF DSCBS RETURNED
         BZ    TST_RSUM              IF ZERO, GO TEST FOR RESUME
         LA    R4,DSCB_DEF           R4 -> FIRST DSCB BUFFER
         USING DSCB_MAP,R4           ESTABLISH ADDRESSABILITY
COUNTNXT CLI   DS1FMTID,X'F1'        IF FORMAT 1 DSCB
         BE    COUNT_F1                GO INCREMENT ITS COUNTER
         CLI   DS1FMTID,X'F2'        IF FORMAT 2 DSCB
         BE    COUNT_F2                GO INCREMENT ITS COUNTER
         LA    R3,1(R3)              ELSE INCREMENT FORMAT 3 DSCB COUNT
         B     COUNTTST              GO TEST FOR MORE DSCBS
*
COUNT_F2 LA    R2,1(R2)              INCREMENT COUNT OF FORMAT 2 DSCBS
         B     COUNTTST              GO TEST FOR MORE DSCBS
*
COUNT_F1 LA    R1,1(R1)              INCREMENT COUNT OF FORMAT 1 DSCBS
COUNTTST LA    R4,DSCB_SIZ(R4)       R4 -> NEXT DSCB BUFFER
         BCT   R5,COUNTNXT           LOOP THROUGH ALL RETURNED DSCBS
         STM   R1,R3,RET_F1          SAVE UPDATED DSCB COUNTS
         DROP  R4                    FINISHED COUNTING CURR DSCB SET
*
```

Figure 123 (Part 3 of 5). Example of the CVAFFILT Macro

```
*******************************************************************
* CONDITIONALLY ISSUE CVAFFILT ACCESS=RESUME REQUEST
*******************************************************************
TST_RSUM TM    FLAGS,F_LAST          IF LAST SET OF DSCBS COUNTED
         BO    RLSE_REQ                 GO REQUEST A RELEASE
         MVI   RET_FTN,RET__SUM      IDENTIFY THE CURRENT FUNCTION
         CVAFFILT ACCESS=RESUME,MF=(E,CVPL_DEF)  RESUME REQUEST
         B     TEST_RET              GO TEST THE RETURN CODES
*
*******************************************************************
* ISSUE CVAFFILT ACCESS=RLSE REQUEST
*******************************************************************
RLSE_REQ MVI   RET_FTN,RET_RLSE      IDENTIFY THE CURRENT FUNCTION
         CVAFFILT ACCESS=RLSE,FLC=0,BUFLIST=0,FLTAREA=NOKEEP,          X
               MF=(E,CVPL_DEF)
         LTR   R15,R15               IF NO ERROR ON RELEASE REQUEST
         BZ    RETURN                   GO RETURN TO CALLER
         LA    R1,RET_RLSE           ELSE INDICATE ERROR IN RELEASE
*
*******************************************************************
* PLACE ERROR INFORMATION IN THE OUTPUT DATA RETURN AREA
*******************************************************************
ERR_RET  MVC   RET_STAT,CVSTAT       COPY CVSTAT TO RETURN AREA
         ST    R15,RET_RC            COPY CVAFFILT'S RETURN CODE
*
*******************************************************************
* ASSUME R15 = DESIRED RETURN CODE, SET R1 -> DATA RETURN AREA,
* RESTORE CALLER'S REGISTERS 0, 2-14, AND RETURN TO CALLER
*******************************************************************
RETURN   LA    R1,RET_AREA           R1 -> OUTPUT DATA RETURN AREA
         L     R13,SAVEAREA+4        R13 -> CALLER'S REG SAVE AREA
         L     R14,12(R13)           RESTORE CALLER'S REGISTER 14
         L     R0,20(R13)            RESTORE CALLER'S REGISTER 0
         LM    R2,R12,28(R13)        RESTORE CALLER'S REGISTERS 2-12
         BR    R14                   RETURN TO CALLER
FLAGS    DC    X'00'                 LOCAL FLAG BYTE
F_RSET   EQU   X'00'                 FLAG RESET VALUE
F_LAST   EQU   X'80'                 LAST DSCB SET ENCOUNTERED
*
         LTORG
*
SAVEAREA DC    18F'0'                REGISTER SAVE AREA
*
```

Figure 123 (Part 4 of 5). Example of the CVAFFILT Macro

```
*****************************************************************
* MAPPING MACROS
*****************************************************************
CVPL_MAP ICVAFPL CVPLFSA=YES
FCL_MAP  ICVFCL
BFL_MAP  ICVAFBFL
         PUSH PRINT
         PRINT NOGEN
DSCB_MAP DSECT
         IECSDSL1 (1)     USE FORMAT 1 DSCB MAPPING TO GET BUFFER SIZE
DSCB_SIZ EQU   *-IECSDSL1     LENGTH OF FULL DSCB
         POP PRINT
*******************************************************************
*        SPACE ALLOCATION FOR CVPL, FCL, BFL, AND DSCB BUFFERS
*******************************************************************
FLTXMP   CSECT , CONTINUATION OF CSECT
CVPL_DEF CVAFFILT MF=L,BRANCH=NO,FLTAREA=KEEP
FCL_ID   DC    CL4'FCL '
FCL_DEF  DS    (FCLHDLEN+FCLDSNEL)X   FCL HEADER AND ONE FCL ELEMENT
FCL_SIZE EQU   *-FCL_DEF
* DEFINE A CVAF BUFFER LIST WITH N BUFFER LIST ELEMENTS
BFLH_DEF DS    (BFLHLN)X           BUFFER LIST HEADER
BFLE_DEF DS    (BFLE_N*BFLELN)X    N BUFFER LIST ELEMENTS
BFL_SIZE EQU   *-BFLH_DEF
* DEFINE N FULL DSCB BUFFERS
DSCB_DEF DS    (BFLE_N*DSCB_SIZ)X
*******************************************************************
*        OUTPUT DATA RETURN AREA
*******************************************************************
         DS    0F
RET_AREA DS    0XL20    OUTPUT DATA RETURN AREA
RET_F1   DC    F'0'     COUNT OF FORMAT 1 DSCBS RETURNED BY CVAFFILT
RET_F2   DC    F'0'     COUNT OF FORMAT 2 DSCBS RETURNED BY CVAFFILT
RET_F3   DC    F'0'     COUNT OF FORMAT 3 DSCBS RETURNED BY CVAFFILT
*
RET_ERR  DS    0XL8     IF NO ERROR CONDITION WAS ENCOUNTERED THEN
*                          REGISTER 15 = ZERO ON RETURN
*                       OTHERWISE THE FOLLOWING INFORMATION IS PROVIDED
RET_FTN  DC    X'00'    IDENTIFICATION OF CVAFFILT SUBFUNCTION
RET_READ EQU   X'01'    CVAFFILT ACCESS=READ
RET_RSUM EQU   X'02'    CVAFFILT ACCESS=RESUME
RET_RLSE EQU   X'03'    CVAFFILT ACCESS=RLSE
         DC    XL2'00'  UNUSED
RET_STAT DC    X'00'    COPY OF CVSTAT FROM CVAFFILT
RET_RC   DC    F'0'     COPY OF THE RETURN CODE FROM CVAFFILT
         END
```

Figure 123 (Part 5 of 5). Example of the CVAFFILT Macro

CVAFSEQ Macro Overview and Specification

The CVAFSEQ macro can be used to:

- Read an indexed VTOC sequentially in data-set-name (DSN) order
- Read an indexed VTOC or a nonindexed VTOC in physical-sequential order.

See “Accessing DSNs or DSCBs in Sequential Order” on page 47 for additional information.

The format of the CVAFSEQ macro is:

```
>>──┬──────┬──CVAFSEQ──ACCESS=──┬─GT───┬──┬──────────────────────────────────>
    └label─┘                    └─GTEQ─┘  └─,BUFLIST=addr─┘

>──┬─────────┬──┬──────────────┬──┬──────────────────────────────────────────>
  └─,DSN=addr─┘  ├─,UCB=(ucbaddr)─┤  │                  ┌─NO──┐            │
                 └─,DEB=addr──────┘  └─,DSNONLY=──┬──────┬─┘
                                                  └─YES──┘

>──┬─────────┬──┬──────────────────────────────────────────────────────────┬──>
  └─,ARG=addr─┘  │                  ┌─NOKEEP──────────┐                    │
                 └─,IOAREA=──┬──────┬─┘                │                    │
                             ├─KEEP─┬─────────────────┤                    │
                             ├─(KEEP,addr)────────────┤                    │
                             └─(NOKEEP,addr)──────────┘                    │

>──┬──────────────────────────────────────────────────────────────────────┬──>
  │                  ┌─NOKEEP──────────┐                  │                │
  └─,IXRCDS=──┬──────┬─┘                │                  └─,BRANCH=──┬────┬─┘
              ├─KEEP─┬─────────────────┤                              ├─NO─┬─────────┬──────┬──────┬──
```

Note: If the order is physical-sequential, initialize the argument field in the first buffer list entry to zero or to the argument of the DSCB. If the argument is zero (BFLEARG=00), the read begins at the start of the VTOC. You must be authorized (APF or system key) to read multiple DSCBs with a single invocation of the CVAFSEQ macro. See "Initiating Physical-Sequential Access" on page 48 for more information.

UCB or DEB: Specify the VTOC to Be Accessed

UCB= *rs-type or (2-12)* *standard form* **UCB**= *rx-type or (2-12)* *execute form*
specifies the address of the UCB for the VTOC to be accessed. The UCB address can be for a captured UCB, or for an actual UCB above or below the 16MB line. Use the address of a UCB, not a UCB copy. An unauthorized caller must not use this parameter. If your program is in 31-bit mode, this address must be in 31-bit address; the high order byte is part of the address. You should not code the UCB parameter with MF=L.

Note: Code the address of the UCB parameter only as register (2-12). Coding an RX-type address gives you unpredictable results.

DEB=*addr*
supplies the address of a DEB opened to the VTOC you want to access. CVAF does not allow output requests to the VTOC or VTOC index if you specify the DEB subparameter. If you are not authorized, you cannot perform any asynchronous activity (such as EXCP, CLOSE, EOV), against the data set represented by the DEB because CVAF removes the DEB from the DEB table for the duration of the CVAF call. If you are not authorized (neither APF authorized nor in a system key), specify a DEB address, not a UCB, to CVAFSEQ. See "Identifying the Volume" on page 41 for further details.

If you supply a previously obtained I/O area through the IOAREA keyword, neither UCB nor DEB need be supplied. Otherwise, supply either a UCB or DEB. If you supply a UCB address, it is overlaid in the CVPL by the UCB address in the I/O area. If you supply both the UCB and the DEB addresses in the CVPL, the DEB address is used and the UCB address in the CVPL is overlaid by the UCB address in the DEB.

DSNONLY: Specify That Only the Data Set Name Is Read

This keyword is applicable only to accessing an indexed VTOC in DSN order.

DSNONLY=NO
requests that the data set name be obtained from the VTOC index and the DSCB be read into a buffer supplied through the BUFLIST keyword. BUFLIST is required.

DSNONLY=YES
requests that only the data set name be obtained from the VTOC index. If the ARG keyword is coded, the argument of the DSCB is returned.

ARG: Specify Where the Argument of the DSCB Is to Be Returned

This keyword is applicable only to accessing an indexed VTOC in DSN order with DSNONLY=YES coded.

ARG=*addr*
supplies the address of the 5-byte area where the CCHHR of each data set name in the VTOC index is returned when DSNONLY=YES is coded.

IOAREA: Keep or Free the I/O Work Area

IOAREA=KEEP

specifies that the CVAF I/O area associated with the CVAF parameter list is to be kept upon completion of the CVAF request. IOAREA=KEEP can be coded with BRANCH=NO only if the caller is authorized (APF, or system key).

If IOAREA=KEEP is coded, the caller must call CVAF with IOAREA=NOKEEP specified at some future time, whether or not any further VTOC access is required: for example, the recovery routine of the caller of CVAF.

Coding IOAREA=KEEP allows subsequent CVAF requests to be more efficient, because certain initialization functions can be bypassed. Neither DEB nor UCB need be specified when a previously obtained CVAF I/O area is supplied; nor can they be changed.

When IOAREA=KEEP is first issued, CVAF returns the CVAF I/O area in the CVAF parameter list (CVIOAR). Subsequent calls of CVAF can use that same parameter list, and CVAF obtains its I/O area from the CVIOAR.

When processing on the current volume is finished, release all areas that were kept.

IOAREA=(KEEP,*addr*)

supplies the address of a previously obtained I/O area. If a different CVAF parameter list is used, the previously obtained CVAF I/O area can be passed to CVAF by coding its address as the second parameter of the IOAREA keyword.

IOAREA=NOKEEP

causes the work area to be freed upon completion of the CVAF request.

IOAREA=(NOKEEP,*addr*)

causes a previously obtained work area to be freed upon completion of the CVAF request.

IXRCDS: Retain VIERs in Virtual Storage

This keyword applies to an indexed VTOC only.

IXRCDS=KEEP

specifies that the VIERs read into storage during the CVAF function are to be kept in virtual storage. The VIERs are retained even if the index function is unsuccessful. The VIERs are accessed from the CVAF parameter list (CVIRCDS). CVIRCDS is the address of a buffer list containing the VIR buffer addresses and RBAs of the VIERs read.

Index search function dynamically updates the buffer list and, when necessary, obtains additional buffer lists and chains them together.

If IXRCDS=KEEP is specified and no buffer list is supplied to CVAF in the CVPL, CVAF obtains a buffer list and buffers and reads the first high-level VIER. The address of the buffer list is placed in the CVIRCDS field of the CVPL.

The buffer list and VIR buffers are in the caller's protect key. The subpool is 0 if the caller is not authorized; subpool 229 if the caller is authorized.

If IXRCDS=KEEP for an nonindexed VTOC, a request to read a DSCB can be performed, but an error code is returned.

When processing on the current volume is finished, release all areas that were kept.

IXRCDS=(KEEP,*addr*)
The CVIRCDS from one CVAF call can be passed to another CVAF parameter list by specifying the address as the second parameter in the IXRCDS keyword.

IXRCDS=NOKEEP
If IXRCDS=NOKEEP is coded, the VIERs that are accessed (if any) are not retained. Furthermore, the buffer list supplied in the CVIRCDS field in the CVAF parameter list is released, as are all buffers found in the buffer list. If the skip bit is set in any entry in the buffer list, the buffer and buffer list are not freed.

IXRCDS=(NOKEEP,*addr*)
specifies that previously accessed VIERs are not to be retained.

You must free buffer lists and buffers obtained by CVAF. This can be done in one of three ways:

- By coding IXRCDS=NOKEEP on the CVAFSEQ macro that obtained the buffers
- By coding IXRCDS=NOKEEP on a subsequent CVAF macro
- By coding CVAFDIR ACCESS=RLSE and providing the address of the buffer list in the BUFLIST keyword.

Note: You must enqueue the VTOC and reserve the unit to maintain the integrity of the VIERs read.

BRANCH: Specify the Entry to the Macro

BRANCH=(YES,SUP)
requests the branch entry. The caller must be in supervisor state. Protect key checking is bypassed.

If BRANCH=YES is coded, an 18-word save area must be supplied. No lock can be held on entry to CVAF. SRB mode is not allowed.

BRANCH=YES
Equivalent to BRANCH=(YES,SUP), because SUP is the default when YES is coded. Protect key checking is bypassed.

BRANCH=(YES,PGM)
requests the branch entry. The caller must be APF authorized and in problem state. Protect key checking is bypassed.

BRANCH=NO
requests the SVC entry. The caller must be APF authorized if any output operations are requested. Protect key checking is performed. This is the default.

MF: Specify the Form of the Macro

This keyword specifies whether the list, execute, or normal form of the macro is requested.

MF=I
If I is coded, or neither L nor E is coded, the CVAF parameter list is generated, as is code, to call CVAF. This is the normal form of the macro.

MF=L
indicates the list form of the macro. A parameter list is generated, but code to call CVAF is not generated.

MF=(E,*addr*)
indicates the execute form of the macro. The remote CVAF parameter list supplied as *addr* is used in and can be modified by the execute form of the macro.

Return Codes from CVAFSEQ

On return from CVAF, register 1 contains the address of the CVPL (CVAF parameter list), and register 15 contains one of the following return codes:

Return Code	Meaning
0 (X'00')	The request was successful.
4 (X'04')	End of data (CVSTAT is set to decimal 32), or an error was encountered. The CVSTAT field in the CVPL contains an indication of the cause of the error. Error descriptions are in "VTOC Index Error Message and Associated Codes" on page 360.
8 (X'08')	Invalid VTOC index structure. CVSTAT contains an indication of the cause of the error. Error descriptions are in "VTOC Index Error Message and Associated Codes" on page 360.
12 (X'0C')	The CVPL (CVAF parameter list) is not in your protect key, or is not valid (the ID is not valid, or the length field is incorrect, or the CVFCTN field is not valid). The CVPL has not been modified.
16 (X'10')	An I/O error was encountered.

Example of Using the CVAFSEQ Macro with an Indexed VTOC

This example uses the CVAFSEQ macro to count the number of ISAM data sets whose data set names are within the range defined by two supplied data set names. The addresses of the two data set names are supplied to the program in registers 6 and 7, labeled RDSN1 and RDSN2, respectively. The address of a DEB open to the VTOC is supplied in register 4, labeled RDEB.

The CVAF parameter list is expanded by a list form of the CVAFSEQ macro. ACCESS=GTEQ is specified on the list form of macro and is, therefore, not coded in the first execution of the CVPL. Subsequent executions of the CVPL (at label RELOOP) specify ACCESS=GT.

End of data is tested by comparing the CVSTAT field to the value of STAT032, which is an equate in the ICVAFPL mapping macro.

The count of ISAM DSCBs matching the data set name criterion is returned in register 15, unless an error is encountered, in which case a negative 1 is returned in register 15.

```
SEQXMP1  CSECT
         STM   14,12,12(13)
         BALR  12,0
         USING *,12
         ST    13,SAVEAREA+4
         LA    RWORK,SAVEAREA
         ST    RWORK,8(,13)
         LR    13,RWORK
*************************************************************
*
*        REGISTERS
*
*************************************************************
REG1     EQU   1                  REGISTER 1
RWORK    EQU   3                  WORK REGISTER
RDEB     EQU   4                  DEB ADDRESS
RDSN1    EQU   6                  ADDRESS OF DATA SET NAME 1
RDSN2    EQU   7                  ADDRESS OF DATA SET NAME 2
REG15    EQU   15                 RETURN CODE REGISTER 15
*************************************************************
*
*        COUNT THE NUMBER OF ISAM DATA SETS WHOSE DATA SET NAMES ARE
*           BETWEEN DSN1 AND DSN2 INCLUSIVELY.
*           RDSN1 CONTAINS ADDRESS OF DSN1.
*           RDSN2 CONTAINS ADDRESS OF DSN2.
*        ADDRESS OF DEB OPEN TO VTOC SUPPLIED IN RDEB.
*
*************************************************************
         XC   BUFLIST(BFLHLN+BFLELN),BUFLIST ZERO BUFFER LIST
         OI   BFLHFL,BFLHDSCB      DSCBS TO BE READ WITH BUFFER LIST
         MVI  BFLHNOE,1            ONE BUFFER LIST ENTRY
         LA   RWORK,DS1FMTID       ADDRESS OF DSCB BUFFER
         ST   RWORK,BFLEBUF        PLACE IN BUFFER LIST
         MVI  BFLELTH,DSCBLTH      DATA PORTION OF DSCB READ - DSN     *
                                   SUPPLIED IN CVPL
         MVC  DS1DSNAM,0(RDSN1)    MOVE IN STARTING DATA SET NAME TO   *
                                   WORKAREA
         XR   RWORK,RWORK          ZERO COUNT
         CVAFSEQ DEB=(RDEB),       FIND FIRST DATA SET WHOSE DATA SET  *
               BUFLIST=BUFLIST,    NAME IS GREATER THAN OR EQUAL TO    *
               MF=(E,CVPL)         THAT OF DSN1
LOOP     EQU  *                    LOOP UNTIL END OF DATA OR DATA SET  *
                                   NAME GREATER THAN DSN2
         USING CVPL,REG1           ADDRESSABILITY TO CVPL
         LTR  REG15,REG15          ANY ERROR
         BZ   TESTDSN              BRANCH IF NOT-CHECK DSN LIMIT
```

Figure 124 (Part 1 of 3). Example of the CVAFSEQ Macro with an Indexed VTOC

```
*********************************************************
*
*        DETERMINE WHAT ERROR IS
*
*********************************************************
         C    REG15,ERROR4         IS RETURN CODE 4
         BNE  OTHERERR             BRANCH IF NOT 4
         CLI  CVSTAT,STAT032       IS IT END OF DATA?
         BNE  OTHERERR             BRANCH IF NOT
         DROP REG1                 ADDRESSABILITY TO CVPL NOT NEEDED
*********************************************************
*
*        END OF DATA
*
*********************************************************
         B    RELEASE              RELEASE CVAF RESOURCES AND RETURN
TESTDSN  EQU  *                   IS DATA SET NAME GREATER THAN DSN2
         CLI  DS1FMTID,C'1'        IS THIS A FORMAT 1 DSCB?
         BNE  CKLAST               BRANCH IF NO. CAN NOT BE ISAM.
         CLC  DS1DSNAM,0(RDSN2)    HAS LIMIT BEEN REACHED?
         BNH  TESTIS               BRANCH IF NO-TEST FOR ISAM
         B    RELEASE              RELEASE CVAF RESOURCES AND RETURN
TESTIS   EQU  *                   ONLY COUNT ISAM
         TM   DS1DSORG,DS1DSGIS    IS DATA SET ISAM
         BZ   CKLAST               BRANCH IF NO-DO NOT COUNT IT
         LA   RWORK,1(,RWORK)      INCREMENT COUNT BY ONE
CKLAST   EQU  *                   CHECK IF LAST DATA SET NAME (DSN2)
         CLC  DS1DSNAM,0(RDSN2)    HAS LIMIT BEEN REACHED?
         BNH  RELOOP               BRANCH IF NO-READ NEXT ONE
         B    RELEASE              RELEASE CVAF RESOURCES AND RETURN
RELOOP   EQU  *                   READ NEXT DSCB
         CVAFSEQ ACCESS=GT,MF=(E,CVPL) GET DSCB WITH DATA SET NAME     *
                                   GREATER THAN THE ONE LAST READ
         B    LOOP                 CHECK RESULTS OF CVAFSEQ
OTHERERR EQU  *                   UNEXPECTED ERROR
```

Figure 124 (Part 2 of 3). Example of the CVAFSEQ Macro with an Indexed VTOC

```
*********************************************************
*
*        UNEXPECTED ERROR PROCESSING
*
*********************************************************
         LA   RWORK,1(0,0)         ONE IN RWORK
         LNR  RWORK,RWORK          SET NEGATIVE COUNT INDICATING ERROR
RELEASE  CVAFDIR ACCESS=RLSE,      RELEASE CVAF BUFFERS/IOAREA         *
               BUFLIST=0,          DO NOT RELEASE USER BUFFER LIST     *
               IXRCDS=NOKEEP,      RELEASE CVAF VIER BUFFERS           *
               MF=(E,CVPL)         RELEASE CVAF I/O AREA
         LR   REG15,RWORK          CURRENT COUNT IS RETURN CODE
         L    13,SAVEAREA+4
         RETURN (14,12),RC=(15)    RETURN CURRENT COUNT
ERROR4   DC   F'4'                 ERROR RETURN CODE 4
BUFLIST  ICVAFBFL DSECT=NO         BUFFER LIST
         IECSDSL1 (1)              FORMAT 1 DSCB DATASET NAME AND      *
                                   BUFFER
DSCBLTH  EQU  *-IECSDSL1-L'DS1DSNAM LENGTH OF DATA PORTION OF DSCB
SAVEAREA DS   18F                  SAVE AREA
CVPL     CVAFSEQ ACCESS=GTEQ,      READ DSCB WITH DSN >= SUPPLIED DSN  *
               IXRCDS=KEEP,        KEEP VIERS IN STORAGE DURING CALLS  *
               DSN=DS1DSNAM,       SUPPLIED DATA SET NAME              *
               BUFLIST=BUFLIST,                                        *
               MF=L
         ORG  CVPL                 EXPAND MAP OVER LIST
CVPLMAP  ICVAFPL DSECT=NO          CVPL MAP

         END
```

Figure 124 (Part 3 of 3). Example of the CVAFSEQ Macro with an Indexed VTOC

Example of Using the CVAFSEQ Macro with a Nonindexed VTOC

This example reads as many as five DSCBs in physical-sequential order. The address of the UCB is supplied to the program in register 5 (labeled RUCB). The address of a parameter list is supplied in register 4 (labeled RLIST). The first word of the parameter list contains the address of a 5-byte field. On entry, this field is set to zero if no previous DSCBs have been read; otherwise, the field is set to the CCHHR of the last DSCB read. This 5-byte field is supplied by the caller of this program and is not modified by this program.

The remainder of the parameter list consists of one or more 2-word entries, to a maximum of five 2-word entries. The first word of each entry contains the address of a 140-byte DSCB buffer. The second word contains the address of a 5-byte field that is to contain the CCHHR of the DSCB.

A buffer list with five buffer list entries is contained in the program. The ICVAFBFL macro generates the buffer list header and one buffer list entry. The remaining buffer list entries are generated following the ICVAFBFL macro.

The CVAFSEQ macro is used once in the program to read as many DSCBs as there are 2-word entries in the parameter list. The buffer list header field BFLHNOE is initialized with the number of buffer list entries that CVAFSEQ is to process. The number matches the number of 2-word entries in the parameter list supplied to this program.

After the CVAFSEQ call, the CCHHR for each DSCB read is moved from the buffer list entry field BFLEARG to the field whose address is supplied by the caller of the program. If the BFLEARG field is zero, the previous DSCB read was the last in the VTOC.

The BFLEARG in the first buffer list entry is initialized with the CCHHR supplied by the caller; its address is the third word in the parameter list. This CCHHR serves as the starting place for the CVAFSEQ call. DSCBs with a CCHHR greater than the supplied CCHHR are read.

This program must be APF authorized.

```
SEQXMP2  CSECT
         STM   14,12,12(13)
         BALR  12,0
         USING *,12
         ST    13,SAVEAREA+4
         LA    RWORK,SAVEAREA
         ST    RWORK,8(,13)
         LR    13,RWORK
************************************************************
*
*        REGISTERS
*
************************************************************
REG1     EQU   1                  REGISTER 1
RWORK    EQU   3                  WORK REGISTER
RLIST    EQU   4                  ADDRESS OF PARM LIST
RUCB     EQU   5                  UCB ADDRESS
RCURRENT EQU   6                  CURRENT ENTRY IN PARM LIST
RBLE     EQU   7                  CURRENT BUFFER LIST ENTRY
RCOUNT   EQU   8                  COUNT OF ENTRIES IN BUFFER LIST
REG15    EQU   15                 RETURN CODE REGISTER 15
************************************************************
*
*        READ UP TO 5 DSCBS.
*        RUCB CONTAINS ADDRESS OF UCB.
*        RLIST CONTAINS ADDRESS OF PARAMETER LIST.
*          WORD 0 = ADDRESS OF CCHHR OF LAST DSCB READ. THIS DSCB IS
*                   NOT TO BE READ
*          WORD 1 = ADDRESS OF DSCB BUFFER.
*          WORD 2 = ADDRESS OF CCHHR OF DSCB READ.
*            WORD1 AND WORD2 REPEATED UP TO 4 TIMES.
*            HIGH ORDER BIT OF WORD 2 SET TO ONE FOR LAST ENTRY.
*
************************************************************
```

Figure 125 (Part 1 of 4). Example of the CVAFSEQ Macro with a Nonindexed VTOC

```
         USING LIST,RLIST          ADDRESSABILITY TO PARM LIST
         XC   BFLHDR(BFLHLN+5*BFLELN),BFLHDR ZERO BUFFER LIST WITH     *
                                   5 BUFFER LIST ENTRIES
         OI   BFLHFL,BFLHDSCB      DSCBS TO BE READ WITH BUFFER LIST
         LA   RCURRENT,LISTPRMS    FIRST DOUBLEWORD ENTRY IN PARM LIST
         USING LISTPRMS,RCURRENT   USING ON DOUBLEWORDS
         LA   RBLE,BFLE            FIRST BUFFER LIST ENTRY
         USING BFLE,RBLE
         L    RWORK,LISTSTRT       ADDRESS OF STARTING CCHHR
         MVC  BFLEARG,0(RWORK)     MOVE STARTING CCHHR INTO FIRST      *
                                   BUFFER LIST ENTRY
         XR   RCOUNT,RCOUNT        ZERO COUNT
BUFLOOP  EQU  *                    PUT BUFFER ADDRESSES IN BUFFER LIST *
                                   ENTRIES
         LA   RCOUNT,1(,RCOUNT)    INCREMENT COUNT
         L    RWORK,LISTBUF        ADDRESS OF DSCB BUFFER
         ST   RWORK,BFLEBUF-BFLE(,RBLE) PLACE IN BUFFER LIST
         MVI  BFLELTH-BFLE(RBLE),DSCBLTH FULL DSCB READ
         TM   LISTLAST,LASTBIT     IS IT LAST ENTRY IN LIST
         LA   RCURRENT,LISTNEXT    INCREMENT TO NEXT ENTRY IN LIST
         LA   RBLE,BFLELN(,RBLE)   INCREMENT TO NEXT BUFFER LIST ENTRY
         BZ   BUFLOOP              LOOP TO PUT NEXT BUFFER IN BFLE
         STC  RCOUNT,BFLHNOE       SET NUMBER OF ENTRIES IN BUFFER     *
                                   LIST HEADER
         DROP RCURRENT,RBLE
************************************************************
*
*        READ UP TO 5 DSCBS WHOSE CCHHR IS GREATER THAN THE CCHHR IN
*        THE FIRST BUFFER LIST ENTRY
*
************************************************************
         CVAFSEQ UCB=(RUCB),       CALL CVAF                           *
               BRANCH=YES,         BRANCH ENTER                        *
               MF=(E,CVPL)
         USING CVPL,REG1           ADDRESSABILITY TO CVPL
         LTR  REG15,REG15          ANY ERROR
         BZ   MOVECHR              BRANCH IF MOVE IN CCHHRS
************************************************************
*
*        DETERMINE WHAT ERROR IS
*
************************************************************
         C    REG15,ERROR4         IS RETURN CODE 4
         BNE  OTHERERR             BRANCH IF NOT 4
         CLI  CVSTAT,STAT032       IS IT END OF DATA?
         BNE  OTHERERR             BRANCH IF NOT
         DROP REG1                 ADDRESSABILITY TO CVPL NOT NEEDED
```

Figure 125 (Part 2 of 4). Example of the CVAFSEQ Macro with a Nonindexed VTOC

```
*********************************************************
*
*        DETERMINE IF ANY DSCBS HAVE BEEN READ. BFLEARG IS NON-ZERO
*        IN EACH BUFFER LIST ENTRY FOR WHICH A DSCB HAS BEEN READ
*
*********************************************************
MOVECHR  EQU  *                    IS DATA SET NAME GREATER THAN DSN2
         LA   RCURRENT,LISTPRMS    FIRST ENTRY IN PARM LIST
         USING LISTPRMS,RCURRENT
         LA   RBLE,BFLE            FIRST BUFFER LIST ENTRY
         USING BFLE,RBLE
CHRLOOP  EQU  *                    MOVE CCHHR ARGUMENT TO CALLER AREA
         L    RWORK,LISTCHR        ADDRESS OF CCHHR OF CALLER
         XC   0(L'BFLEARG,RWORK),0(RWORK) ZERO CALLER CCHHR AREA
         NC   BFLEARG,BFLEARG      IS CCHHR ZERO
         BZ   EXIT                 BRANCH IF YES-NO MORE DSCBS
         MVC  0(L'BFLEARG,RWORK),BFLEARG MOVE CCHHR TO CALLER AREA
         TM   LISTLAST,LASTBIT     LAST ENTRY IN PARM LIST?
         BO   EXIT                 BRANCH IF YES
         LA   RCURRENT,LISTNEXT    NEXT ENTRY IN LIST
         LA   RBLE,BFLELN(,RBLE)   NEXT BUFFER LIST ENTRY
         B    CHRLOOP              TEST NEXT BFLE
EXIT     EQU  *                    RETURN TO CALLER
         L    13,SAVEAREA+4
         RETURN (14,12)
OTHERERR EQU  *                    ERROR PROCESSING
*
*
*
         B    EXIT                 RETURN
ERROR4   DC   F'4'                 RETURN CODE 4
         ICVAFBFL DSECT=NO         BUFFER LIST WITH ONE BUFFER LIST    *
                                   ENTRY
         DS   CL(4*BFLELN)         FOUR BUFFER LIST ENTRIES
SAVEAREA DS   18F                  SAVE AREA
DSCB     DSECT
         IECSDSL1 (1)              FORMAT 1 DSCB DATASET NAME AND      *
                                   DATA
DSCBLTH  EQU  *-IECSDSL1           LENGTH OF DSCB
LIST     DSECT                     PARAMETER LIST
LISTSTRT DS    F                   ADDRESS OF CCHHR TO START SEARCH
LISTPRMS EQU   *
LISTBUF  DS    F                   BUFFER ADDRESS
LISTCHR  DS    0F                  ADDRESS OF CCHHR FIELD
LISTLAST DS    X                   BYTE
LASTBIT  EQU   X'80'               LAST DOUBLE WORD
         DS    AL3                 3 BYTE ADDRESS OF CCHHR
LISTNEXT EQU   *                   NEXT DOUBLEWORD
SEQXMP2  CSECT
```

Figure 125 (Part 3 of 4). Example of the CVAFSEQ Macro with a Nonindexed VTOC

```
*********************************************************
*
*        READ DSCBS WITH CCHHR GREATER THAN THE CCHHR IN THE FIRST
*        BUFFER LIST ENTRY.
*
*********************************************************
CVPL     CVAFSEQ ACCESS=GT,                                        *
               BUFLIST=BFLHDR,     ADDRESS OF BUFFER LIST          *
               MF=L
         ORG  CVPL                 EXPAND MAP OVER LIST
CVPLMAP  ICVAFPL DSECT=NO          CVPL MAP
         END
```

Figure 125 (Part 4 of 4). Example of the CVAFSEQ Macro with a Nonindexed VTOC

CVAFTST Macro Overview and Specification

The CVAFTST macro determines whether the system supports an indexed VTOC, and, if it does, whether the VTOC on the unit whose UCB is supplied is indexed or nonindexed.

When you issue CVAFTST, register 13 must contain the address of a standard 18 word save area.

You will get a return code of 12 if CVAFTST cannot determine whether an indexed or nonindexed VTOC is on the unit's volume. You should not receive a return code of 12 from CVAFTST if you have opened a data set (including the VTOC) on the volume.

You need no authorization to issue the CVAFTST macro.

See “CVAFDSM Macro Overview and Specification” on page 330 for an example of using the CVAFTST macro with the CVAFDSM macro.

The format of the CVAFTST macro is:

```
►►──┬──────────┬──CVAFTST──UCB=(reg)──────────────────────────────────►◄
    └─label────┘
```

UCB: Specify the VTOC to Be Tested

UCB=(*reg*)

supplies the address of the UCB for the volume whose VTOC is to be tested. The UCB address can be for a captured UCB, or for an actual UCB above or below the 16MB line. If your program is in 31-bit mode, this address must be in 31-bit address; the high order byte is part of the address.

Note: Code the address of the UCB parameter as register (2-12). Coding an RX-type address gives unpredictable results.

The CVAFTST macro accepts the address of a UCB or UCB copy. Unauthorized programs can get a copy of the UCB by using the UCBSCAN macro and specifying the COPY, UCBAREA, CMXTAREA, and DCEAREA keywords. The UCB copy and common extension copy must be below the 16MB line and on a

word boundary. Data accessed with DCEAREA can be above the 16MB line. Refer to *OS/390 HCD Planning* for details.

Return Codes from CVAFTST

On return from CVAF, register 15 contains one of the following return codes:

Return Code	Meaning
0 (X'00')	The system does not support an indexed VTOC. The volume should be considered to have a nonindexed VTOC. The UCB was not inspected to determine its validity or status.
4 (X'04')	The system supports an indexed VTOC, but the volume has a nonindexed VTOC.
8 (X'08')	The system supports an indexed VTOC and the volume has an indexed VTOC.
12 (X'0C')	The system supports an indexed VTOC, but the volume is not mounted or the VIB is not initialized for it; thus, the status (indexed or nonindexed) of the VTOC cannot be determined.
16 (X'10')	The system supports an indexed VTOC, but the unit is not a DASD or has a VIO UCB, or the UCB address is not valid. The address of a UCB copy is not valid without a CMXTAREA and a DCEAREA.

VTOC Index Error Message and Associated Codes

Error Message

When CVAF finds an error in a VTOC index, it issues the following message:

IEC606I VTOC INDEX DISABLED ON
dev,volser,code,[*rba*[*,secno,offset*]]

In addition, CVAF puts a return code in the CVSTAT field of the CVPL.

Explanation

The Common VTOC Access Facility (CVAF) detected a VTOC index error on the device *dev* with volume serial number *volser*. A number that represents the kind of VTOC index error is provided in the *code* field. The RBA of the VIR in the VTOC index that contains the structure error indicated by *code* is provided in the *rba* field. If the VIR is a VIER, the section number in the VIER containing the VTOC index entry is supplied in the *secno* field, and the offset into the section of that VTOC index entry is supplied in the *offset* field.

System Action

The VTOC index is disabled. The VTOC will be converted to nonindexed format when DADSM next allocates space on the volume. A system dump is written to the SYS1.DUMP data set, and an entry is made in the SYS1.LOGREC data set. The message IEC604I (which indicates that the VTOC convert routines have been used) will be issued later.

Programmer Response

Examine the system dump and a print of the VTOC index, and use the information in message IEC606I to determine the cause of the VTOC index structure error.

Routing and Descriptor Codes

The routing codes are 4 (direct access pool) and 10 (system/error maintenance), and the descriptor code is 4 (system status).

Codes Put in the CVSTAT Field

If you are diagnosing an error and require a description of the CVSTAT field codes, see the publication *DFSMS/MVS DFSMSdfp Diagnosis Reference*.

Appendix B. ANTRQST Macro — Call to the System Data Mover API

This section describes the application program call to the OS/390 system data mover (SDM) application programming interface (API).

The environment, programming requirements, input and output register information are summarized, followed by the macro syntax and individual parameter descriptions. Descriptions of the associated return and reason codes are followed by an example of the ANTRQST macro in assembler. Additional details on the XRC and PPRC command parameters can be found in *DFSMS/MVS Remote Copy Administrator's Guide and Reference*.

Latest Updates to This Macro

Every effort has been made to provide you with the most accurate and current information for this ANTRQST macro at the time of availability. **To stay informed about the latest functional changes and additions, refer to the extensive documentation at the top of the latest copy of this macro**. Each copy of the ANTRQST macro contains the corresponding documentation for that release level.

Figure 126 shows how various OS/390 or MVS functions are addressed by the macro.

Figure 126. How the ANTRQST Macro Relates to Major OS/390 Functions

REQUEST=	Related Function
LEVEL	PPRC, Snapshot, and XRC
PDELPAIR **P**DELPATH **P**ESTPAIR **P**ESTPATH **P**FREEZE **P**QUERY **P**RECOVER **P**RUN **P**SUSPEND	PPRC (peer-to-peer remote copy)
SDVCINFO **S**QRYDVCS **S**QRYSSYS **S**RELEASE **S**SNAP	Snapshot
XCONTIME	XRC (extended remote copy)

Environment

The requirements for the caller are:

© Copyright IBM Corp. 1979, 1999

Minimum authorization:	The user identifier found in the ACEEUSRI field of the ACEE is used for all access authorization checking. This user identifier must have read access to the FACILITY class profile. For XRC requests the profile is found in STGADMIN.ANT.XRC.COMMANDS. For PPRC requests the profile is found in STGADMIN.ANT.PPRC.COMMANDS. For Snapshot requests the profile is found in STGADMIN.ANT.SNAPSHOT.COMMANDS.
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	31-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held.
Control parameters:	Control parameters must be in the primary address space.

Programming Requirements

For PPRC requests the ANTAS000 address space must be active at the time of macro invocation.

For XRC requests the ANTAS000 and ANTAS001 address spaces must be active at the time of macro invocation.

For Snapshot requests, the ANTMAIN address space must be active at the time of macro invocation.

Restrictions

None.

Input Register Information

Before macro invocation register 13 should point to a standard 72-byte save area.

Output Register Information

When control returns to the caller, the GPRs contain:

Register	Contents
0	Reason code
1	Used as a work register by the system
2-13	Unchanged
14	Used as a work register by the system
15	Return code

Performance Implications

None.

Syntax

The ANTRQST macro is written as follows:

name	*name:* symbol. Begin *name* in column 1.
(blank)	One or more blanks must precede ANTRQST.
ANTRQST	
(blank)	One or more blanks must follow ANTRQST.
REQUEST=LEVEL	
,RETINFO=*retinfo*	*retinfo:* RS-type address or address in register (2) - (12).
REQUEST=PDELPAIR	
,DEVN=*devn*	*devn:* RS-type address or address in register (2) - (12).
,PCCA=*pcca*	*pcca:* RS-type address or address in register (2) - (12).
,PSERIAL=*pserial*	*pserial:* RS-type address or address in register (2) - (12).
,PSSID=*pssid*	*pssid:* RS-type address or address in register (2) - (12).
,SCCA=*scca*	*scca:* RS-type address or address in register (2) - (12).
,SSERIAL=*sserial*	*sserial:* RS-type address or address in register (2) - (12).
,SSSID=*sssid*	*sssid:* RS-type address or address in register (2) - (12).
,RETINFO=*retinfo*	*retinfo:* RS-type address or address in register (2) - (12).
,ASYNCH=*asynch*	*.asynch:* RS-type address or address in register (2) - (12). **Default:** ASYNCH=NO.
,ECB=*ecb*	*.ecb:* RS-type address or address in register (2) - (12). **Default:** ECB=NO_ECB.
,WAITTIME=*waittime*	*.waittime:* RS-type address or address in register (2) - (12). **Default:** WAITTIME=0.
REQUEST=PDELPATH	
,DEVN=*devn*	*devn:* RS-type address or address in register (2) - (12).
,PSERIAL=*pserial*	*pserial:* RS-type address or address in register (2) - (12).
,PSSID=*pssid*	*pssid:* RS-type address or address in register (2) - (12).
,SSERIAL=*sserial*	*sserial:* RS-type address or address in register (2) - (12).
,SSSID=*sssid*	*sssid:* RS-type address or address in register (2) - (12).
,RETINFO=*retinfo*	*retinfo:* RS-type address or address in register (2) - (12).
,ASYNCH=*asynch*	*.asynch:* RS-type address or address in register (2) - (12). **Default:** ASYNCH=NO.
,ECB=*ecb*	*.ecb:* RS-type address or address in register (2) - (12). **Default:** ECB=NO_ECB.
,WAITTIME=*waittime*	*.waittime:* RS-type address or address in register (2) - (12). **Default:** WAITTIME=0.
REQUEST=PESTPAIR	
,DEVN=*devn*	*devn:* RS-type address or address in register (2) - (12).
,PCCA=*pcca*	*pcca:* RS-type address or address in register (2) - (12).
,PSERIAL=*pserial*	*pserial:* RS-type address or address in register (2) - (12).
,PSSID=*pssid*	*pssid:* RS-type address or address in register (2) - (12).
,SCCA=*scca*	*scca:* RS-type address or address in register (2) - (12).
,SSERIAL=*sserial*	*sserial:* RS-type address or address in register (2) - (12).
,SSSID=*sssid*	*sssid:* RS-type address or address in register (2) - (12).
,RETINFO=*retinfo*	*retinfo:* RS-type address or address in register (2) - (12).

,ASYNCH=*asynch*	*.asynch:* RS-type address or address in register (2) - (12). **Default:** ASYNCH=NO.
,ECB=*ecb*	*.ecb:* RS-type address or address in register (2) - (12). **Default:** ECB=NO_ECB.
,CRIT=*crit*	*.crit:* RS-type address or address in register (2) - (12). **Default:** CRIT=NO.
,MODE=*mode*	*.mode:* RS-type address or address in register (2) - (12). **Default:** MODE=COPY.
,MSGREQ=*msgreq*	*.msgreq:* RS-type address or address in register (2) - (12). **Default:** MSGREQ=NO.
,PACE=*pace*	*.pace:* RS-type address or address in register (2) - (12). **Default:** PACE=DEFAULT.
,WAITTIME=*waittime*	*.waittime:* RS-type address or address in register (2) - (12). **Default:** WAITTIME=0.
REQUEST=PESTPATH	
,DEVN=*devn*	*devn:* RS-type address or address in register (2) - (12).
,LINK1=*link1*	*link1:* RS-type address or address in register (2) - (12).
,LINK2=*link2*	*.link2:* RS-type address or address in register (2) - (12). **Default:** LINK2=0.
,LINK3=*link3*	*.link3:* RS-type address or address in register (2) - (12). **Default:** LINK3=0.
,LINK4=*link4*	*.link4:* RS-type address or address in register (2) - (12). **Default:** LINK4=0.
,PSERIAL=*pserial*	*pserial:* RS-type address or address in register (2) - (12).
,PSSID=*pssid*	*pssid:* RS-type address or address in register (2) - (12).
,SSERIAL=*sserial*	*sserial:* RS-type address or address in register (2) - (12).
,SSSID=*sssid*	*sssid:* RS-type address or address in register (2) - (12).
,RETINFO=*retinfo*	*retinfo:* RS-type address or address in register (2) - (12).
,ASYNCH=*asynch*	*.asynch:* RS-type address or address in register (2) - (12). **Default:** ASYNCH=NO.
,ECB=*ecb*	*.ecb:* RS-type address or address in register (2) - (12). **Default:** ECB=NO_ECB.
,WAITTIME=*waittime*	*.waittime:* RS-type address or address in register (2) - (12). **Default:** WAITTIME=0.
REQUEST=PFREEZE	
,DEVN=*devn*	*devn:* RS-type address or address in register (2) - (12).
,PSERIAL=*pserial*	*pserial:* RS-type address or address in register (2) - (12).
,PSSID=*pssid*	*pssid:* RS-type address or address in register (2) - (12).
,SSERIAL=*sserial*	*sserial:* RS-type address or address in register (2) - (12).
,SSSID=*sssid*	*sssid:* RS-type address or address in register (2) - (12).
,RETINFO=*retinfo*	*retinfo:* RS-type address or address in register (2) - (12).
,ASYNCH=*asynch*	*.asynch:* RS-type address or address in register (2) - (12). **Default:** ASYNCH=NO.

,ECB=*ecb*	*.ecb:* RS-type address or address in register (2) - (12). **Default:** ECB=NO_ECB.
,WAITTIME=*waittime*	*.waittime:* RS-type address or address in register (2) - (12). **Default:** WAITTIME=0.
REQUEST=PQUERY	
,DEVN=*devn*	*devn:* RS-type address or address in register (2) - (12).
,QRYSIZE=*qrysize*	*qrysize:* RS-type address or address in register (2) - (12).
,QRYINFO=*qryinfo*	*qryinfo:* RS-type address or address in register (2) - (12).
,RETINFO=*retinfo*	*retinfo:* RS-type address or address in register (2) - (12).
,ALET=*alet*	*.alet:* RS-type address or address in register (2) - (12). **Default:** ALET=0.
,ASYNCH=*asynch*	*.asynch:* RS-type address or address in register (2) - (12). **Default:** ASYNCH=NO.
,ECB=*ecb*	*.ecb:* RS-type address or address in register (2) - (12). **Default:** ECB=NO_ECB.
,BITMAP=*bitmap*	*.bitmap:* RS-type address or address in register (2) - (12). **Default:** BITMAP=YES.
,PATHS=*paths*	*.paths:* RS-type address or address in register (2) - (12). **Default:** PATHS=NO.
,WAITTIME=*waittime*	*.waittime:* RS-type address or address in register (2) - (12). **Default:** WAITTIME=0.
REQUEST=PRECOVER	
,DEVN=*devn*	*devn:* RS-type address or address in register (2) - (12).
,PCCA=*pcca*	*pcca:* RS-type address or address in register (2) - (12).
,PSERIAL=*pserial*	*pserial:* RS-type address or address in register (2) - (12).
,PSSID=*pssid*	*pssid:* RS-type address or address in register (2) - (12).
,SCCA=*scca*	*scca:* RS-type address or address in register (2) - (12).
,SSERIAL=*sserial*	*sserial:* RS-type address or address in register (2) - (12).
,SSSID=*sssid*	*sssid:* RS-type address or address in register (2) - (12).
,RETINFO=*retinfo*	*retinfo:* RS-type address or address in register (2) - (12).
,ASYNCH=*asynch*	*.asynch:* RS-type address or address in register (2) - (12). **Default:** ASYNCH=NO.
,ECB=*ecb*	*.ecb:* RS-type address or address in register (2) - (12). **Default:** ECB=NO_ECB.
,OLDVOLSER=*oldvolser*	*.oldvolser:* RS-type address or address in register (2) - (12). **Default:** OLDVOLSER=NO_VOLSER.
,NEWVOLSER=*newvolser*	*.newvolser:* RS-type address or address in register (2) - (12). **Default:** NEWVOLSER=NO_VOLSER.
,WAITTIME=*waittime*	*.waittime:* RS-type address or address in register (2) - (12). **Default:** WAITTIME=0.
REQUEST=PRUN	
,DEVN=*devn*	*devn:* RS-type address or address in register (2) - (12).
,PSERIAL=*pserial*	*pserial:* RS-type address or address in register (2) - (12).
,PSSID=*pssid*	*pssid:* RS-type address or address in register (2) - (12).

,SSERIAL=*sserial*	*sserial:* RS-type address or address in register (2) - (12).
,SSSID=*sssid*	*sssid:* RS-type address or address in register (2) - (12).
,RETINFO=*retinfo*	*retinfo:* RS-type address or address in register (2) - (12).
,ASYNCH=*asynch*	*.asynch:* RS-type address or address in register (2) - (12). **Default:** ASYNCH=NO.
,ECB=*ecb*	*.ecb:* RS-type address or address in register (2) - (12). **Default:** ECB=NO_ECB.
,WAITTIME=*waittime*	*.waittime:* RS-type address or address in register (2) - (12). **Default:** WAITTIME=0.
REQUEST=PSUSPEND	
,DEVN=*devn*	*devn:* RS-type address or address in register (2) - (12).
,PRIMARY=*primary*	*.primary:* RS-type address or address in register (2) - (12). **Default:** PRIMARY=NO.
,QUIESCE=*quiesce*	*.quiesce:* RS-type address or address in register (2) - (12). **Default:** QUIESCE=NO.
,PCCA=*pcca*	*pcca:* RS-type address or address in register (2) - (12).
,PSERIAL=*pserial*	*pserial:* RS-type address or address in register (2) - (12).
,PSSID=*pssid*	*pssid:* RS-type address or address in register (2) - (12).
,SCCA=*scca*	*scca:* RS-type address or address in register (2) - (12).
,SSERIAL=*sserial*	*sserial:* RS-type address or address in register (2) - (12).
,SSSID=*sssid*	*sssid:* RS-type address or address in register (2) - (12).
,RETINFO=*retinfo*	*retinfo:* RS-type address or address in register (2) - (12).
,ASYNCH=*asynch*	*.asynch:* RS-type address or address in register (2) - (12). **Default:** ASYNCH=NO.
,ECB=*ecb*	*.ecb:* RS-type address or address in register (2) - (12). **Default:** ECB=NO_ECB.
,WAITTIME=*waittime*	*.waittime:* RS-type address or address in register (2) - (12). **Default:** WAITTIME=0.
REQUEST=SDVCINFO	
,DEVN=*devn*	*devn:* RS-type address or address in register (2) - (12).
,VOLSER=*volser*	*volser:* RS-type address or address in register (2) - (12).
,UCBPTR=*ucbptr*	*ucbptr:* RS-type address or address in register (2) - (12).
,BASICSIZE=*basicsize*	*basicsize:* RS-type address or address in register (2) - (12).
,BASICINFO=*basicinfo*	*basicinfo:* RS-type address or address in register (2) - (12).
,RETINFO=*retinfo*	*retinfo:* RS-type address or address in register (2) - (12).
,ALET=*alet*	*.alet:* RS-type address or address in register (2) - (12). **Default:** ALET=0.
,ASYNCH=*asynch*	*.asynch:* RS-type address or address in register (2) - (12). **Default:** ASYNCH=NO.
,ECB=*ecb*	*.ecb:* RS-type address or address in register (2) - (12). **Default:** ECB=NO_ECB.
,EXTNDSIZE=*extndsize*	*extndsize:* RS-type address or address in register (2) - (12). **Default:** EXTNDSIZE=0.

,WAITTIME=*waittime*	*.waittime:* RS-type address or address in register (2) - (12). **Default:** WAITTIME=0.
,EXTNDINFO=*extndinfo*	*extndinfo:* RS-type address or address in register (2) - (12).
REQUEST=SQRYDVCS	
,QRYSIZE=*qrysize*	*qrysize:* RS-type address or address in register (2) - (12).
,QRYINFO=*qryinfo*	*qryinfo:* RS-type address or address in register (2) - (12).
,RETINFO=*retinfo*	*retinfo:* RS-type address or address in register (2) - (12).
,ALET=*alet*	*.alet:* RS-type address or address in register (2) - (12). **Default:** ALET=0.
,ASYNCH=*asynch*	*.asynch:* RS-type address or address in register (2) - (12). **Default:** ASYNCH=NO.
,ECB=*ecb*	*.ecb:* RS-type address or address in register (2) - (12). **Default:** ECB=NO_ECB.
,SSFILTER=*ssfilter*	*.ssfilter:* RS-type address or address in register (2) - (12). **Default:** SSFILTER=NO_FILTER.
,SSFSIZE=*ssfsize*	*ssfsize:* RS-type address or address in register (2) - (12).
,VLFILTER=*vlfilter*	*.vlfilter:* RS-type address or address in register (2) - (12). **Default:** VLFILTER=NO_FILTER.
,VLFSIZE=*vlfsize*	*vlfsize:* RS-type address or address in register (2) - (12).
,LOWDEVN=*lowdevn*	*.lowdevn:* RS-type address or address in register (2) - (12). **Default:** LOWDEVN=NO_DEVN.
,HIGHDEVN=*highdevn*	*highdevn:* RS-type address or address in register (2) - (12).
,DEVTYPE=*devtype*	*.devtype:* RS-type address or address in register (2) - (12). **Default:** DEVTYPE=NO.
,PARTITION=*partition*	*.partition:* RS-type address or address in register (2) - (12). **Default:** PARTITION=NO.
,ACCESS=*access*	*.access:* RS-type address or address in register (2) - (12). **Default:** ACCESS=NO.
,BEGINCYL0=*begincyl0*	*.begincyl0:* RS-type address or address in register (2) - (12). **Default:** BEGINCYL0=YES.
,DEVN=*devn*	*.devn:* RS-type address or address in register (2) - (12). **Default:** DEVN=NO_DEVN.
,VOLSER=*volser*	*volser:* RS-type address or address in register (2) - (12).
,UCBPTR=*ucbptr*	*ucbptr:* RS-type address or address in register (2) - (12).
,WAITTIME=*waittime*	*.waittime:* RS-type address or address in register (2) - (12). **Default:** WAITTIME=0.
REQUEST=SQRYSSYS	
,QRYSIZE=*qrysize*	*qrysize:* RS-type address or address in register (2) - (12).
,QRYINFO=*qryinfo*	*qryinfo:* RS-type address or address in register (2) - (12).

,RETINFO=*retinfo*	*retinfo:* RS-type address or address in register (2) - (12).
,ALET=*alet*	*.alet:* RS-type address or address in register (2) - (12). **Default:** ALET=0.
,ASYNCH=*asynch*	*.asynch:* RS-type address or address in register (2) - (12). **Default:** ASYNCH=NO.
,ECB=*ecb*	*.ecb:* RS-type address or address in register (2) - (12). **Default:** ECB=NO_ECB.
,SSFILTER=*ssfilter*	*.ssfilter:* RS-type address or address in register (2) - (12). **Default:** SSFILTER=NO_FILTER.
,SSFSIZE=*ssfsize*	*ssfsize:* RS-type address or address in register (2) - (12).
,WAITTIME=*waittime*	*.waittime:* RS-type address or address in register (2) - (12). **Default:** WAITTIME=0.
REQUEST=SRELEASE	
,DEVN=*devn*	*devn:* RS-type address or address in register (2) - (12).
,VOLSER=*volser*	*volser:* RS-type address or address in register (2) - (12).
,UCBPTR=*ucbptr*	*ucbptr:* RS-type address or address in register (2) - (12).
,EXTINFO=*extinfo*	*extinfo:* RS-type address or address in register (2) - (12).
,EXTNBR=*extnbr*	*extnbr:* RS-type address or address in register (2) - (12).
,RETINFO=*retinfo*	*retinfo:* RS-type address or address in register (2) - (12).
,ASYNCH=*asynch*	*.asynch:* RS-type address or address in register (2) - (12). **Default:** ASYNCH=NO.
,ECB=*ecb*	*.ecb:* RS-type address or address in register (2) - (12). **Default:** ECB=NO_ECB.
,WAITTIME=*waittime*	*.waittime:* RS-type address or address in register (2) - (12). **Default:** WAITTIME=0.
REQUEST=SSNAP	
,SDEVN=*sdevn*	*sdevn:* RS-type address or address in register (2) - (12).
,SVOLSER=*svolser*	*svolser:* RS-type address or address in register (2) - (12).
,SUCBPTR=*sucbptr*	*sucbptr:* RS-type address or address in register (2) - (12).
,SEXTINFO=*sextinfo*	*sextinfo:* RS-type address or address in register (2) - (12).
,SEXTNBR=*sextnbr*	*sextnbr:* RS-type address or address in register (2) - (12).
,TDEVN=*tdevn*	*tdevn:* RS-type address or address in register (2) - (12).
,TVOLSER=*tvolser*	*tvolser:* RS-type address or address in register (2) - (12).
,TUCBPTR=*tucbptr*	*tucbptr:* RS-type address or address in register (2) - (12).
,TEXTINFO=*textinfo*	*textinfo:* RS-type address or address in register (2) - (12).
,TEXTNBR=*textnbr*	*textnbr:* RS-type address or address in register (2) - (12).
,RETINFO=*retinfo*	*retinfo:* RS-type address or address in register (2) - (12).
,ASYNCH=*asynch*	*.asynch:* RS-type address or address in register (2) - (12). **Default:** ASYNCH=NO.
,ECB=*ecb*	*.ecb:* RS-type address or address in register (2) - (12). **Default:** ECB=NO_ECB.

,TRELOCATE=*trelocate*	*.trelocate:* RS-type address or address in register (2) - (12). **Default:** TRELOCATE=NO.
,WAITTIME=*waittime*	*.waittime:* RS-type address or address in register (2) - (12). **Default:** WAITTIME=0.
REQUEST=XCONTIME	
,SID=*sid*	*sid:* RS-type address or address in register (2) - (12).
,RETINFO=*retinfo*	*retinfo:* RS-type address or address in register (2) - (12).
,TIME=*time*	*time:* RS-type address or address in register (2) - (12).
,ALET=*alet*	*.alet:* RS-type address or address in register (2) - (12). **Default:** ALET=0.
,ASYNCH=*asynch*	*.asynch:* RS-type address or address in register (2) - (12). **Default:** ASYNCH=NO.
,ECB=*ecb*	*.ecb:* RS-type address or address in register (2) - (12). **Default:** ECB=NO_ECB.
,HLQ=*hlq*	*.hlq:* RS-type address or address in register (2) - (12). **Default:** HLQ=SYS1.
,WAITTIME=*waittime*	*.waittime:* RS-type address or address in register (2) - (12). **Default:** WAITTIME=0.
,RETCODE=*retcode*	*retcode:* RS-type address or register (2) - (12).
,RSNCODE=*rsncode*	*rsncode:* RS-type address or register (2) - (12).
,PLISTVER=MAX	**Default:** PLISTVER=IMPLIED_VERSION.
,PLISTVER=0	
,MF=S	**Default:** MF=S.
,MF=(L,*list addr*)	*list addr:* RS-type addr. or register (1)-(12)
,MF=(L,*list addr,attr*)	**Default:** MF=(L,*list addr*,X'0D').
,MF=(E,*list addr*)	
,MF=(E,*list addr*,NOCHECK)	**Default:** MF=(E,*list addr*,COMPLETE).
,MF=(M,*list addr*)	
,MF=(M,*list addr*,NOCHECK)	**Default:** MF=(M,*list addr*,COMPLETE).

Parameters

The parameters are explained as follows:

name
An optional symbol, starting in column 1, that is the name on the ANTRQST macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

REQUEST=LEVEL asks for the level of ANTRQST that is installed on the system.

REQUEST=PDELPAIR asks for the CDELPAIR function of PPRC to be executed.

REQUEST=PDELPATH asks for the CDELPATH function of PPRC to be executed.

REQUEST=PESTPAIR asks for the CESTPAIR function of PPRC to be executed.

REQUEST=PESTPATH asks for the CESTPATH function of PPRC to be executed.

REQUEST=PFREEZE asks for the CGROUP Freeze function of PPRC to be executed.

REQUEST=PQUERY asks for the CQUERY function of PPRC to be executed.

REQUEST=PRECOVER asks for the CRECOVER function of PPRC to be executed.

REQUEST=PRUN asks for the CGROUP Run function of PPRC to be executed.

REQUEST=PSUSPEND asks for the CSUSPEND function of PPRC to be executed.

REQUEST=SDVCINFO asks for information on a Ramac Virtual Array (RVA) device.

REQUEST=SQRYDVCS asks for information on the RVA devices attached to a host.

REQUEST=SQRYSSYS asks for a list of RVA subsystems attached to a host.

REQUEST=SRELEASE is used to release functional tracks on an RVA device.

REQUEST=SSNAP is used to snap (quickly copy) functional tracks from one RVA device to another.

REQUEST=XCONTIME asks for the current XRC consistency time.

Subparameters for REQUEST

REQUEST=LEVEL

Asks for the level of ANTRQST that is installed on the system.

,RETINFO=*retinfo*

A required output parameter field used to return the level of ANTRQST that is installed on the system. If ANTRQST is not installed, RQST_PC_NUMBER_ZERO (see ANTRQSTL) is returned in the RETCODE field. If ANTRQST is installed, one of the following level values is placed in the first four bytes of the RETINFO field.

Level 1 supports:

- PDELPAIR
- PDELPATH
- PESTPAIR
- PESTPATH
- PFREEZE
- PQUERY
- PRECOVER
- PRUN
- PSUSPEND
- XCONTIME

Level 2 supports **all Level 1 parameters, plus**:

- SDVCINFO
- SQRYDVCS
- SQRYSSYS
- SRELEASE
- SSNAP

REQUEST=PDELPAIR

Asks for the CDELPAIR function of PPRC to be executed.

,DEVN=*devn*

A required input parameter field containing the hexadecimal device number to use for I/O.

To code: Specify the RS-type address, or address in register (2)-(12), of a 2-character field.

,PCCA=*pcca*

A required input parameter field containing the hexadecimal channel connection address of the primary device.

To code: Specify the RS-type address, or address in register (2)-(12), of a 1-character field.

,PSERIAL=*pserial*

A required input parameter field containing the EBCDIC serial number of the primary storage control.

To code: Specify the RS-type address, or address in register (2)-(12), of a 12-character field.

,PSSID=*pssid*

A required input parameter field containing the hexadecimal subsystem ID of the primary storage control.

To code: Specify the RS-type address, or address in register (2)-(12), of a 2-character field.

,SCCA=*scca*

A required input parameter field containing the hexadecimal channel connection address of the secondary device.

To code: Specify the RS-type address, or address in register (2)-(12), of a 1-character field.

,SSERIAL=*sserial*

A required input parameter field containing the EBCDIC serial number of the secondary storage control.

To code: Specify the RS-type address, or address in register (2)-(12), of a 12-character field.

,SSSID=*sssid*

A required input parameter field containing the hexadecimal subsystem ID of the secondary storage control.

To code: Specify the RS-type address, or address in register (2)-(12), of a 2-character field.

,RETINFO=*retinfo*

A required output parameter field used to return detailed information about the results of *executing* the request. Information about the results of *scheduling* the request are returned in the RETCODE and RSNCODE fields. The area should be aligned on a word boundary. The first four bytes contain the return code and the second four bytes contain the reason code. The remainder of the information is dependent on the return and reason codes. See the coding example 'TESTER: Test ANTRQST' at the end of this

appendix for how to coordinate the RETCODE, RSNCODE, and RETINFO fields.

To code: Specify the RS-type address, or address in register (2)-(12), of a 100-character field.

,ASYNCH=*asynch*
,ASYNCH=NO
An optional input parameter field containing a keyword that indicates if the request is to be asynchronous. The value is left justified and padded on the right with blanks. If the field contains binary zeros the default will be taken.

NO indicates that the request is to be synchronous.

YES indicates that the request is to be asynchronous. The ECB specified will be posted upon completion of the request.

The default is NO.

To code: Specify the RS-type address, or address in register (2)-(12), of a 3-character field.

,ECB=*ecb*
,ECB=NO_ECB
An optional input parameter field that SDM will post for an asynchronous request. This field is ignored for synchronous requests. The default is NO_ECB.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field.

,WAITTIME=*waittime*
,WAITTIME=0
An optional input parameter field containing how long in seconds SDM will wait for a request to complete. A value of zero says do not time the request. If the time expires before the request is complete a value of RQST_WAITTIME_EXPIRED (see ANTRQSTL) will be returned. If the request is a synchronous request the value will be placed in the RTNCODE field. If the request is an asynchronous request the value will be used as the ECB post code. The default is 0.

To code: Specify the RS-type address, or address in register (2)-(12), of a halfword field.

REQUEST=PDELPATH
Asks for the CDELPATH function of PPRC to be executed.

,DEVN=*devn*
A required input parameter field containing the hexadecimal device number to use for I/O.

To code: Specify the RS-type address, or address in register (2)-(12), of a 2-character field.

,PSERIAL=*pserial*
A required input parameter field containing the EBCDIC serial number of the primary storage control.

To code: Specify the RS-type address, or address in register (2)-(12), of a 12-character field.

,PSSID=*pssid*

A required input parameter field containing the hexadecimal subsystem ID of the primary storage control.

To code: Specify the RS-type address, or address in register (2)-(12), of a 2-character field.

,SSERIAL=*sserial*

A required input parameter field containing the EBCDIC serial number of the secondary storage control.

To code: Specify the RS-type address, or address in register (2)-(12), of a 12-character field.

,SSSID=*sssid*

A required input parameter field containing the hexadecimal subsystem ID of the secondary storage control.

To code: Specify the RS-type address, or address in register (2)-(12), of a 2-character field.

,RETINFO=*retinfo*

A required output parameter field used to return detailed information about the results of *executing* the request. Information about the results of *scheduling* the request are returned in the RETCODE and RSNCODE fields. The area should be aligned on a word boundary. The first four bytes contain the return code and the second four bytes contain the reason code. The remainder of the information is dependent on the return and reason codes. See the coding example 'TESTER: Test ANTRQST' at the end of this appendix for how to coordinate the RETCODE, RSNCODE, and RETINFO fields.

To code: Specify the RS-type address, or address in register (2)-(12), of a 100-character field.

,ASYNCH=*asynch*
,ASYNCH=NO

An optional input parameter field containing a keyword that indicates if the request is to be asynchronous. The value is left justified and padded on the right with blanks. If the field contains binary zeros the default will be taken.

NO indicates that the request is to be synchronous.

YES indicates that the request is to be asynchronous. The ECB specified will be posted upon completion of the request.

The default is NO.

To code: Specify the RS-type address, or address in register (2)-(12), of a 3-character field.

,ECB=*ecb*
,ECB=NO_ECB

An optional input parameter field that SDM will post for an asynchronous request. This field is ignored for synchronous requests. The default is NO_ECB.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field.

,WAITTIME=*waittime*
,WAITTIME=0

An optional input parameter field containing how long in seconds SDM will wait for a request to complete. A value of zero says do not time the request. If the time expires before the request is complete a value of RQST_WAITTIME_EXPIRED (see ANTRQSTL) will be returned. If the request is a synchronous request the value will be placed in the RTNCODE field. If the request is an asynchronous request the value will be used as the ECB post code. The default is 0.

To code: Specify the RS-type address, or address in register (2)-(12), of a halfword field.

REQUEST=PESTPAIR

Asks for the CESTPAIR function of PPRC to be executed.

,DEVN=*devn*

A required input parameter field containing the hexadecimal device number to use for I/O.

To code: Specify the RS-type address, or address in register (2)-(12), of a 2-character field.

,PCCA=*pcca*

A required input parameter field containing the hexadecimal channel connection address of the primary device.

To code: Specify the RS-type address, or address in register (2)-(12), of a 1-character field.

,PSERIAL=*pserial*

A required input parameter field containing the EBCDIC serial number of the primary storage control.

To code: Specify the RS-type address, or address in register (2)-(12), of a 12-character field.

,PSSID=*pssid*

A required input parameter field containing the hexadecimal subsystem ID of the primary storage control.

To code: Specify the RS-type address, or address in register (2)-(12), of a 2-character field.

,SCCA=*scca*

A required input parameter field containing the hexadecimal channel connection address of the secondary device.

To code: Specify the RS-type address, or address in register (2)-(12), of a 1-character field.

,SSERIAL=*sserial*

A required input parameter field containing the EBCDIC serial number of the secondary storage control.

To code: Specify the RS-type address, or address in register (2)-(12), of a 12-character field.

,SSSID=*sssid*

A required input parameter field containing the hexadecimal subsystem ID of the secondary storage control.

To code: Specify the RS-type address, or address in register (2)-(12), of a 2-character field.

,RETINFO=*retinfo*

A required output parameter field used to return detailed information about the results of *executing* the request. Information about the results of *scheduling* the request are returned in the RETCODE and RSNCODE fields. The area should be aligned on a word boundary. The first four bytes contain the return code and the second four bytes contain the reason code. The remainder of the information is dependent on the return and reason codes. See the coding example 'TESTER: Test ANTRQST' at the end of this appendix for how to coordinate the RETCODE, RSNCODE, and RETINFO fields.

To code: Specify the RS-type address, or address in register (2)-(12), of a 100-character field.

,ASYNCH=*asynch*
,ASYNCH=NO

An optional input parameter field containing a keyword that indicates if the request is to be asynchronous. The value is left justified and padded on the right with blanks. If the field contains binary zeros the default will be taken.

NO indicates that the request is to be synchronous.

YES indicates that the request is to be asynchronous. The ECB specified will be posted upon completion of the request.

The default is NO.

To code: Specify the RS-type address, or address in register (2)-(12), of a 3-character field.

,ECB=*ecb*
,ECB=NO_ECB

An optional input parameter field that SDM will post for an asynchronous request. This field is ignored for synchronous requests. The default is NO_ECB.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field.

,CRIT=*crit*
,CRIT=NO

An optional input parameter field containing a keyword that indicates if this is a critical volume. The value is left justified and padded on the right with blanks. If the field contains binary zeros the default will be taken.

NO specifies that, following an I/O completion error, subsequent write requests to the PPRC primary volume are allowed.

YES specifies that if an I/O error occurs, subsequent writes are either allowed or not allowed, based on the 3990 maintenance panel setting.

The default is NO.

To code: Specify the RS-type address, or address in register (2)-(12), of a 3-character field.

,MODE=*mode*
,MODE=COPY

An optional input parameter field containing a keyword that specifies how a volume pair is to be setup. The value is left justified and padded on the right with blanks. If the field contains binary zeros the default will be taken.

COPY specifies that all tracks on the primary volume are to be copied to the secondary volume.

NOCOPY specifies that only those tracks on the primary volume that are updated after this request is executed are to be copied to the secondary volume.

RESYNC specifies that a suspended copy be reestablished.

The default is COPY.

To code: Specify the RS-type address, or address in register (2)-(12), of a 6-character field.

,MSGREQ=*msgreq*
,MSGREQ=NO

An optional input parameter field containing a keyword that specifies whether or not to wait for the initial copy of the volume to complete. The value is left justified and padded on the right with blanks. If the field contains binary zeros the default will be taken.

NO indicates to not wait.

YES indicates to wait.

The default is NO.

To code: Specify the RS-type address, or address in register (2)-(12), of a 3-character field.

,PACE=*pace*
,PACE=DEFAULT

An optional input parameter field that specifies the minimum number of tracks to be copied at one time before allowing another host interrupt. If a value is not supplied the default setup in the storage control will be used. The default is DEFAULT.

To code: Specify the RS-type address, or address in register (2)-(12), of an one-byte field.

,WAITTIME=*waittime*
,WAITTIME=0

An optional input parameter field containing how long in seconds SDM will wait for a request to complete. A value of zero says do not time the request. If the time expires before the request is complete a value of RQST_WAITTIME_EXPIRED (see ANTRQSTL) will be returned. If the request is a synchronous request the value will be placed in the RTNCODE field. If the request is an asynchronous request the value will be used as the ECB post code. The default is 0.

To code: Specify the RS-type address, or address in register (2)-(12), of a halfword field.

REQUEST=PESTPATH

Asks for the CESTPATH function of PPRC to be executed.

,DEVN=*devn*

A required input parameter field containing the hexadecimal device number to use for I/O.

To code: Specify the RS-type address, or address in register (2)-(12), of a 2-character field.

,LINK1=*link1*

A required input parameter field containing a hexadecimal path address from the application site storage control to the recovery site storage control.

To code: Specify the RS-type address, or address in register (2)-(12), of a 4-character field.

,LINK2=*link2*
,LINK2=0

An optional input parameter field containing a hexadecimal path address from the application site storage control to the recovery site storage control. If the field contains binary zeros the default will be taken. The default is 0.

To code: Specify the RS-type address, or address in register (2)-(12), of a 4-character field.

,LINK3=*link3*
,LINK3=0

When LINK2=*link2* is specified, an optional input parameter field containing a hexadecimal path address from the application site storage control to the recovery site storage control. If the field contains binary zeros the default will be taken. The default is 0.

To code: Specify the RS-type address, or address in register (2)-(12), of a 4-character field.

,LINK4=*link4*
,LINK4=0

When LINK3=*link3* and LINK2=*link2* are specified, an optional input parameter field containing a hexadecimal path address from the application site storage control to the recovery site storage control. If the field contains binary zeros the default will be taken. The default is 0.

To code: Specify the RS-type address, or address in register (2)-(12), of a 4-character field.

,PSERIAL=*pserial*

A required input parameter field containing the EBCDIC serial number of the primary storage control.

To code: Specify the RS-type address, or address in register (2)-(12), of a 12-character field.

,PSSID=*pssid*

A required input parameter field containing the hexadecimal subsystem ID of the primary storage control.

To code: Specify the RS-type address, or address in register (2)-(12), of a 2-character field.

,SSERIAL=*sserial*

A required input parameter field containing the EBCDIC serial number of the secondary storage control.

To code: Specify the RS-type address, or address in register (2)-(12), of a 12-character field.

,SSSID=*sssid*
A required input parameter field containing the hexadecimal subsystem ID of the secondary storage control.

To code: Specify the RS-type address, or address in register (2)-(12), of a 2-character field.

,RETINFO=*retinfo*
A required output parameter field used to return detailed information about the results of *executing* the request. Information about the results of *scheduling* the request are returned in the RETCODE and RSNCODE fields. The area should be aligned on a word boundary. The first four bytes contain the return code and the second four bytes contain the reason code. The remainder of the information is dependent on the return and reason codes. See the coding example 'TESTER: Test ANTRQST' at the end of this appendix for how to coordinate the RETCODE, RSNCODE, and RETINFO fields.

To code: Specify the RS-type address, or address in register (2)-(12), of a 100-character field.

,ASYNCH=*asynch*
,ASYNCH=NO
An optional input parameter field containing a keyword that indicates if the request is to be asynchronous. The value is left justified and padded on the right with blanks. If the field contains binary zeros the default will be taken.

NO indicates that the request is to be synchronous.

YES indicates that the request is to be asynchronous. The ECB specified will be posted upon completion of the request.

The default is NO.

To code: Specify the RS-type address, or address in register (2)-(12), of a 3-character field.

,ECB=*ecb*
,ECB=NO_ECB
An optional input parameter field that SDM will post for an asynchronous request. This field is ignored for synchronous requests. The default is NO_ECB.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field.

,WAITTIME=*waittime*
,WAITTIME=0
An optional input parameter field containing how long in seconds SDM will wait for a request to complete. A value of zero says do not time the request. If the time expires before the request is complete a value of RQST_WAITTIME_EXPIRED (see ANTRQSTL) will be returned. If the request is a synchronous request the value will be placed in the RTNCODE field. If the request is an asynchronous request the value will be used as the ECB post code. The default is 0.

To code: Specify the RS-type address, or address in register (2)-(12), of a halfword field.

REQUEST=PFREEZE
Asks for the CGROUP Freeze function of PPRC to be executed.

,DEVN=*devn*
A required input parameter field containing the hexadecimal device number to use for I/O.

To code: Specify the RS-type address, or address in register (2)-(12), of a 2-character field.

,PSERIAL=*pserial*
A required input parameter field containing the EBCDIC serial number of the primary storage control.

To code: Specify the RS-type address, or address in register (2)-(12), of a 12-character field.

,PSSID=*pssid*
A required input parameter field containing the hexadecimal subsystem ID of the primary storage control.

To code: Specify the RS-type address, or address in register (2)-(12), of a 2-character field.

,SSERIAL=*sserial*
A required input parameter field containing the EBCDIC serial number of the secondary storage control.

To code: Specify the RS-type address, or address in register (2)-(12), of a 12-character field.

,SSSID=*sssid*
A required input parameter field containing the hexadecimal subsystem ID of the secondary storage control.

To code: Specify the RS-type address, or address in register (2)-(12), of a 2-character field.

,RETINFO=*retinfo*
A required output parameter field used to return detailed information about the results of *executing* the request. Information about the results of *scheduling* the request are returned in the RETCODE and RSNCODE fields. The area should be aligned on a word boundary. The first four bytes contain the return code and the second four bytes contain the reason code. The remainder of the information is dependent on the return and reason codes. See the coding example 'TESTER: Test ANTRQST' at the end of this appendix for how to coordinate the RETCODE, RSNCODE, and RETINFO fields.

To code: Specify the RS-type address, or address in register (2)-(12), of a 100-character field.

,ASYNCH=*asynch*
,ASYNCH=NO
An optional input parameter field containing a keyword that indicates if the request is to be asynchronous. The value is left justified and padded on the right with blanks. If the field contains binary zeros the default will be taken.

NO indicates that the request is to be synchronous.

YES indicates that the request is to be asynchronous. The ECB specified will be posted upon completion of the request.

The default is NO.

To code: Specify the RS-type address, or address in register (2)-(12), of a 3-character field.

,ECB=*ecb*
,ECB=NO_ECB
An optional input parameter field that SDM will post for an asynchronous request. This field is ignored for synchronous requests. The default is NO_ECB.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field.

,WAITTIME=*waittime*
,WAITTIME=0
An optional input parameter field containing how long in seconds SDM will wait for a request to complete. A value of zero says do not time the request. If the time expires before the request is complete a value of RQST_WAITTIME_EXPIRED (see ANTRQSTL) will be returned. If the request is a synchronous request the value will be placed in the RTNCODE field. If the request is an asynchronous request the value will be used as the ECB post code. The default is 0.

To code: Specify the RS-type address, or address in register (2)-(12), of a halfword field.

REQUEST=PQUERY
Asks for the CQUERY function of PPRC to be executed.

,DEVN=*devn*
A required input parameter field containing the hexadecimal device number of the volume to be used for the query.

To code: Specify the RS-type address, or address in register (2)-(12), of a 2-character field.

,QRYSIZE=*qrysize*
A required input parameter field that tells how big the area is that QRYINFO refers to.

To code: Specify the RS-type address, or address in register (2)-(12), of a halfword field.

,QRYINFO=*qryinfo*
A required output parameter field to be used to return the result of the query. If the field *is not* big enough to contain the complete result a return code of RQST_PQUERY_QRYSIZE_TOO_SMALL (see ANTRQSTL) will be placed in the return code part of RETINFO, and the total amount of space required will be placed in the reason code part of RETINFO. If the field *is* big enough to contain the complete result a return code of RQST_PQUERY_QRYSIZE_BIG ENOUGH (see ANTRQSTL) will be placed in the return code part of RETINFO, and the total amount of space actually used will be placed in the reason code part of RETINFO.

For PATHS(NO) see documentation of message ANTP0025I for a description of the output. For PATHS(YES) see documentation of message ANTP0080I for a description of the output.

The optional ALET parameter may be used to specify the location of the QRYINFO field.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,RETINFO=*retinfo*

A required output parameter field used to return detailed information about the results of *executing* the request. Information about the results of *scheduling* the request are returned in the RETCODE and RSNCODE fields. The area should be aligned on a word boundary. The first four bytes contain the return code and the second four bytes contain the reason code. The remainder of the information is dependent on the return and reason codes. See the coding example 'TESTER: Test ANTRQST' at the end of this appendix for how to coordinate the RETCODE, RSNCODE, and RETINFO fields.

To code: Specify the RS-type address, or address in register (2)-(12), of a 100-character field.

,ALET=*alet*
,ALET=0

An optional input parameter field that specifies the ALET to be used if the QRYINFO field has an ALET different from the one associated with the caller. The default is 0.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field.

,ASYNCH=*asynch*
,ASYNCH=NO

An optional input parameter field containing a keyword that indicates if the request is to be asynchronous. The value is left justified and padded on the right with blanks. If the field contains binary zeros the default will be taken.

NO indicates that the request is to be synchronous.

YES indicates that the request is to be asynchronous. The ECB specified will be posted upon completion of the request.

The default is NO.

To code: Specify the RS-type address, or address in register (2)-(12), of a 3-character field.

,ECB=*ecb*
,ECB=NO_ECB

An optional input parameter field that SDM will post for an asynchronous request. This field is ignored for synchronous requests. The default is NO_ECB.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field.

,BITMAP=*bitmap*
,BITMAP=YES

An optional input parameter field containing a keyword that specifies whether to access the bitmap for a primary device when determining the percentage of synchronization for a pending or suspended volume pair. This field is used only if DEVN specifies a primary volume. The value is left justified and padded on the right with blanks. If the field contains binary zeros the default will be taken.

NO directs PPRC to not read the bitmap for the primary volume specified by DEVN.

YES directs PPRC to read the primary volume's bitmap.

The default is YES.

Note: If a PFREEZE command has been issued to this device, the I/O to read the bitmap will not complete until a PRUN command is issued to this device. Use the WAITTIME parameter to return to the program that executed ANTRQST.

To code: Specify the RS-type address, or address in register (2)-(12), of a 3-character field.

,PATHS=*paths*
,PATHS=NO

An optional input parameter field containing a keyword that specifies whether to specify volume information or path information. The value is left justified and padded on the right with blanks. If the field contains binary zeros the default will be taken.

NO directs PPRC to display the status of the volume specified by DEVN.

YES directs PPRC to display all of the paths associated with the application site storage control, and the status of each path.

The default is NO.

To code: Specify the RS-type address, or address in register (2)-(12), of a 3-character field.

,WAITTIME=*waittime*
,WAITTIME=0

An optional input parameter field containing how long in seconds SDM will wait for a request to complete. A value of zero says do not time the request. If the time expires before the request is complete a value of RQST_WAITTIME_EXPIRED (see ANTRQSTL) will be returned. If the request is a synchronous request the value will be placed in the RTNCODE field. If the request is an asynchronous request the value will be used as the ECB post code. The default is 0.

To code: Specify the RS-type address, or address in register (2)-(12), of a halfword field.

REQUEST=PRECOVER

Asks for the CRECOVER function of PPRC to be executed.

,DEVN=*devn*

A required input parameter field containing the hexadecimal device number to use for I/O.

To code: Specify the RS-type address, or address in register (2)-(12), of a 2-character field.

,PCCA=*pcca*
A required input parameter field containing the hexadecimal channel connection address of the primary device.

To code: Specify the RS-type address, or address in register (2)-(12), of a 1-character field.

,PSERIAL=*pserial*
A required input parameter field containing the EBCDIC serial number of the primary storage control.

To code: Specify the RS-type address, or address in register (2)-(12), of a 12-character field.

,PSSID=*pssid*
A required input parameter field containing the hexadecimal subsystem ID of the primary storage control.

To code: Specify the RS-type address, or address in register (2)-(12), of a 2-character field.

,SCCA=*scca*
A required input parameter field containing the hexadecimal channel connection address of the secondary device.

To code: Specify the RS-type address, or address in register (2)-(12), of a 1-character field.

,SSERIAL=*sserial*
A required input parameter field containing the EBCDIC serial number of the secondary storage control.

To code: Specify the RS-type address, or address in register (2)-(12), of a 12-character field.

,SSSID=*sssid*
A required input parameter field containing the hexadecimal subsystem ID of the secondary storage control.

To code: Specify the RS-type address, or address in register (2)-(12), of a 2-character field.

,RETINFO=*retinfo*
A required output parameter field used to return detailed information about the results of *executing* the request. Information about the results of *scheduling* the request are returned in the RETCODE and RSNCODE fields. The area should be aligned on a word boundary. The first four bytes contain the return code and the second four bytes contain the reason code. The remainder of the information is dependent on the return and reason codes. See the coding example 'TESTER: Test ANTRQST' at the end of this appendix for how to coordinate the RETCODE, RSNCODE, and RETINFO fields.

To code: Specify the RS-type address, or address in register (2)-(12), of a 100-character field.

,ASYNCH=*asynch*
,ASYNCH=NO

An optional input parameter field containing a keyword that indicates if the request is to be asynchronous. The value is left justified and padded on the right with blanks. If the field contains binary zeros the default will be taken.

NO indicates that the request is to be synchronous.

YES indicates that the request is to be asynchronous. The ECB specified will be posted upon completion of the request.

The default is NO.

To code: Specify the RS-type address, or address in register (2)-(12), of a 3-character field.

,ECB=*ecb*
,ECB=NO_ECB

An optional input parameter field that SDM will post for an asynchronous request. This field is ignored for synchronous requests. The default is NO_ECB.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field.

,OLDVOLSER=*oldvolser*
,OLDVOLSER=NO_VOLSER

An optional input parameter field that specifies the old volume serial number. If NEWVOLSER is not specified, PPRC verifies the serial number and resets the volume to simplex state. If the field contains binary zeros the default will be taken. The default is NO_VOLSER.

To code: Specify the RS-type address, or address in register (2)-(12), of a 6-character field.

,NEWVOLSER=*newvolser*
,NEWVOLSER=NO_VOLSER

When OLDVOLSER=*oldvolser* is specified, an optional input parameter field that specifies the new volume serial number. PPRC verifies the old serial number, resets the volume to the simplex state, then replaces the old serial number with the new serial number. If the field contains binary zeros the default will be taken. The default is NO_VOLSER.

To code: Specify the RS-type address, or address in register (2)-(12), of a 6-character field.

,WAITTIME=*waittime*
,WAITTIME=0

An optional input parameter field containing how long in seconds SDM will wait for a request to complete. A value of zero says do not time the request. If the time expires before the request is complete a value of RQST_WAITTIME_EXPIRED (see ANTRQSTL) will be returned. If the request is a synchronous request the value will be placed in the RTNCODE field. If the request is an asynchronous request the value will be used as the ECB post code. The default is 0.

To code: Specify the RS-type address, or address in register (2)-(12), of a halfword field.

REQUEST=PRUN

Asks for the CGROUP Run function of PPRC to be executed.

,DEVN=*devn*

A required input parameter field containing the hexadecimal device number to use for I/O.

To code: Specify the RS-type address, or address in register (2)-(12), of a 2-character field.

,PSERIAL=*pserial*

A required input parameter field containing the EBCDIC serial number of the primary storage control.

To code: Specify the RS-type address, or address in register (2)-(12), of a 12-character field.

,PSSID=*pssid*

A required input parameter field containing the hexadecimal subsystem ID of the primary storage control.

To code: Specify the RS-type address, or address in register (2)-(12), of a 2-character field.

,SSERIAL=*sserial*

A required input parameter field containing the EBCDIC serial number of the secondary storage control.

To code: Specify the RS-type address, or address in register (2)-(12), of a 12-character field.

,SSSID=*sssid*

A required input parameter field containing the hexadecimal subsystem ID of the secondary storage control.

To code: Specify the RS-type address, or address in register (2)-(12), of a 2-character field.

,RETINFO=*retinfo*

A required output parameter field used to return detailed information about the results of *executing* the request. Information about the results of *scheduling* the request are returned in the RETCODE and RSNCODE fields. The area should be aligned on a word boundary. The first four bytes contain the return code and the second four bytes contain the reason code. The remainder of the information is dependent on the return and reason codes. See the coding example 'TESTER: Test ANTRQST' at the end of this appendix for how to coordinate the RETCODE, RSNCODE, and RETINFO fields.

To code: Specify the RS-type address, or address in register (2)-(12), of a 100-character field.

,ASYNCH=*asynch*
,ASYNCH=<u>NO</u>

An optional input parameter field containing a keyword that indicates if the request is to be asynchronous. The value is left justified and padded on the right with blanks. If the field contains binary zeros the default will be taken.

NO indicates that the request is to be synchronous.

YES indicates that the request is to be asynchronous. The ECB specified will be posted upon completion of the request.

The default is NO.

To code: Specify the RS-type address, or address in register (2)-(12), of a 3-character field.

,ECB=*ecb*
,ECB=NO_ECB
An optional input parameter field that SDM will post for an asynchronous request. This field is ignored for synchronous requests. The default is NO_ECB.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field.

,WAITTIME=*waittime*
,WAITTIME=0
An optional input parameter field containing how long in seconds SDM will wait for a request to complete. A value of zero says do not time the request. If the time expires before the request is complete a value of RQST_WAITTIME_EXPIRED (see ANTRQSTL) will be returned. If the request is a synchronous request the value will be placed in the RTNCODE field. If the request is an asynchronous request the value will be used as the ECB post code. The default is 0.

To code: Specify the RS-type address, or address in register (2)-(12), of a halfword field.

REQUEST=PSUSPEND
Asks for the CSUSPEND function of PPRC to be executed.

,DEVN=*devn*
A required input parameter field containing the hexadecimal device number to use for I/O.

To code: Specify the RS-type address, or address in register (2)-(12), of a 2-character field.

,PRIMARY=*primary*
,PRIMARY=NO
An optional input parameter field containing a keyword that indicates ICKDSF media maintenance is wanted. The value is left justified and padded on the right with blanks. If the field contains binary zeros the default will be taken.

NO indicates that the request is a normal suspend.

YES directs PPRC to suspend the PPRC volume pair and to reject all write data I/O to the primary volume except for ICKDSF media maintenance channel programs.

The default is NO.

To code: Specify the RS-type address, or address in register (2)-(12), of a 3-character field.

,QUIESCE=*quiesce*
,QUIESCE=NO

An optional input parameter field containing a keyword that directs PPRC to suspend the PPRC volume and to send a “state change pending” indication to OS/390 to prevent issuance of any application system I/O. The value is left-justified and padded on the right with blanks. If the field contains binary zeros the default will be taken.

NO indicates that quiesce is not wanted.

YES indicates quiesce is wanted.

The default is NO.

To code: Specify the RS-type address, or address in register (2)-(12), of a 3-character field.

,PCCA=*pcca*

A required input parameter field containing the hexadecimal channel connection address of the primary device.

To code: Specify the RS-type address, or address in register (2)-(12), of a 1-character field.

,PSERIAL=*pserial*

A required input parameter field containing the EBCDIC serial number of the primary storage control.

To code: Specify the RS-type address, or address in register (2)-(12), of a 12-character field.

,PSSID=*pssid*

A required input parameter field containing the hexadecimal subsystem ID of the primary storage control.

To code: Specify the RS-type address, or address in register (2)-(12), of a 2-character field.

,SCCA=*scca*

A required input parameter field containing the hexadecimal channel connection address of the secondary device.

To code: Specify the RS-type address, or address in register (2)-(12), of a 1-character field.

,SSERIAL=*sserial*

A required input parameter field containing the EBCDIC serial number of the secondary storage control.

To code: Specify the RS-type address, or address in register (2)-(12), of a 12-character field.

,SSSID=*sssid*

A required input parameter field containing the hexadecimal subsystem ID of the secondary storage control.

To code: Specify the RS-type address, or address in register (2)-(12), of a 2-character field.

,RETINFO=*retinfo*

A required output parameter field used to return detailed information about the results of *executing* the request. Information about the results of *sched-*

uling the request are returned in the RETCODE and RSNCODE fields. The area should be aligned on a word boundary. The first four bytes contain the return code and the second four bytes contain the reason code. The remainder of the information is dependent on the return and reason codes. See the coding example 'TESTER: Test ANTRQST' at the end of this appendix for how to coordinate the RETCODE, RSNCODE, and RETINFO fields.

To code: Specify the RS-type address, or address in register (2)-(12), of a 100-character field.

,ASYNCH=*asynch*
,ASYNCH=NO

An optional input parameter field containing a keyword that indicates if the request is to be asynchronous. The value is left justified and padded on the right with blanks. If the field contains binary zeros the default will be taken.

NO indicates that the request is to be synchronous.

YES indicates that the request is to be asynchronous. The ECB specified will be posted upon completion of the request.

The default is NO.

To code: Specify the RS-type address, or address in register (2)-(12), of a 3-character field.

,ECB=*ecb*
,ECB=NO_ECB

An optional input parameter field that SDM will post for an asynchronous request. This field is ignored for synchronous requests. The default is NO_ECB.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field.

,WAITTIME=*waittime*
,WAITTIME=0

An optional input parameter field containing how long in seconds SDM will wait for a request to complete. A value of zero says do not time the request. If the time expires before the request is complete a value of RQST_WAITTIME_EXPIRED (see ANTRQSTL) will be returned. If the request is a synchronous request the value will be placed in the RTNCODE field. If the request is an asynchronous request the value will be used as the ECB post code. The default is 0.

To code: Specify the RS-type address, or address in register (2)-(12), of a halfword field.

REQUEST=SDVCINFO

Asks for information on an RVA device.

,DEVN=*devn*
,VOLSER=*volser*
,UCBPTR=*ucbptr*

One of the following three input parameters is required:

,DEVN=*devn*

A parameter field containing the hexadecimal device number of the device you are interested in.

To code: Specify the RS-type address, or address in register (2)-(12), of a 2-character field.

,VOLSER=*volser*
A parameter field containing the volser of the device you are interested in.

To code: Specify the RS-type address, or address in register (2)-(12), of a 6-character field.

,UCBPTR=*ucbptr*
A parameter field containing the UCB address of the device you are interested in.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field.

,BASICSIZE=*basicsize*
A required input field that indicates the size of the area that BASICINFO refers to. The minimum size that can be specified is the length of PSL10_HEADER (see BASICINFO).

To code: Specify the RS-type address of a halfword field, or register (2)-(12).

,BASICINFO=*basicinfo*
A required output parameter field used to contain the result of the query for the device you are interested in. The information is described in mapping macro ANTPSL10, which is included as part of Snapshot support. If the field is large enough to hold all of the data, PSL10_DATA_USED tells how much data was returned. If the field is not large enough, PSL10_DATA_TRUNC is turned on, PSL10_DATA_USED tells how much space was required, and a return code of RQST_SDVCINFO_BASICSIZE_TOO_SMALL (see ANTRQSTL) is placed in the return code part of RETINFO.

The optional ALET parameter can be used to specify the location of the BASICINFO field.

To code: Specify the RS-type address of a variable-length field, or register (2)-(12).

,RETINFO=*retinfo*
A required output parameter field used to return detailed information about the results of *executing* the request. Information about the results of *scheduling* the request are returned in the RETCODE and RSNCODE fields. The area should be aligned on a word boundary. The first four bytes contain the return code and the second four bytes contain the reason code. The remainder of the information is dependent on the return and reason codes. See the coding example 'TESTER: Test ANTRQST' at the end of this appendix for how to coordinate the RETCODE, RSNCODE, and RETINFO fields.

To code: Specify the RS-type address, or address in register (2)-(12), of a 100-character field.

,ALET=*alet*
,ALET=0

An optional input parameter field that specifies the ALET to be used if the BASICINFO and EXTNDINFO fields have an ALET different from the one associated with the caller. The default is 0.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field.

,ASYNCH=*asynch*
,ASYNCH=NO

An optional input parameter field containing a keyword that indicates if the request is to be asynchronous. The value is left justified and padded on the right with blanks. If the field contains binary zeros the default will be taken.

NO indicates that the request is to be synchronous.

YES indicates that the request is to be asynchronous. The ECB specified will be posted upon completion of the request.

The default is NO.

To code: Specify the RS-type address, or address in register (2)-(12), of a 3-character field.

,ECB=*ecb*
,ECB=NO_ECB

An optional input parameter field that SDM will post for an asynchronous request. This field is ignored for synchronous requests. The default is NO_ECB.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field.

,EXTNDSIZE=*extndsize*
,EXTNDSIZE=0

An optional input field that indicates the size of the area that EXTNDINFO refers to. The minimum size that can be specified is the length of PSL11_HEADER (see EXTNDINFO). The default is 0.

To code: Specify the RS-type address, or address in register (2)-(12), of a halfword field.

,WAITTIME=*waittime*
,WAITTIME=0

An optional input parameter field containing how long in seconds SDM will wait for a request to complete. A value of zero says do not time the request. If the time expires before the request is complete a value of RQST_WAITTIME_EXPIRED (see ANTRQSTL) will be returned. If the request is a synchronous request the value will be placed in the RTNCODE field. If the request is an asynchronous request the value will be used as the ECB post code. The default is 0.

To code: Specify the RS-type address, or address in register (2)-(12), of a halfword field.

,EXTNDINFO=*extndinfo*

An optional output field used to contain the extended information for the device you are interested in. The information is described in mapping macro ANTPSL11, which is included as part of Snapshot support. If the

field is large enough to hold all of the data, PSL11_DATA_USED tells how much data was returned. If the field is not large enough, PSL11_DATA_TRUNC is turned on, PSL11_DATA_USED tells how much space was required, and a return code of RQST_SDVCINFO_EXTNDSIZE_TOO_SMALL (see ANTRQSTL) is placed in the return code part of RETINFO.

The optional ALET parameter can be used to specify the location of the EXTNDINFO field.

To code: Specify the RS-type address, or address in register (2)-(12), of a variable-length field.

REQUEST=SQRYDVCS

Asks for information on the RVA devices attached to a host.

,QRYSIZE=*qrysize*

A required input parameter field that tells how big the area is that QRYINFO refers to. The minimum size that can be specified is the length of PSL30_HEADER (see QRYINFO).

To code: Specify the RS-type address, or address in register (2)-(12), of a halfword field.

,QRYINFO=*qryinfo*

A required output parameter field to be used to return the result of the query. The returned information is described in mapping macro ANTPSL30, which is included as part of Snapshot support. If the field is large enough to hold all of the data PSL30_DATA_USED will tell how much data was returned. If the field was not large enough, PSL30_DATA_TRUNC will be turned on, PSL30_DATA_USED will tell how much space was required, and a return code of RQST_SQRYDVCS_QRYSIZE_TOO_SMALL (see ANTRQSTL) will be placed in the return code part of RETINFO.

The optional ALET parameter may be used to specify the location of the QRYINFO field.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,RETINFO=*retinfo*

A required output parameter field used to return detailed information about the results of *executing* the request. Information about the results of *scheduling* the request are returned in the RETCODE and RSNCODE fields. The area should be aligned on a word boundary. The first four bytes contain the return code and the second four bytes contain the reason code. The remainder of the information is dependent on the return and reason codes. See the coding example 'TESTER: Test ANTRQST' at the end of this appendix for how to coordinate the RETCODE, RSNCODE, and RETINFO fields.

To code: Specify the RS-type address, or address in register (2)-(12), of a 100-character field.

,ALET=*alet*
,ALET=0

An optional input parameter field that specifies the ALET to be used if the QRYINFO field has an ALET different from the one associated with the caller. The default is 0.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field.

,ASYNCH=*asynch*
,ASYNCH=NO

An optional input parameter field containing a keyword that indicates if the request is to be asynchronous. The value is left justified and padded on the right with blanks. If the field contains binary zeros the default will be taken.

NO indicates that the request is to be synchronous.

YES indicates that the request is to be asynchronous. The ECB specified will be posted upon completion of the request.

The default is NO.

To code: Specify the RS-type address, or address in register (2)-(12), of a 3-character field.

,ECB=*ecb*
,ECB=NO_ECB

An optional input parameter field that SDM will post for an asynchronous request. This field is ignored for synchronous requests. The default is NO_ECB.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field.

,SSFILTER=*ssfilter*
,SSFILTER=NO_FILTER

An optional input parameter field containing a filter to be used in determining what subsystems the devices being listed may come from. If this parameter is not specified, or the value in SSFSIZE is zero, subsystem names will not be considered when determining device eligibility. The rules for filter patterns are in item “Character Filtering” on page 407. The default is NO_FILTER.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,SSFSIZE=*ssfsize*

When SSFILTER=*ssfilter* is specified, a required input parameter field that tells how big the area is that SSFILTER refers to. If the value in the field is zero filtering will not be done.

To code: Specify the RS-type address, or address in register (2)-(12), of a halfword field.

,VLFILTER=*vlfilter*
,VLFILTER=NO_FILTER

An optional input parameter field containing a filter to be used in determining what volumes the devices being listed may come from. If this parameter is not specified, or the value in VLFSIZE is zero, volume serial numbers will not be considered when determining device eligibility. The rules for filter patterns are in item “Character Filtering” on page 407. The default is NO_FILTER.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,VLFSIZE=*vlfsize*

When VLFILTER=*vlfilter* is specified, a required input parameter field that tells how big the area is that VLFILTER refers to. If the value in the field is zero filtering will not be done.

To code: Specify the RS-type address, or address in register (2)-(12), of a halfword field.

,LOWDEVN=*lowdevn*
,LOWDEVN=NO_DEVN

An optional input parameter field containing the low hexadecimal device number of a range of devices you are interested in. If the field contains binary zeros the default will be taken. The default is NO_DEVN.

To code: Specify the RS-type address, or address in register (2)-(12), of a 2-character field.

,HIGHDEVN=*highdevn*

When LOWDEVN=*lowdevn* is specified, a required input parameter field. It contains the high hexadecimal device number of a range of devices you are interested in.

To code: Specify the RS-type address, or address in register (2)-(12), of a 2-character field.

,DEVTYPE=*devtype*
,DEVTYPE=NO

An optional input parameter field containing a keyword that indicates whether or not device type is to be used as a selection criteria. The value is left justified and padded on the right with blanks. If the field contains binary zeros the default will be taken.

NO indicates that device type is not to be considered.

3380 indicates that selected devices must have a device type of 3380.

3390 indicates that selected devices must have a device type of 3390.

The default is NO.

To code: Specify the RS-type address, or address in register (2)-(12), of a 4-character field.

,PARTITION=*partition*
,PARTITION=NO

An optional input parameter field containing a keyword that indicates whether or not RVA partition is to be used as a selection criteria. The value is left justified and padded on the right with blanks. If the field contains binary zeros the default will be taken.

NO indicates that partition type is not to be considered.

TEST indicates that selected devices must be in a test partition.

PROD indicates that selected devices must be in a production partition.

The default is NO.

To code: Specify the RS-type address, or address in register (2)-(12), of a 4-character field.

,ACCESS=*access*
,ACCESS=NO

An optional input parameter field containing a keyword that indicates whether or not read/write status is to be used as a selection criteria. The value is left justified and padded on the right with blanks. If the field contains binary zeros the default will be taken.

NO indicates that read/write status is not to be considered.

R indicates that selected devices must be read-only devices.

RW indicates that selected devices must be read/write devices.

The default is NO.

To code: Specify the RS-type address, or address in register (2)-(12), of a 2-character field.

,BEGINCYL0=*begincyl0*
,BEGINCYL0=YES

An optional input parameter field containing a keyword that indicates devices who do or do not begin on real cylinder 0 are to be included in the device list. The value is left justified and padded on the right with blanks. If the field contains binary zeros the default will be taken.

YES indicates that *only* devices who begin at real cylinder 0 are eligible to be in the list.

NO indicates that *all* devices, whether or not they begin at real cylinder 0 (e.g. VM minidisks), are eligible to be in the list.

The default is YES.

To code: Specify the RS-type address, or address in register (2)-(12), of a 3-character field.

,DEVN=*devn*
,DEVN=NO_DEVN
,VOLSER=*volser*
,UCBPTR=*ucbptr*

Only one of the following three optional input parameters may be coded:

,DEVN=*devn*

A parameter field containing the hexadecimal device number of a device. All devices on the same subsystem as the specified device are eligible to be in the list. The default is NO_DEVN.

To code: Specify the RS-type address, or address in register (2)-(12), of a 2-character field.

,VOLSER=*volser*

A parameter field containing the volser of a device. All devices on the same subsystem as the specified device are eligible to be in the list.

To code: Specify the RS-type address, or address in register (2)-(12), of a 6-character field.

,UCBPTR=*ucbptr*

A parameter field containing the UCB address of a device. All devices on the same subsystem as the specified device are eligible to be in the list.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field.

,WAITTIME=*waittime*
,WAITTIME=0

An optional input parameter field containing how long in seconds SDM will wait for a request to complete. A value of zero says do not time the request. If the time expires before the request is complete a value of RQST_WAITTIME_EXPIRED (see ANTRQSTL) will be returned. If the request is a synchronous request the value will be placed in the RTNCODE field. If the request is an asynchronous request the value will be used as the ECB post code. The default is 0.

To code: Specify the RS-type address, or address in register (2)-(12), of a halfword field.

REQUEST=SQRYSSYS

Asks for a list of RVA subsystems attached to a host.

,QRYSIZE=*qrysize*

A required input parameter field that tells how big the area is that QRYINFO refers to. The minimum size that can be specified is the length of PSL20_HEADER (see QRYINFO).

To code: Specify the RS-type address, or address in register (2)-(12), of a halfword field.

,QRYINFO=*qryinfo*

A required output parameter field to be used to return the result of the query. The returned information is described in mapping macro ANTPSL20, which is included as part of Snapshot support. If the field is large enough to hold all of the data PSL20_DATA_USED will tell how much data was returned. If the field was not large enough, PSL20_DATA_TRUNC will be turned on, PSL20_DATA_USED will tell how much space was required, and a return code of RQST_SQRYDVCS_QRYSIZE_TOO_SMALL (see ANTRQSTL) will be placed in the return code part of RETINFO.

The optional ALET parameter may be used to specify the location of the QRYINFO field.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,RETINFO=*retinfo*

A required output parameter field used to return detailed information about the results of *executing* the request. Information about the results of *scheduling* the request are returned in the RETCODE and RSNCODE fields. The area should be aligned on a word boundary. The first four bytes contain the return code and the second four bytes contain the reason code. The remainder of the information is dependent on the return and reason codes. See the coding example 'TESTER: Test ANTRQST' at the end of this appendix for how to coordinate the RETCODE, RSNCODE, and RETINFO fields.

To code: Specify the RS-type address, or address in register (2)-(12), of a 100-character field.

,ALET=*alet*
,ALET=0

An optional input parameter field that specifies the ALET to be used if the QRYINFO field has an ALET different from the one associated with the caller. The default is 0.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field.

,ASYNCH=*asynch*
,ASYNCH=NO

An optional input parameter field containing a keyword that indicates if the request is to be asynchronous. The value is left justified and padded on the right with blanks. If the field contains binary zeros the default will be taken.

NO indicates that the request is to be synchronous.

YES indicates that the request is to be asynchronous. The ECB specified will be posted upon completion of the request.

The default is NO.

To code: Specify the RS-type address, or address in register (2)-(12), of a 3-character field.

,ECB=*ecb*
,ECB=NO_ECB

An optional input parameter field that SDM will post for an asynchronous request. This field is ignored for synchronous requests. The default is NO_ECB.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field.

,SSFILTER=*ssfilter*
,SSFILTER=NO_FILTER

An optional input parameter field containing a filter to be used in determining what subsystem names to return. If this parameter is not specified, or the value in SSFSIZE is zero, all subsystem names will be returned. The rules for filter patterns are in item “Character Filtering” on page 407. The default is NO_FILTER.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,SSFSIZE=*ssfsize*

When SSFILTER=*ssfilter* is specified, a required input parameter field that tells how big the area is that SSFILTER refers to. If the value in the field is zero filtering will not be done.

To code: Specify the RS-type address, or address in register (2)-(12), of a halfword field.

,WAITTIME=*waittime*
,WAITTIME=0

An optional input parameter field containing how long in seconds SDM will wait for a request to complete. A value of zero says do not time the request. If the time expires before the request is complete a value of RQST_WAITTIME_EXPIRED (see ANTRQSTL) will be returned. If the request is a synchronous request the value will be placed in the RTNCODE

field. If the request is an asynchronous request the value will be used as the ECB post code. The default is 0.

To code: Specify the RS-type address, or address in register (2)-(12), of a halfword field.

REQUEST=SRELEASE
is used to release functional tracks on an RVA device.

,DEVN=*devn*
,VOLSER=*volser*
,UCBPTR=*ucbptr*
One of the following three input parameters is required:

,DEVN=*devn*
A parameter field containing the hexadecimal device number of a device whose functional tracks are to be released.

To code: Specify the RS-type address, or address in register (2)-(12), of a 2-character field.

,VOLSER=*volser*
A parameter field containing the volser of a device whose functional tracks are to be released.

To code: Specify the RS-type address, or address in register (2)-(12), of a 6-character field.

,UCBPTR=*ucbptr*
A parameter field containing the UCB address of a device whose functional tracks are to be released.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field.

,EXTINFO=*extinfo*
A required input parameter field that is an array of eight byte extent pairs to release. The first four bytes of a pair gives the starting track address and the last four bytes gives the ending track address (for a single track the two addresses would be the same).

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,EXTNBR=*extnbr*
A required input parameter field that tells how many extent pairs have been supplied in EXTINFO.

To code: Specify the RS-type address, or address in register (2)-(12), of a halfword field.

,RETINFO=*retinfo*
A required output parameter field used to return detailed information about the results of *executing* the request. Information about the results of *scheduling* the request are returned in the RETCODE and RSNCODE fields. The area should be aligned on a word boundary. The first four bytes contain the return code and the second four bytes contain the reason code. The remainder of the information is dependent on the return and reason codes. See the coding example 'TESTER: Test ANTRQST' at the end of this

appendix for how to coordinate the RETCODE, RSNCODE, and RETINFO fields.

To code: Specify the RS-type address, or address in register (2)-(12), of a 100-character field.

,ASYNCH=*asynch*
,ASYNCH=NO

An optional input parameter field containing a keyword that indicates if the request is to be asynchronous. The value is left justified and padded on the right with blanks. If the field contains binary zeros the default will be taken.

NO indicates that the request is to be synchronous.

YES indicates that the request is to be asynchronous. The ECB specified will be posted upon completion of the request.

The default is NO.

To code: Specify the RS-type address, or address in register (2)-(12), of a 3-character field.

,ECB=*ecb*
,ECB=NO_ECB

An optional input parameter field that SDM will post for an asynchronous request. This field is ignored for synchronous requests. The default is NO_ECB.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field.

,WAITTIME=*waittime*
,WAITTIME=0

An optional input parameter field containing how long in seconds SDM will wait for a request to complete. A value of zero says do not time the request. If the time expires before the request is complete a value of RQST_WAITTIME_EXPIRED (see ANTRQSTL) will be returned. If the request is a synchronous request the value will be placed in the RTNCODE field. If the request is an asynchronous request the value will be used as the ECB post code. The default is 0.

To code: Specify the RS-type address, or address in register (2)-(12), of a halfword field.

REQUEST=SSNAP

is used to snap (quickly copy) functional tracks from one RVA device to another.

,SDEVN=*sdevn*
,SVOLSER=*svolser*
,SUCBPTR=*sucbptr*

One of the following three input parameters is required:

,SDEVN=*sdevn*

A parameter field containing the hexadecimal device number of the source device you want to snap.

To code: Specify the RS-type address, or address in register (2)-(12), of a 2-character field.

,SVOLSER=*svolser*

A parameter field containing the volser of the source device you want to snap.

To code: Specify the RS-type address, or address in register (2)-(12), of a 6-character field.

,SUCBPTR=*sucbptr*

A parameter field containing the UCB address of the source device you want to snap.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field.

,SEXTINFO=*sextinfo*

A required input parameter field that is an array of eight byte source extent pairs to snap. The first four bytes of a pair gives the starting track address and the last four bytes gives the ending track address (for a single track the two addresses would be the same).

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,SEXTNBR=*sextnbr*

A required input parameter field that tells how many source extent pairs have been supplied in SEXTINFO.

To code: Specify the RS-type address, or address in register (2)-(12), of a halfword field.

,TDEVN=*tdevn*
,TVOLSER=*tvolser*
,TUCBPTR=*tucbptr*

One of the following three input parameters is required:

,TDEVN=*tdevn*

A parameter field containing the hexadecimal device number of the target device you want to snap.

To code: Specify the RS-type address, or address in register (2)-(12), of a 2-character field.

,TVOLSER=*tvolser*

A parameter field containing the volser of the target device you want to snap.

To code: Specify the RS-type address, or address in register (2)-(12), of a 6-character field.

,TUCBPTR=*tucbptr*

A parameter field containing the UCB address of the target device you want to snap.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field.

,TEXTINFO=*textinfo*

A required input parameter field that is an array of eight byte target extent pairs to snap. The first four bytes of a pair gives the starting track address and the last four bytes gives the ending track address (for a single track the two addresses would be the same).

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,TEXTNBR=*textnbr*
A required input parameter field that tells how many target extent pairs have been supplied in TEXTINFO.

To code: Specify the RS-type address, or address in register (2)-(12), of a halfword field.

,RETINFO=*retinfo*
A required output parameter field used to return detailed information about the results of *executing* the request. Information about the results of *scheduling* the request are returned in the RETCODE and RSNCODE fields. The area should be aligned on a word boundary. The first four bytes contain the return code and the second four bytes contain the reason code. The remainder of the information is dependent on the return and reason codes. See the coding example 'TESTER: Test ANTRQST' at the end of this appendix for how to coordinate the RETCODE, RSNCODE, and RETINFO fields.

To code: Specify the RS-type address, or address in register (2)-(12), of a 100-character field.

,ASYNCH=*asynch*
,ASYNCH=NO
An optional input parameter field containing a keyword that indicates if the request is to be asynchronous. The value is left-justified and padded on the right with blanks. If the field contains binary zeros the default will be taken.

NO indicates that the request is to be synchronous.

YES indicates that the request is to be asynchronous. The ECB specified will be posted upon completion of the request.

The default is NO.

To code: Specify the RS-type address, or address in register (2)-(12), of a 3-character field.

,ECB=*ecb*
,ECB=NO_ECB
An optional input parameter field that SDM will post for an asynchronous request. This field is ignored for synchronous requests. The default is NO_ECB.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field.

,TRELOCATE=*trelocate*
,TRELOCATE=NO
An optional input parameter field containing a keyword that indicates whether or not to perform cylinder and track relocation of the target extent's count field identifiers. The value is left-justified and padded on the right with blanks. If the field contains binary zeros the default will be taken.

NO indicates that relocation is not to take place.

YES indicates that relocation is to take place. For example, if a source extent is cylinder 03 track 05, and the corresponding target extent is cyl-

inder 08 track 06, then when the track is copied the count field will be changed on the target device to have a count value of cylinder 08 track 06.

The default is NO.

To code: Specify the RS-type address, or address in register (2)-(12), of a 3-character field.

,WAITTIME=*waittime*
,WAITTIME=0
An optional input parameter field containing how long in seconds SDM will wait for a request to complete. A value of zero says do not time the request. If the time expires before the request is complete a value of RQST_WAITTIME_EXPIRED (see ANTRQSTL) will be returned. If the request is a synchronous request the value will be placed in the RTNCODE field. If the request is an asynchronous request the value will be used as the ECB post code. The default is 0.

To code: Specify the RS-type address, or address in register (2)-(12), of a halfword field.

REQUEST=XCONTIME
Asks for the current XRC consistency time.

,SID=*sid*
A required input parameter field containing the XRC session ID to which the request is sent. The ID is left-justified and padded on the right with blanks.

To code: Specify the RS-type address, or address in register (2)-(12), of an 8-character field.

,RETINFO=*retinfo*
A required output parameter field used to return detailed information about the results of *executing* the request. Information about the results of *scheduling* the request are returned in the RETCODE and RSNCODE fields. The area should be aligned on a word boundary. The first four bytes contain the return code and the second four bytes contain the reason code. The remainder of the information is dependent on the return and reason codes. See the coding example 'TESTER: Test ANTRQST' at the end of this appendix for how to coordinate the RETCODE, RSNCODE, and RETINFO fields.

To code: Specify the RS-type address, or address in register (2)-(12), of a 100-character field.

,TIME=*time*
A required output parameter field to be used to return the consistency time for the specified XRC session. The returned time is a 64-bit time-of-day (TOD) value in UTC (universal time, coordinated) format. The consistency time returned is the time to which volume pairs in the XRC session were last consistent. The time is taken from the application system clock.

The optional ALET parameter may be used to specify the location of the TIME field.

To code: Specify the RS-type address, or address in register (2)-(12), of an 8-character field.

,ALET=*alet*
,ALET=0

An optional input parameter field that specifies the ALET to be used if the TIME field has an ALET different from the one associated with the caller. The default is 0.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field.

,ASYNCH=*asynch*
,ASYNCH=NO

An optional input parameter field containing a keyword that indicates if the request is to be asynchronous. The value is left-justified and padded on the right with blanks. If the field contains binary zeros the default will be taken.

NO indicates that the request is to be synchronous.

YES indicates that the request is to be asynchronous. The ECB specified will be posted upon completion of the request.

The default is NO.

To code: Specify the RS-type address, or address in register (2)-(12), of a 3-character field.

,ECB=*ecb*
,ECB=NO_ECB

An optional input parameter field that SDM will post for an asynchronous request. This field is ignored for synchronous requests. The default is NO_ECB.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field.

,HLQ=*hlq*
,HLQ=SYS1

An optional input parameter field containing the high level qualifier that was used on the XSTART command. The name is left justified and padded on the right with blanks. If the field contains binary zeros the default will be taken.

The HLQ name will only be used if XRC is not active. In this case the consistency time will be obtained from the XRC state data set that has the supplied high level qualifier in the data set name, and a code of RQST_XCONTIME_SAVED_CONSISTENCY_TIME (see ANTRQSTL) will be returned. The default is SYS1.

To code: Specify the RS-type address, or address in register (2)-(12), of an 8-character field.

,WAITTIME=*waittime*
,WAITTIME=0

An optional input parameter field containing how long in seconds SDM will wait for a request to complete. A value of zero says do not time the request. If the time expires before the request is complete a value of RQST_WAITTIME_EXPIRED (see ANTRQSTL) will be returned. If the request is a synchronous request the value will be placed in the RTNCODE field. If the request is an asynchronous request the value will be used as the ECB post code. The default is 0.

To code: Specify the RS-type address, or address in register (2)-(12), of a halfword field.

,RETCODE=*retcode*

An optional output parameter into which the return code is to be copied from GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,RSNCODE=*rsncode*

An optional output parameter into which the reason code is to be copied from GPR 0.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=0

An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are as follows:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.
- **0**, if you use the currently available parameters.

To code: Specify one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 0

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr,attr*)
,MF=(L,*list addr*,'X'0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)
,MF=(E,*list addr*,NOCHECK)
,MF=(M,*list addr*)
,MF=(M,*list addr*,COMPLETE)
,MF=(M,*list addr*,NOCHECK)

An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

Use MF=M together with the list and execute forms of the macro for service routines that need to provide different options according to user-provided input. Use the list form to define a storage area. Use the modify form to set the appropriate options, then use the execute form to call the service.

IBM recommends that you use the modify and execute forms of ANTRQST in the following order:

- Use ANTRQST ...MF=(M,list-addr,COMPLETE) specifying appropriate parameters, including all required ones.
- Use ANTRQST ...MF=(M,list-addr,NOCHECK), specifying the parameters that you want to change.
- Use ANTRQST ...MF=(E,list-addr,NOCHECK), to execute the macro.

,*list addr*
The name of a storage area to contain the parameters. For MF=S, MF=E, and MF=M, this can be an RS-type address or an address in register (1)-(12).

,*attr*
An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of `X'0F'` to force the parameter list to a word boundary, or `X'0D'` to force the parameter list to a doubleword boundary. If you do not code *attr*, the system provides a value of `X'0D'`.

,COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

,NOCHECK
Specifies that the system is not to check for required parameters and is not to supply defaults for omitted optional parameters.

Character Filtering

The rules for a filter pattern are as follows:

- An alphanumeric character represents itself. *Characters are case-sensitive*.
- A question mark (?) represents any character.
- An asterisk (*) represents zero or more characters. Two or more consecutive asterisks are redundant.
- Characters in square brackets ([and]) represent *one of* the enclosed characters. If the first character in brackets is a logical not (shift 6 on the keyboard), the meaning changes to *not one of* the enclosed characters.
- A back-slash (\) allows one of the above characters, or itself, to be used in a pattern.

ABEND Codes

None.

Return and Reason Codes

When the ANTRQST macro returns control to your program:

- GPR 15 (and *retcode*, when you code RETCODE) contains a return code.
- When the value in GPR 15 is not zero, GPR 0 (and *rsncode*, when you code RSNCODE) contains reason code.

There are two ways that return and reason codes can be returned to the caller:

1. In the RTNCODE and RSNCODE fields. These codes are returned by the system data mover (SDM) module ANTCDPI that runs in the caller's address space. They are associated with the validation of common parameters supplied on the ANTRQST macro (ASYNCH, for example) along with verifying the environment (for example, a PPRC request that address space ANTAS000 is active). The parameters from the ANTRQST invocation are placed in a parameter list (called the ANTRQSTL).
2. In the return code and reason code part of the RETINFO field. Once the request has been transferred to the appropriate SDM address space, there is no mechanism provided by OS/390 to get information back to the caller's RTNCODE and RSNCODE fields (actually registers 15 and 0). The codes returned in the RETINFO field have to do with additional validation of parameters, scheduling of processing that runs in an SDM address space, and the results of executing the request.

Return codes will be found in either RTNCODE or RETINFO, but not both. EXCEPTION: return code 0 can be found in both places.

Executable macro ANTRQSTL has been provided to generate the macro list that ANTRQST fills in along with definitions of the return code values, which are shown below. There are four parameters that can be coded on the ANTRQSTL macro. The description of these four parameters is found in the prolog of the macro.

Figure 127 identifies the hexadecimal return codes and reason codes. IBM support personnel may request the entire reason code, SYSLOG, and any associated dumps.

Figure 127 (Page 1 of 8). Return Codes for the ANTRQST Macro. Reason code information, where applicable, is included in the "Meaning and Action" column.

Return Code	Meaning and Action
0	**Meaning**: Function has completed successfully. **Action**: None.
7000	**Meaning**: The PC number used to invoke ANTCDPI is zero. **Action**: Make sure ANTAS000 is active by issuing "D A,ANTAS000." If it is not active, follow the appropriate procedure in *DFSMS/MVS Remote Copy Administrator's Guide and Reference* to start the address space.
7001	**Meaning**: ANTCDPI abended while referencing the ANTRQSTL. The reason code contains the ABEND code. **Action**: Make sure the address of the ANTRQSTL is not modified before ANTCDPI is invoked. If problem persists contact IBM.
7002	**Meaning**: The ANTRQSTL had an incorrect eyecatcher. **Action**: Make sure the address of the ANTRQSTL is not modified before ANTCDPI is invoked. If problem persists contact IBM.
7003	**Meaning**: ANTCDPI abended while referencing the last byte in the ANTRQSTL. The reason code contains the ABEND code. **Action**: Make sure the address of the ANTRQSTL is not modified before ANTCDPI is invoked. If problem persists contact IBM.
7004	**Meaning**: ANTCDPI abended while referencing the field pointed to by the ANTRQST ECB parameter. The reason code contains the ABEND code. **Action**: Make sure the address of the ECB field is correct.
7005	**Meaning**: ANTCDPI abended while referencing the field pointed to by the ANTRQST RETINFO parameter. The reason code contains the ABEND code. **Action**: Make sure the address of the RETINFO field is correct.
7006 to 7008	**Meaning**: These are internal errors. **Action**: Contact IBM.
7009	**Meaning**: The request is not currently supported by ANTRQSTL. **Action**: Upgrade to the ANTRQST level that supports this request. Execute REQUEST=LEVEL to find the current ANTRQST level.
7010	**Meaning**: Caller not authorized to execute request. **Action**: Make sure caller has read access to the FACILITY class profile of STGADMIN.ANT.XRC.COMMANDS for XRC requests, STGADMIN.ANT.PPRC.COMMANDS for PPRC requests, or STGADMIN.ANT.SNAPSHOT.COMMANDS for Snapshot requests.
7011	**Meaning**: This is an internal error. **Action**: Contact IBM.
7012	**Meaning**: Unable to get STOKEN for the ALET specified on the ANTRQST macro. The reason code contains the ALESERV EXTRACT return code. **Action**: Make sure a valid ALET is supplied. If problem persists contact IBM.

Figure 127 (Page 2 of 8). Return Codes for the ANTRQST Macro. Reason code information, where applicable, is included in the "Meaning and Action" column.

Return Code	Meaning and Action
7013	**Meaning**: An SDM address space is not active. **Action**: For PPRC and XRC requests, make sure ANTAS000 is started. For Snapshot requests, make sure ANTMAIN is started. Follow the appropriate procedure in *DFSMS/MVS Remote Copy Administrator's Guide and Reference* to start either address space.
7014	**Meaning**: The value supplied in the ASYNCH parameter is invalid. If the ASYNCH parameter is used, its value must be NO or YES. **Action**: Supply a correct ASYNCH value.
7015	**Meaning**: The ECB parameter was not used when ASYNCH(YES) was coded. **Action**: Supply an ECB value.
7029 to 7032	**Meaning**: These are internal errors. **Action**: Contact IBM.
7033	**Meaning**: Unable to get the STOKEN for an SDM request router address space. The reason code contains the ALESERV EXTRACT return code. **Action**: Contact IBM.
7034	**Meaning**: The ANTRQSTL passed into an SDM request router/handler no longer has a valid eyecatcher. **Action**: Make sure fields in the ANTRQSTL have not been changed while waiting for completion of the request.
7035	**Meaning**: An SDM request router/handler was no longer able to access the ANTRQSTL. **Action**: Make sure the space the ANTRQSTL resides in has not been FREEMAINed.
7036	**Meaning**: An SDM address space is not active. **Action**: For XRC requests make sure ANTAS001 is started. If it is not active, follow the appropriate procedure in *DFSMS/MVS Remote Copy Administrator's Guide and Reference* to start the address space.
7037	**Meaning**: Invalid parameter list passed to an SDM request handler. The reason code tell which field in the parameter list was bad. **Action**: Contact IBM.
7038	**Meaning**: An SDM request handler running in ANTAS001 was unable to add the STOKEN for ANTAS000 to the access list for ANTAS001. **Action**: Contact IBM.
7039	**Meaning**: The value supplied in the WAITTIME parameter has been reached. SDM has returned to the caller. **Action**: Determine if the requested action has been suspended, or if the requested action normally takes longer than the WAITTIME value. If increasing the WAITTIME value does not correct the problem contact IBM.
7040	**Meaning**: A request has been made to a Snapshot subsystem handler, and an error had been detected. **Action**: Contact IBM.
7041	**Meaning**: An SDM address space has ended. If the request is synchronous 7041 will be placed in the RETINFO field. If the request is asynchronous 7041 will be the post code. **Action**: Resubmit the request after the SDM address space has been restarted.

Figure 127 (Page 3 of 8). Return Codes for the ANTRQST Macro. Reason code information, where applicable, is included in the "Meaning and Action" column.

Return Code	Meaning and Action
7042	**Meaning**: The SDM service that moves the results of the request back into the caller's ANTRQSTL has abended. The reason code contains the ABEND code. **Action**: Make sure that the areas pointed to by the ANTRQSTL have not been freed. If the problem persists contact IBM.
7097	**Meaning**: Unexpected ABEND in SDM request router. **Action**: Contact IBM.
7098	**Meaning**: Unexpected ABEND in an SDM level 1 request handler. The reason code contains the ABEND code. **Action**: Contact IBM.
7099	**Meaning**: Unexpected ABEND in an SDM level 2 request handler. The reason code contains the ABEND code. **Action**: Contact IBM.
7100	**Meaning**: For an XCONTIME request, an SDM request router/handler abended while referencing the field pointed to by the ANTRQST TIME parameter. The reason code contains the ABEND code. **Action**: Make sure that the TIME parameter points to a valid location in memory, and that if an ALET was specified on the ANTRQST macro, its value is correct.
7101	**Meaning**: For an XCONTIME request, the XRC system ID pointed to by the ANTRQST SID parameter was invalid. **Action**: Make sure the system ID supplied by the SID parameter matches the ID of a valid XRC session.
7102	**Meaning**: For an XCONTIME request, the high-level qualifier of an XRC state data set provided by the ANTRQST HLQ parameter was invalid. **Action**: Make sure the name supplied by the HLQ parameter matches the high-level qualifier of an XRC state data set.
7103	**Meaning**: For an XCONTIME request, the consistency time being maintained by XRC is zero. **Action**: Consistency time is initialized to zero, and does not change until an application update takes place to a primary volume in an XRC session.
7104	**Meaning**: For an XCONTIME request, XRC was unable to acquire working storage. **Action**: Contact IBM.
7105	**Meaning**: For an XCONTIME request, the table that contains the skeleton name of the state data set could not be found. **Action**: Contact IBM.
7106	**Meaning**: For an XCONTIME request, XRC was unable to allocate the state data set. This can be for one of three reasons: 1. The state data set has not been pre-allocated. 2. The high-level qualifier supplied in the HLQ parameter is invalid. 3. The session ID supplied in the SID parameter is invalid. **Action**: For reason 1, pre-allocate the state data set. For reasons 2 and 3 correct the invalid value.
7107	**Meaning**: For an XCONTIME request, XRC was unable to open the state data set. **Action**: Contact IBM.
7108	**Meaning**: For an XCONTIME request, XRC was unable to create a DECB control block for the state data set. **Action**: Contact IBM.

Figure 127 (Page 4 of 8). Return Codes for the ANTRQST Macro. Reason code information, where applicable, is included in the "Meaning and Action" column.

Return Code	Meaning and Action
7109	**Meaning**: For an XCONTIME request, the member in the state data set containing the consistency time could not be found. **Action**: If XRC has never been started then an XSTART will need to be done to create the member. If an XSTART has been done contact IBM.
7110	**Meaning**: For an XCONTIME request, the data in the state data set is not in the proper format. **Action**: Contact IBM.
7111	**Meaning**: For an XCONTIME request, XRC was unable to read the state data set. **Action**: Contact IBM.
7112	**Meaning**: For an XCONTIME request, XRC was unable to load the DCB exits used in reading the state data set. **Action**: Contact IBM.
7113	**Meaning**: For an XCONTIME request ANTAS001 was not active, so the consistency time being returned is from the state data set. **Action**: Contact IBM.
7120	**Meaning**: For a PESTPATH request, an SDM request handler detected an error. Byte nine of the RETINFO field contains the length of the following error message. Starting at byte ten will be a message that describes the error. Bytes ten through thirteen contain the ending part of an ANTP0xxxx message identifier (an example is 108I for message identifier ANTP0108I). **Action**: Take the action described in the documentation for the message identifier.
7140	**Meaning**: For a PDELPAIR request, an SDM request handler detected an error. Byte nine of the RETINFO field contains the length of the following error message. Starting at byte ten will be a message that describes the error. Bytes ten through thirteen contain the ending part of an ANTP0xxxx message identifier (an example is 108I for message identifier ANTP0108I). **Action**: Take the action described in the documentation for the message identifier.
7160	**Meaning**: For a PDELPATH request, an SDM request handler detected an error. Byte nine of the RETINFO field contains the length of the following error message. Starting at byte ten will be a message that describes the error. Bytes ten through thirteen contain the ending part of an ANTP0xxxx message identifier (an example is 108I for message identifier ANTP0108I). **Action**: Take the action described in the documentation for the message identifier.
7180	**Meaning**: For a PSUSPEND request, an SDM request handler detected an error. Byte nine of the RETINFO field contains the length of the following error message. Starting at byte ten will be a message that describes the error. Bytes ten through thirteen contain the ending part of an ANTP0xxxx message identifier (an example is 108I for message identifier ANTP0108I). **Action**: Take the action described in the documentation for the message identifier.
7181	**Meaning**: For a PSUSPEND request, the value supplied in the PRIMARY parameter is invalid. If the PRIMARY parameter is used, its value must be NO or YES. **Action**: Supply a correct PRIMARY value.
7182	**Meaning**: For a PSUSPEND request, the value supplied in the QUIESCE parameter is invalid. If the QUIESCE parameter is used, its value must be NO or YES. **Action**: Supply a correct PRIMARY value.

Figure 127 (Page 5 of 8). Return Codes for the ANTRQST Macro. Reason code information, where applicable, is included in the "Meaning and Action" column.

Return Code	Meaning and Action
7200	**Meaning**: For a PESTPAIR request, an SDM request handler detected an error. Byte nine of the RETINFO field contains the length of the following error message. Starting at byte ten will be a message that describes the error. Bytes ten through thirteen contain the ending part of an ANTP0xxxx message identifier (an example is 108I for message identifier ANTP0108I). **Action**: Take the action described in the documentation for the message identifier.
7201	**Meaning**: For a PESTPAIR request, the value supplied in the CRIT parameter is invalid. If the CRIT parameter is used, its value must be NO or YES. **Action**: Supply a correct CRIT value.
7202	**Meaning**: For a PESTPAIR request, the value supplied in the MODE parameter is invalid. If the MODE parameter is used, its value must be COPY, NOCOPY, or RESYNC. **Action**: Supply a correct MODE value.
7203	**Meaning**: For a PESTPAIR request, the value supplied in the MSGREQ parameter is invalid. If the MSGREQ parameter is used, its value must be NO or YES. **Action**: Supply a correct MSGREQ value.
7220	**Meaning**: For a PRECOVER request, an SDM request handler detected an error. Byte nine of the RETINFO field contains the length of the following error message. Starting at byte ten will be a message that describes the error. Bytes ten through thirteen contain the ending part of an ANTP0xxxx message identifier (an example is 108I for message identifier ANTP0108I). **Action**: Take the action described in the documentation for the message identifier.
7240	**Meaning**: For a PQUERY request, an SDM request handler detected an error. Byte nine of the RETINFO field contains the length of the following error message. Starting at byte ten will be a message that describes the error. Bytes ten through thirteen contain the ending part of an ANTP0xxxx message identifier (an example is 108I for message identifier ANTP0108I). **Action**: Take the action described in the documentation for the message identifier.
7241	**Meaning**: A PQUERY request was made, and the size of the QRYINFO field was not large enough to hold all of the unformatted status information. As much information as possible was placed into the QRYINFO field. **Action**: The total amount of space needed is returned in the reason code.
7242	**Meaning**: For a PQUERY request, the value supplied in the PATHS parameter is invalid. If the PATHS parameter is used, its value must be NO or YES. **Action**: Supply a correct PATHS value.
7243	**Meaning**: For a PQUERY request, the value supplied in the QRYSIZE parameter is zero. **Action**: Supply a non-zero QRYSIZE value.
7244	**Meaning**: For a PQUERY request, an SDM request router/handler abended while referencing the field pointed to by the ANTRQST QRYINFO parameter. The reason code contains the ABEND code. **Action**: Make sure that the QRYINFO parameter points to a valid location in memory, and that if an ALET was specified on the ANTRQST macro, its value is correct.
7245	**Meaning**: A PQUERY request was made, and the size of the QRYINFO field was large enough to hold all of the unformatted status information. **Action**: The total amount of space used is returned in the reason code.

Figure 127 (Page 6 of 8). Return Codes for the ANTRQST Macro. Reason code information, where applicable, is included in the "Meaning and Action" column.

Return Code	Meaning and Action
7246	**Meaning**: For a PQUERY request, the value supplied in the BITMAP parameter is invalid. If the BITMAP parameter is used, its value must be YES or NO. **Action**: Supply a correct BITMAP value.
7260	**Meaning**: For a PRUN request, an SDM request handler detected an error. Byte nine of the RETINFO field contains the length of the following error message. Starting at byte ten will be a message that describes the error. Bytes ten through thirteen contain the ending part of an ANTP0xxxx message identifier (an example is 108I for message identifier ANTP0108I). **Action**: Take the action described in the documentation for the message identifier.
7280	**Meaning**: For a PFREEZE request, an SDM request handler detected an error. Byte nine of the RETINFO field contains the length of the following error message. Starting at byte ten will be a message that describes the error. Bytes ten through thirteen contain the ending part of an ANTP0xxxx message identifier (an example is 108I for message identifier ANTP0108I). **Action**: Take the action described in the documentation for the message identifier.
7300	**Meaning**: For a SDVCINFO request, a Snapshot subsystem detected an error. Bytes five through eight of the RETINFO field contain the Snapshot subsystem error code. **Action**: Take the action described in the documentation for the Snapshot subsystem.
7301	**Meaning**: For a SDVCINFO request, the number supplied in the BASICSIZE parameter is too small. The value must be at least as large as the length of PSL10_HEADER. **Action**: Provide a larger BASICINFO area.
7302	**Meaning**: For a SDVCINFO request, an SDM request router/handler abended while referencing the field pointed to by the ANTRQST BASICINFO parameter. The reason code contains the ABEND code. **Action**: Make sure that the BASICINFO parameter points to a valid location in memory, and that if an ALET was specified on the ANTRQST macro, its value is correct.
7303	**Meaning**: For a SDVCINFO request, the number supplied in the EXTNDSIZE parameter is too small. The value must be at least as large as the length of PSL11_HEADER. **Action**: Provide a larger EXTNDINFO area.
7304	**Meaning**: For a SDVCINFO request, an SDM request router/handler abended while referencing the field pointed to by the ANTRQST EXTNDINFO parameter. The reason code contains the ABEND code. **Action**: Make sure that the EXTNDINFO parameter points to a valid location in memory, and that if an ALET was specified on the ANTRQST macro, its value is correct.
7320	**Meaning**: For a SQRYDVCS request, a Snapshot subsystem detected an error. Bytes five through eight of the RETINFO field contain the Snapshot subsystem error code. **Action**: Take the action described in the documentation for the Snapshot subsystem.
7321	**Meaning**: For a SQRYDVCS request, the value supplied in the ACCESS parameter is invalid. If the ACCESS parameter is used, its value must be NO, R, or RW. **Action**: Supply a correct ACCESS value.
7322	**Meaning**: For a SQRYDVCS request, the value supplied in the BEGINCYL0 parameter is invalid. If the BEGINCYL0 parameter is used, its value must be YES or NO. **Action**: Supply a correct BEGINCYL0 value.

Figure 127 (Page 7 of 8). Return Codes for the ANTRQST Macro. Reason code information, where applicable, is included in the "Meaning and Action" column.

Return Code	Meaning and Action
7323	**Meaning**: For a SQRYDVCS request, the value supplied in the DEVTYPE parameter is invalid. If the DEVTYPE parameter is used, its value must be NO, 3380, or 3390. **Action**: Supply a correct DEVTYPE value.
7324	**Meaning**: For a SQRYDVCS request, the value supplied in the PARTITION parameter is invalid. If the PARTITION parameter is used, its value must be NO, TEST, or PROD. **Action**: Supply a correct PARTITION value.
7325	**Meaning**: For a SQRYDVCS request, the number supplied in the QRYSIZE parameter is too small. The value must be at least as large the length of PSL30_HEADER. **Action**: Provide a larger QRYINFO area.
7326	**Meaning**: For a SQRYDVCS request, the LOWDEVN parameter was specified but a value was not provided for the HIGHDEVN parameter. **Action**: Supply a HIGHDEVN value.
7327	**Meaning**: For a SQRYDVCS request, an SDM request router/handler abended while referencing the field pointed to by the ANTRQST VLFILTER parameter. The reason code contains the ABEND code. **Action**: Make sure that the VLFILTER parameter points to a valid location in memory, and that if an ALET was specified on the ANTRQST macro, its value is correct.
7328	**Meaning**: For a SQRYDVCS request, an SDM request router/handler abended while referencing the field pointed to by the ANTRQST SSFILTER parameter. The reason code contains the ABEND code. **Action**: Make sure that the SSFILTER parameter points to a valid location in memory, and that if an ALET was specified on the ANTRQST macro, its value is correct.
7329	**Meaning**: For a SQRYDVCS request, an SDM request router/handler abended while referencing the field pointed to by the ANTRQST QRYINFO parameter. The reason code contains the ABEND code. **Action**: Make sure that the QRYINFO parameter points to a valid location in memory, and that if an ALET was specified on the ANTRQST macro, its value is correct.
7340	**Meaning**: For a SQRYSSYS request, a Snapshot subsystem detected an error. Bytes five through eight of the RETINFO field contain the Snapshot subsystem error code. **Action**: Take the action described in the documentation for the Snapshot subsystem.
7341	**Meaning**: For a SQRYSSYS request, the number supplied in the QRYSIZE parameter is too small. The value must be at least as large the length of PSL20_HEADER. **Action**: Provide a larger QRYINFO area.
7342	**Meaning**: For a SQRYSSYS request, an SDM request router/handler abended while referencing the field pointed to by the ANTRQST SSFILTER parameter. The reason code contains the ABEND code. **Action**: Make sure that the SSFILTER parameter points to a valid location in memory, and that if an ALET was specified on the ANTRQST macro, its value is correct.

Figure 127 (Page 8 of 8). Return Codes for the ANTRQST Macro. Reason code information, where applicable, is included in the "Meaning and Action" column.

Return Code	Meaning and Action
7343	**Meaning**: For a SQRYSSYS request, an SDM request router/handler abended while referencing the field pointed to by the ANTRQST QRYINFO parameter. The reason code contains the ABEND code. **Action**: Make sure that the QRYINFO parameter points to a valid location in memory, and that if an ALET was specified on the ANTRQST macro, its value is correct.
7360	**Meaning**: For a SRELEASE request, a Snapshot subsystem detected an error. Bytes five through eight of the RETINFO field contain the Snapshot subsystem error code. **Action**: Take the action described in the documentation for the Snapshot subsystem.
7361	**Meaning**: For a SRELEASE request, the number supplied in the EXTNBR parameter is zero. At least one extent pair must be provided. **Action**: Provide at least one extent pair.
7362	**Meaning**: For a SRELEASE request, an SDM request router/handler abended while referencing the field pointed to by the ANTRQST EXTINFO parameter. The reason code contains the ABEND code. **Action**: Make sure that the EXTINFO parameter points to a valid location in memory, and that if an ALET was specified on the ANTRQST macro, its value is correct.
7380	**Meaning**: For a SSNAP request, a Snapshot subsystem detected an error. Bytes five through eight of the RETINFO field contain the Snapshot subsystem error code. **Action**: Take the action described in the documentation for the Snapshot subsystem.
7381	**Meaning**: For a SQRYDVCS request, the value supplied in the TRELOCATE parameter is invalid. If the TRELOCATE parameter is used, its value must be NO or YES. **Action**: Supply a correct TRELOCATE value.
7382	**Meaning**: For a SSNAP request, the number supplied in the SEXTNBR parameter is zero. At least one extent pair must be provided. **Action**: Provide at least one extent pair.
7383	**Meaning**: For a SSNAP request, the number supplied in the TEXTNBR parameter is zero. At least one extent pair must be provided. **Action**: Provide at least one extent pair.
7384	**Meaning**: For a SSNAP request, an SDM request router/handler abended while referencing the field pointed to by the ANTRQST SEXTINFO parameter. The reason code contains the ABEND code. **Action**: Make sure that the SEXTINFO parameter points to a valid location in memory, and that if an ALET was specified on the ANTRQST macro, its value is correct.
7385	**Meaning**: For a SSNAP request, an SDM request router/handler abended while referencing the field pointed to by the ANTRQST TEXTINFO parameter. The reason code contains the ABEND code. **Action**: Make sure that the TEXTINFO parameter points to a valid location in memory, and that if an ALET was specified on the ANTRQST macro, its value is correct.

Example: Getting the Current Consistency Time for XRC Session TUCSON in Assembler

```
         TITLE 'TESTER: Test ANTRQST'
TESTER   CSECT ,
         USING *,15
         B     @PROLOG
         DC    AL1(16)
         DC    C'TESTER    97.133'
         DROP  15
@PROLOG  STM   14,12,12(13)
         LR    12,15
@PSTART  EQU   TESTER
         USING @PSTART,12
         LA    15,0
         L     0,@SIZDATD+4
         GETMAIN  RU,LV=(0),SP=(15)
         LR    15,13
         LR    13,01
         USING @DATD,13
         ST    15,4(,13)
         ST    13,8(,15)
         LM    15,01,16(15)
         MVC   XRCSID,=CL8'TUCSON  '
         ANTRQST REQUEST=XCONTIME,            X
               SID=XRCSID,                    X
               TIME=XCONTIME,                 X
               RETINFO=XRETINFO,              X
               RETCODE=RTNCD,                 X
               RSNCODE=RSNCD,                 X
               MF=(E,P_LIST)
         CLC   RTNCD,=F'0'
         BE    SKIP1
         MVC   RTCD,RTNCD
         MVC   RSCD,RSNCD
         B     SKIP2
SKIP1    DS    0H
         MVC   RTCD,RTC
         MVC   RSCD,RSC
SKIP2    DS    0H
         CLC   RTCD,=A(RQST_PC_NUMBER_ZERO)
         BNE   SKIP3
         WTO   'ANTAS000 NOT ACTIVE'
SKIP3    DS    0H
         L     2,RTCD
         L     3,RSCD
         LR    1,13
         L     13,4(,13)
         LA    15,0
         L     0,@SIZDATD+4
         FREEMAIN RU,LV=(0),A=(1),SP=(15)
         LR    15,2
         LR    0,3
         L     14,12(,13)
         LM    1,12,24(13)
         BR    14
         LTORG
```

```
@SIZDATD DS    0A
         DC    AL1(0)
         DC    AL3(@DYNSIZE)
@DATD    DSECT
         DS    18F
RTNCD    DS    F
RSNCD    DS    F
RTCD     DS    F
RSCD     DS    F
XRCSID   DS    CL8
XCONTIME DS    CL8
XRETINFO DS    0F,CL100
         ORG   XRETINFO
RTC      DS    F
RSC      DS    F
         ORG   XRETINFO+100
         ANTRQSTL NAME=P_LIST,BASE=0F
         ORG   *+1-(*-@DATD)/(*-@DATD)
@ENDDATD DS    0X
@DYNSIZE EQU   ((@ENDDATD-@DATD+7)/8)*8
         END   TESTER
```

Appendix C. Control Blocks

The following control blocks are described in this appendix:

- Data Extent Block (DEB) for EXCP and EXCPVR (see *DFSMS/MVS DFSMSdfp Diagnosis Reference* for all fields in the DEB).
- Data Facilities Area (DFA).

Data Extent Block (DEB) Fields

The data extent block (DEB) fields shown here should *only* be used with EXCP and EXCPVR (see *DFSMS/MVS DFSMSdfp Diagnosis Reference* for all fields in the DEB).

Common Name: Data Extent Block
Macro ID: IEZDEB
DSECT Name: DEB (DSECT card precedes AVT section) DEBBASIC should be used for USING for basic section. DEBDASD (DSECT name for direct access section) DEBACSMD (DSECT name for access method sections) DEBSUBNM (DSECT name for subroutine name section) DEBXTN (DSECT name for DEB extension)
Size: Variable (device and access method dependent sections)
Created by: OPEN
Pointed to by: DCBDEBAD field of the DCB data area DEBDEBAD field of the DEB data area (next DEB on the chain)
Function: The DEB is an extension of the information in the DCB. Each DEB is associated with a DCB, and the two point to each other. It contains information about the physical characteristics of the data set, and other information used by the control program.

Figure 128 (Page 1 of 7). Partial Listing of DEB Fields

Offset	Type	Length	Name	Description
APPENDAGE VECTOR TABLE SECTION OF THE DEB POINTED TO BY DEBAPPAD				
36(24)	DBL WORD	8	DEBAVT(0)	APPENDAGE VECTOR TABLE
36(24)	ADDRESS	4	DEBEOEA(0)	ADDRESS OF END-OF-EXTENT APPENDAGE ROUTINE
36(24)	BITSTRING	1	DEBEOEAB	FLAG BYTE
	 xxxx		DEBEOENP	X'OF'– NUMBER OF 2K PAGES TO BE FIXED FOR THE END–OF–EXTENT APPENDAGE
35(23)	ADDRESS	3	DEBEOEAD	ADDRESS OF END-OF-EXTENT APPENDAGE ROUTINE
32(20)	ADDRESS	4	DEBSIOA(0)	ADDRESS OF START I/O APPENDAGE ROUTINE
32(20)	BITSTRING	1	DEBSIOAB	FLAG BYTE
	1.		DEBPGFX	X'80' ADDRESS IN DEBSIOAD CAN BE USED TO DETERMINE THE ENTRY POINT TO THE PAGE FIX (PGFX) APPENDAGE ROUTINE BY ADDING 4 TO THE ADDRESS IN DEBSIOAD
	.1.		DEBSIOX	X'40'– IF ZERO, DO NOT ENTER SIO APPENDAGE WHEN ERP IS ACTIVE. IF ONE, ENTER SIO APPENDAGE EVEN WHEN ERP IS ACTIVE.
	. .1.		DEBIOVR	X'20'– IF ONE, EXCPVR REQUEST IS VALID. IF ZERO, EXCPVR REQUEST IS INVALID AND WILL NOT BE EXECUTED. CURRENTLY HAS NO EFFECT.
	. . .1		DEBFIX	X'10' INDICATION THAT DEB HAS BEEN FIXED
	 xxxx		DEBSIONP	X'0F'– NUMBER OF 2K PAGES TO BE FIXED FOR THE SIO APPENDAGE

© Copyright IBM Corp. 1979, 1999

Figure 128 (Page 2 of 7). Partial Listing of DEB Fields

Offset	Type	Length	Name	Description
31(1F)	ADDRESS	3	DEBSIOAD	ADDRESS OF START I/O APPENDAGE ROUTINE
28(1C)	ADDRESS	4	DEBPCIA(0)	ADDRESS OF PCI APPENDAGE ROUTINE
28(1C)	BITSTRING	1	DEBPCIAB	FLAG BYTE
	 xxxx		DEBPCINP	X'0F'–NUMBER OF 2K PAGES TO BE FIXED FOR THE PCI APPENDAGE
27(1B)	ADDRESS	3	DEBPCIAD	ADDRESS OF PROGRAM CONTROLLED INTERRUPTION (PCI) APPENDAGE ROUTINE
24(18)	ADDRESS	4	DEBCEA(0)	ADDRESS OF CHANNEL END APPENDAGE ROUTINE
24(18)	BITSTRING	1	DEBCEAB	FLAG BYTE
	1.		DEBESMVR	X'80',C'X' VALIDITY CHECK FOR EXCPVR CALLER
	 xxxx		DEBCENP	X'0F'– NUMBER OF 2K PAGES TO BE FIXED FOR THE CHANNEL–END APPENDAGE
23(17)	ADDRESS	3	DEBCEAD	ADDRESS OF CHANNEL END APPENDAGE ROUTINE
20(14)	ADDRESS	4	DEBXCEA(0)	ADDRESS OF ABNORMAL END APPENDAGE ROUTINE
20(14)	BITSTRING	1	DEBXCEAB	FLAG BYTE
	 xxxx		DEBXCENP	X'0F'– NUMBER OF 2K PAGES TO BE FIXED FOR THE ABNORMAL–END APPENDAGE
(13)	ADDRESS	3	DEBXCEAD	ADDRESS OF ABNORMAL END APPENDAGE ROUTINE
	X'14'		DEBAVTE	"*" END OF APPENDAGE VECTOR TABLE
DEB PREFIX TABLE. Addressable as negative offset from DEB basic section. Not necessarily contiguous with DEB Appendage Vector Table.				
16(10)	ADDRESS	4	DEBPREFX(0)	DEB PREFIX TABLE
16(10)	BITSTRING	1	DEBWKARA	O/C/E WORK AREA (DIRECT ACCESS)
15(F)	BITSTRING	7	DEBDSCBA	DSCB ADDRESS (BBCCHHR) USED BY O/C/E (DIRECT ACCESS)
8(8)	ADDRESS	4	DEBXTNP(0)	POINTER TO DEB EXTENSION
4(4)	BITSTRING	1	DEBLNGTH	LENGTH OF DEB IN DOUBLE WORDS
3(3)	CHARACTER	1	DEBAMTYP	ACCESS METHOD TYPE
	X'0'		DEBAMNON	"0" ACCESS METHOD TYPE NOT KNOWN
	X'1'		DEBAMVSM	"1" VSAM ACCESS METHOD TYPE
	X'2'		DEBAMXCP	"2" EXCP ACCESS METHOD TYPE
	X'4'		DEBAMTCM	"4" TCAM ACCESS METHOD TYPE
	X'8'		DEBAMGAM	"8" GRAPHICS ACCESS METHOD TYPE
	X'10'		DEBAMTAM	"16" BTAM ACCESS METHOD TYPE
	X'20'		DEBAMBPM	"32" BPAM ACCESS METHOD TYPE
	X'20'		DEBAMSAM	"32" SEQUENTIAL ACCESS METHOD TYPE
	X'40'		DEBAMBDM	"64" DIRECT ACCESS METHOD TYPE
	X'80'		DEBAMISM	"128" ISAM ACCESS METHOD TYPE
	X'81'		DEBAMSUB	"129" SUBSYSTEM ACCESS METHOD TYPE
	X'82'		DEBAMVTM	"130" VTAM ACCESS METHOD TYPE
	X'84'		DEBAMTAP	"132" TCAM APPLICATION ACC METHOD TYPE
2(2)	UNSIGNED	2	DEBTBLOF	FOR SYSTEM USE
	X'24'		DEBPREFE	"*" END OF DEB PREFIX TABLE
DEB BASIC SECTION				
	X'24'		DEBBASIC	"*"
0(0)	ADDRESS	4	DEBTCBAD(0)	ADDRESS OF TCB FOR THIS DEB
0(0)	BITSTRING	1	DEBNMSUB	OF SUBROUTINES LOADED BY OPEN EXECUTOR ROUTINES
1(1)	ADDRESS	3	DEBTCBB	ADDRESS OF TCB FOR THIS DEB
4(4)	ADDRESS	4	DEBDEBAD(0)	ADDRESS OF THE NEXT DEB IN THE SAME TASK
4(4)	BITSTRING	1	DEBAMLNG	NUMBER OF BYTES IN THE ACCESS METHOD DEPENDENT SECTION. FOR BDAM THIS FIELD CONTAINS THE LENGTH EXPRESSED IN NUMBER OF WORDS.
5(5)	ADDRESS	3	DEBDEBB	ADDRESS OF THE NEXT DEB IN THE SAME TASK
8(8)	ADDRESS	4	DEBIRBAD(0)	IRB STORAGE ADDRESS USED FOR APPENDAGE ASYNCHRONOUS EXITS
8(8)	BITSTRING	1	DEBOFLGS	DATA SET STATUS FLAGS
	xx.		DEBDISP	X'C0'– DATA SET DISPOSITION FLAGS BIT SETTING DISPOSITION
	01.		DEBDSOLD	X'40' OLD DATA SET

Figure 128 (Page 3 of 7). Partial Listing of DEB Fields

Offset	Type	Length	Name	Description
	10		DEBDSMOD	X'80' MOD DATA SET
	11.		DEBDSNEW	X'C0' NEW DATA SET
	. .1.		DEBEOF	X'20'– END OF FILE (EOF) ENCOUNTERED (TAPE INPUT) FORMAT 1 DSCB BIT 93.0 INDICATES THAT THE CURRENT VOLUME IS THE LAST VOLUME OF THE DATA SET (DASD INPUT)
	. . .1		DEBRLSE	X'10'– RELEASE UNUSED EXTERNAL STORAGE (DASD) EMULATOR TAPE WITH SECOND GENERATION FORMAT. TAPE MIGHT CONTAIN BLOCKS SHORTER THAN 12 CHARACTERS. (TAPE)
	. . . . .1. .		DEBSPLIT	X'04'–7–TRACK EMULATOR TAPE WITH POSSIBLE MIXED PARITY RECORDS (TAPE)
	. . . . . .1.		DEBLABEL	X'02'– NONSTANDARD LABELS
	1		DEBRERR	X'01'– USE REDUCED ERROR RECOVERY PROCEDURE (TAPE) CONCATENATED PARTITIONED ORGANIZATION DATA SETS PROCESSED USING BPAM (DASD)
9(9)	ADDRESS	3	DEBIRBB	IRB STORAGE ADDRESS USED FOR APPENDAGE ASYNCHRONOUS EXITS
12(C)	BITSTRING	1	DEBOPATB	FLAGS INDICATING BOTH THE METHOD OF I/O PROCESSING AND THE DISPOSITION THAT IS TO BE PERFORMED WHEN AN END OF VOLUME (EOV) CONDITION OCCURS
	1.		DEBABEND	X'80'– SET BY ABEND INDICATING A SYSABEND OR SYSUDUMP DATA SET
	.1.		DEBZERO	X'40'– ALWAYS ZERO
	. .xx		DEBPOSIT	X'30'– DATA SET POSITIONING FLAGS BIT SETTING POSITIONING
	. . 01		DEBRERED	X'10' REREAD
	. .11		DEBLEAVE	X'30' LEAVE
	 xxxx		DEBACCS	X'0F'– TYPE OF I/O ACCESSING BEING DONE BIT SETTING ACCESSING
	 0000		DEBINPUT	X'0' INPUT
	 1111		DEBOUTPT	X'F' OUTPUT or EXTEND
	 0011		DEBINOUT	X'3' INOUT
	 0111		DEBOUTIN	X'7' OUTIN or OUTINX
	 0001		DEBRDBCK	X'1' RDBACK
	 0100		DEBUPDAT	X'4' UPDAT
13(D)	BITSTRING	1	DEBQSCNT	PURGE (SVC 16) – QUIESCE COUNT. NUMBER OF DEVICES EXECUTING USER'S CHANNEL PROGRAMS, AS SHOWN BY BITS 5 AND 6 OF UCBFL1 FIELDS.
14(E)	BITSTRING	1	DEBFLGS1	FLAG FIELD
	1.		DEBPWCKD	X'80'– PASSWORD WAS SUPPLIED DURING OPEN. EOV WILL NOT REQUEST A PASSWORD FOR EACH ADDITIONAL VOLUME OF A MULTIVOLUME DATA SET.
	.1.		DEBEOFDF	X'40'– SET BY EOV TO INFORM CLOSE THAT AN END OF FILE HAS BEEN ENCOUNTERED AND, THEREFORE, DEFERRED USER LABEL PROCESSING IS ALLOWED.
	. . .1		DEBEXCPA	X'10'– EXCP(VR) IS AUTHORIZED FOR THIS DEB
	. . . . .1. .		DEBF1CEV	X'04'– EOV PROCESSING OCCURRED DURING CLOSE PROCESSING. TESTED AND SET TO ZERO BY CLOSE, SET TO ONE BY EOV.
	. . . . . .1.		DEBAPFIN	X'02'– IF ON, AUTHORIZED PROGRAMS CAN BE LOADED
	1		DEBXTNIN	X'01'– IF ONE, DEB EXTENSION EXISTS
15(F)	BITSTRING	1	DEBFLGS2	FLAG FIELD TWO THE FOLLOWING TWO FLAG BITS ARE USED BY O/C/EOV TO MAINTAIN A UNIFORM RECORDING MODE (COMPACTION OR NON-COMPACTION) ON TAPE DATA SETS THAT SPAN OVER MORE THAN ONE VOLUME.
	. . . . 1. . .		DEBDSCMP	X'08' TAPE DATA SET COMPACTION MODE
	. . . . .1. .		DEBDSNCP	X'04' TAPE DATA SET NON-COMPACTION MODE
16(10)	ADDRESS	4	DEBUSRPG(0)	ADDRESS OF PURGED I/O RESTORE LIST (PIRL)

Figure 128 (Page 4 of 7). Partial Listing of DEB Fields

Offset	Type	Length	Name	Description
16(10)	BITSTRING	1	DEBNMEXT	NUMBER OF EXTENT DESCRIPTIONS STARTING AT DEBBASND. ONE EXTENT PER UNIT FOR EXTENDED FORMAT OR PDSE DATA SETS.
17(11)	ADDRESS	3	DEBUSRPB	ADDRESS OF PURGED I/O RESTORE LIST (PIRL)
20(14)	ADDRESS	4	DEBRRQ(0)	POINTER TO RELATED REQUEST QUEUE
20(14)	BITSTRING	1	DEBPRIOR	PRIORITY OF THE TASK OWNING DEB
24(18)	ADDRESS	4	DEBDCBAD(0)	ADDRESS OF DCB OR ACB ASSOCIATED WITH THIS DEB
24(18)	BITSTRING	1	DEBPROTG(0)	TASK PROTECTION KEY IN HIGH ORDER 4 BITS
24(18)	BITSTRING	1	DEBDEBID	A HEX F IN LOW–ORDER 4 BITS TO IDENTIFY THIS BLOCK AS A DEB
25(19)	ADDRESS	3	DEBDCBB	ADDRESS OF DCB OR ACB ASSOCIATED WITH THIS DEB
28(1C)	ADDRESS	4	DEBAPPAD(0)	ADDRESS OF THE I/O APPENDAGE VECTOR TABLE
28(1C)	ADDRESS	1	DEBEXSCL	THIS FIELD IS USED TO DETERMINE THE SIZE OF THE DEVICE DEPENDENT SECTION. TWO TO THIS POWER GIVES THE LENGTH OF THE DEVICE DEPENDENT SECTION AT DEBBASND. EXTENT SCALE – 4 (16 BYTES) FOR DIRECT ACCESS DEVICE AND 3525 CARD PUNCH WITH DEVICE–ASSOCIATED DATA SET SUPPORT AND 2 (4 BYTES) FOR NONDIRECT ACCESS DEVICE AND COMMUNICATION DEVICE.
29(1D)	ADDRESS	3	DEBAPPB	ADDRESS OF THE I/O APPENDAGE VECTOR TABLE
32(20)			DEBBASND	END OF BASIC SECTION
UNIT RECORD, MAGNETIC TAPE, TELECOMMUNICATIONS DEVICES SECTION				
32(20)			DEBDDS1	"*"
32(20)	ADDRESS	4	DEBSUCBA(0)	ADDRESS OF A UCB ASSOCIATED WITH A GIVEN DATA SET ALWAYS BELOW THE 16MB LINE. IF THE ACTUAL UCB IS ABOVE THE 16MB LINE, ALLOCATION NORMALLY CAPTURES THE UCB TO CREATE A 24-BIT ADDRESS. WHEN USING EXCP, YOU SPECIFY THE NOCAPTURE OPTION OF DYNAMIC ALLOCATION, THEN OPEN OR EOV CAPTURES THE UCB UNTIL A LATER EOV OR CLOSE.
32(20)	BITSTRING	1	DEBSDVM	DEVICE MODIFIER. FOR MAGNETIC TAPE, MODESET OPERATION CODE OR MODE SET FUNCTION BYTE. FOR UNIT RECORD, RESERVED.
	1101 0011		DEBMTDN4	X'D3' 9–TRACK MODESET CCW CODE DENSITY=6250BPI
	1100 0011		DEBMTDN3	X'C3' 9–TRACK MODESET CCW CODE DENSITY=1600BPI
	1100 1011		DEBMTDN2	X'CB' 9–TRACK MODESET CCW CODE DENSITY= 800BPI
				7–TRACK TAPE MODESET SKELETON CODES (MUST BE COMPLETED WITH PARITY, TRANSLATION AND/OR CONVERSION)
	0000 0011		DEBM7DN0	X'03' 7–TRACK MODESET SKELETON DENSITY=200BPI
	0100 0011		DEBM7DN1	X'43' 7–TRACK MODESET SKELETON DENSITY=556BPI
	1000 0011		DEBM7DN2	X'83' 7–TRACK MODESET SKELETON DENSITY=800BPI
				3480 TAPE OPERATION CODE
	1100 0011		DEBMSTWI	X'C3' 3480 SET TAPE WRITE IMMEDIATE CCW CODE
				TAPE MODE SET FUNCTION BYTE
	1.		DEBMTRF0	X'80' TAPE RECORDING FORMAT BIT 0
	.1.		DEBMTRF1	X'40' TAPE RECORDING FORMAT BIT 1
	. .1.		DEBMTWI	X'20' TAPE WRITE IMMEDIATE (NON-BUFFERED WRITE)
	. . .1		DEBMINHS	X'10' INHIBIT SUPERVISOR COMMANDS
	. . . . 1. . .		DEBMCOMP	X'08' COMPACTED RECORDING MODE
	. . . . 1. . .		DEBCMPAC	"DEBMCOMP" COMPACTED RECORDING MODE
	. . . . . .1.		DEBM3424	X'02' 3424 MODE SET FLAG
	1		DEBMINHE	X'01' INHIBIT CONTROL UNIT ERP
	X'C2'		DEBM6250	"DEBMTRF0+ DEBMTRF1+ DEBM3424"SET 3424 DENSITY=6250BPI
	X'42'		DEBM1600	"DEBMTRF1+ DEBM3424" SET 3424 DENSITY=1600BPI
33(21)	ADDRESS	3	DEBSUCBB	ADDRESS OF A UCB ASSOCIATED WITH A GIVEN DATA SET
36(24)	ADDRESS	4	DEBDEVED(0)	END OF COMMON UNIT RECORD FIELDS
THE FOLLOWING FIELDS ARE PRESENT ONLY FOR THE 3525 WITH DEVICE–ASSOCIATED DATA SET SUPPORT				

Figure 128 (Page 5 of 7). Partial Listing of DEB Fields

Offset	Type	Length	Name	Description
36(24)	ADDRESS	4	DEBRDCB(0)	ADDRESS OF DCB FOR THE READ ASSOCIATED DATA SET
36(24)	BITSTRING	1	DEBRSV06	FOR SYSTEM USE
37(25)	ADDRESS	3	DEBRDCBA	ADDRESS OF DCB FOR THE READ ASSOCIATED DATA SET
40(28)	ADDRESS	4	DEBPDCB(0)	ADDRESS OF DCB FOR THE PUNCH ASSOCIATED DATA SET
40(28)	BITSTRING	1	DEBRSV07	FOR SYSTEM USE
41(29)	ADDRESS	3	DEBPDCBA	ADDRESS OF DCB FOR THE PUNCH ASSOCIATED DATA SET
44(2C)	ADDRESS	4	DEBWDCB(0)	ADDRESS OF DCB FOR THE PRINT ASSOCIATED DATA SET
44(2C)	BITSTRING	1	DEBRSV08	FOR SYSTEM USE
45(2D)	ADDRESS	3	DEBWDCBA	ADDRESS OF DCB FOR THE PRINT ASSOCIATED DATA SET
			DEBASDSE	"*" END OF DEVICE–ASSOCIATED DS SEC.
DIRECT–ACCESS STORAGE DEVICE SECTION IT FOLLOWS THE BASIC SECTION. THERE IS ONE OF THESE SECTIONS FOR EACH EXTENT, EXCEPT FOR A PDSE OR EXTENDED FORMAT DATA SET.				
0(0)	ADDRESS	4	DEBUCBAD(0)	ADDRESS OF UCB ASSOCIATED WITH THIS DATA EXTENT ALWAYS BELOW THE 16MB LINE. IF THE ACTUAL UCB IS ABOVE THE 16MB LINE, ALLOCATION NORMALLY CAPTURES THE UCB TO CREATE A 24-BIT ADDRESS. WHEN USING EXCP, IF YOU SPECIFY THE NOCAPTURE OPTION OF DYNAMIC ALLOCATION, THEN OPEN OR EOV CAPTURES THE UCB UNTIL A LATER EOV OR CLOSE.
0(0)	BITSTRING	1	DEBDVMOD	DEVICE MODIFIER – FILE MASK
1(1)	ADDRESS	3	DEBUCBA	ADDRESS OF UCB ASSOCIATED WITH THIS DATA EXTENT
4(4)	CHARACTER	2	DEBBINUM	BIN NUMBER(ZEROES)
6(6)	CHARACTER	2	DEBSTRCC	CYLINDER ADDRESS FOR THE START OF AN EXTENT. FOR A PDSE OR EXTENDED FORMAT DATA SET, BIT ZERO OF THE FIRST BYTE IS ON (X'8xxx').
8(8)	CHARACTER	2	DEBSTRHH	TRACK ADDRESS FOR THE START OF AN EXTENT.
10(A)	CHARACTER	2	DEBENDCC	CYLINDER ADDRESS FOR THE END OF AN EXTENT. FOR A PDSE THIS FIELD IS RESERVED. FOR AN EXTENDED FORMAT DATA SET THIS FIELD CONTAINS THE CYLINDER NUMBER OF THE FORMAT-1 DSCB ADDRESS.
12(C)	CHARACTER	2	DEBENDHH	READ/WRITE TRACK ADDRESS FOR THE END OF AN EXTENT. FOR A PDSE THIS FIELD IS RESERVED. FOR AN EXTENDED FORMAT DATA SET THIS FIELD CONTAINS THE TRACK NUMBER OF THE FORMAT-1 DSCB ADDRESS.
14(E)	CHARACTER	2	DEBNMTRK	NUMBER OF TRACKS ALLOCATED TO A GIVEN EXTENT. FOR A PDSE THIS FIELD IS SET TO ONE(X'0001'). FOR AN EXTENDED SEQUENTIAL DATA SET THE FIRST BYTE CONTAINS THE RECORD NUMBER OF THE FORMAT-1 DSCB ADDRESS, AND THE SECOND BYTE IS ZERO.
			DEBDASDE	"*" END OF DASD DEVICE SECTION
EXCP ACCESS METHOD, BSAM AND QSAM DEPENDENT SECTION				
0(0)	CHARACTER	2	DEBVOLSQ(0)	VOLUME SEQUENCE NUMBER FOR MULTIVOLUME SEQUENTIAL DATA SETS
0(0)	BITSTRING	1	DEBVOLBT	FIRST BYTE OF DEBVOLSQ. RESERVED FOR SYSTEM USE.

Figure 128 (Page 6 of 7). Partial Listing of DEB Fields

Offset	Type	Length	Name	Description
1(1)	SIGNED	1	DEBVLSEQ	FOR DIRECT ACCESS, SEQUENCE NUMBER OF THE VOLUME OF THE DATA SET RELATIVE TO THE FIRST VOLUME OF THE DATA SET. FOR TAPE, SEQUENCE NUMBER OF THE VOLUME OF THE DATA SET RELATIVE TO THE FIRST VOLUME PROCESSED.
2(2)	CHARACTER	2	DEBVOLNM	TOTAL NUMBER OF VOLUMES IN A MULTIVOLUME SEQUENTIAL DATA SET.
4(4)	CHARACTER	8	DEBDSNM(0)	MEMBER NAME. THIS FIELD APPEARS ONLY WHEN AN OUTPUT DATA SET HAS BEEN OPENED FOR A MEMBER NAME AND THE DSCB SPECIFIES A PARTITIONED DATA SET.
4(4)	ADDRESS	4	DEBUTSAA(0)	ADDRESS OF THE USER TOTALING SAVE AREA
4(4)	BITSTRING	1	DEBRSV13	FOR SYSTEM USE
5(5)	ADDRESS	3	DEBUTSAB	ADDRESS OF THE USER TOTALING SAVE AREA
8(8)	BITSTRING	4	DEBRSV14	FOR SYSTEM USE (IF USER TOTALING WAS SPECIFIED)
12(C)	SIGNED	2	DEBBLKSI	MAXIMUM BLOCK SIZE
14(E)	SIGNED	2	DEBLRECL	LOGICAL RECORD LENGTH
BPAM DEPENDENT SECTION				
0(0)	CHARACTER	1	DEBEXTNM(0)	FOR A PARTITIONED DATA SET OPENED FOR INPUT, EACH ONE–BYTE FIELD CONTAINS THE EXTENT NUMBER OF THE FIRST EXTENT ENTRY FOR EACH DATA SET EXCEPT THE FIRST, IF TWO OR MORE DATA SETS ARE CONCATENATED. THE NUMBER OF BYTES IN THE FIELD IS EQUAL TO ONE LESS THAN THE NUMBER OF DATA SETS CONCATENATED.
0(0)	CHARACTER	8	DEBDSNAM	FOR A PARTITIONED DATA SET OPENED FOR OUTPUT FOR A MEMBER NAME, THIS FIELD IS THE MEMBER NAME.
BDAM DEPENDENT SECTION FOR FIXED LENGTH RECORDS WITH THE OPTION OF RELATIVE BLOCK ADDRESSING				
0(0)	SIGNED	4	DEBDBLK(0)	ONE FOUR BYTE FIELD FOR EACH EXTENT DESCRIBED IN THE DEVICE DEPENDENT SECTION
0(0)	ADDRESS	1	DEBDBPT	NUMBER OF BLOCKS PER TRACK
1(1)	CHARACTER	3	DEBDBPE	NUMBER OF BLOCKS PER EXTENT
SUBROUTINE NAME SECTION NOTE FOLLOWS THE ACCESS METHOD DEPENDENT SECTION, OR THE DEVICE DEPENDENT SECTION IF THERE IS NO ACCESS METHOD SECTION				
0(0)	CHARACTER	2	DEBSUBID	SUBROUTINE IDENTIFICATION. EACH ACCESS METHOD SUBROUTINE, APPENDAGE SUBROUTINE, AND IRB ROUTINE WILL HAVE A UNIQUE EIGHT–BYTE NAME. THE LOW–ORDER TWO BYTES OF EACH ROUTINE NAME WILL BE IN THIS FIELD IF THE SUBROUTINE IS LOADED BY THE OPEN ROUTINES.
DEB EXTENSION POINTED TO BY DEBXTNP IN THE DEB PREFIX				
0(0)	SIGNED	2	DEBXLNGH	LENGTH OF DEB EXTENSION
2(2)	BITSTRING	1	DEBXFLG1	FLAG BYTE—FOR SYSTEM USE
3(3)	BITSTRING	1	DEBXFLG2	FLAG BYTE
	.1.		DEBBYP	X'40'– WHEN ON EXCP SCAN ROUTINE WILL SET IOSBYP ON AND BYPASS BUILDING A PREFIX
	. .1.		DEBCHCMP	X'20'– WHEN ON EXCP SCAN ROUTINE WILL SET IOSCHCMP ON AND BYPASS BUILDING A PREFIX
4(4)	ADDRESS	4	DEBXDSAB	POINTER TO DSAB
8(8)	BITSTRING	4		FOR SYSTEM USE
12(C)	ADDRESS	4	DEBXDBPR	POINTER TO DEB
16(10)	CHARACTER	24		FOR SYSTEM USE

Figure 128 (Page 7 of 7). Partial Listing of DEB Fields

Offset	Type	Length	Name	Description
40(28)	BITSTRING	8	DEBXDEF(0)	DEFINE EXTENT DATA AREA
40(28)	BITSTRING	1	DEBDEFG1	FLAG BYTE
	1.		DEBNSHED	X'80' NO SEEK HEAD PERMITTED
	.1.		DEBXVDEF	X'40' DEB DEF EXT DATA PARMS VALID. MUST BE ON FOR DX DATA TO BE USED.
41(29)	BITSTRING	1	DEBGATTR	GLOBAL ATTRIBUTES
	xx.		DEBECKD	X'C0' EXTENT DEFINITION 2 B'00.....' DEFINE EXTENT OPERATES AS DEFINED FOR FIXED BLOCK ARCH B'11.....' DEFINE EXTENT OPERATES AS DEFINED FOR CKD EXT-CCHH EXTENTS
	1.		DEBGAEX1	X'80' EXTENT DEFINITION 1
	.1.		DEBGAEX2	X'40' EXTENT DEFINITION 2
	. .1.		DEBSTRTP	X'20' DATA STORAGE TYPE-NORM/TEMP B'..1.....' NORMAL DATA STORAGE B'..0.....' TEMPORARY DATA STORAGE
	. . .x xx . .		DEBGA345	X'1C' ATTRIBUTES BITS 3,4,5 B'...000..' NORMAL CACHE ACCESS B'...001..' BYPASS CACHE LOAD B'...010..' INHIBIT CACHE LOAD B'...011..' SEQUENTIAL ACCESS B'...100..' RESERVED B'...101..' RESERVED B'...111..' RESERVED B'.....XX' RESERVED
	. . .1		DEBGA1	X'10' ATTRIBUTE 1
	. . . . 1 . . .		DEBGA2	X'08' ATTRIBUTE 2
	. . . . .1 . .		DEBGA3	X'04' ATTRIBUTE 3 B'...000..' NORMAL CACHE ACCESS B'...001..' BYPASS CACHE LOAD B'...010..' INHIBIT CACHE LOAD B'...011..' SEQUENTIAL ACCESS B'...100..' SEQUENTIAL STAGING MODE'...101..' RECORD ACCESS MODE B'...111..' RESERVED B'.....XX' RESERVED
	. . . 0 00 . .		DEBNCACH	X'00' NORMAL CACHE ACCESS
	. . . 0 01 . .		DEBBCACH	X'04' BYPASS CACHE LOAD
	. . . 0 10 . .		DEBICACH	X'08' INHIBIT CACHE LOAD
	. . . 0 11 . .		DEBSCACH	X'0C' SEQUENTIAL ACCESS
	. . . . . .1.		DEBNRD	X'02' NON-RETENTIVE DATA ACCESS
	1		DEBINHFW	X'01' INHIBIT FAST WRITE B'.....10' USE NON-RETENTIVE DATA B'.....11' INHIBIT DASD FAST WRITE
42(2A)	BITSTRING	2	DEBBLKSZ	BLOCKSIZE IN BYTES
44(2C)	CHARACTER	4	DEBEXTOK	ZERO,EXTENT LOCATOR OR TOKEN
44(2C)	CHARACTER	2	DEBNRDID	SUBSYSTEM FUNCTION ID: NON-RETENTIVE DATA ID OR CONCURRENT COPY / XRC SESSION ID
46(2E)	CHARACTER	2		
48(30)	CHARACTER	4	DEBIOPID	FOR SYSTEM USE
52(34)	CHARACTER	4	DEBBLKID	BLOCK ID VALUE USED TO CALCULATE NUMBER OF BLOCKS IN THIS VOLUME FOR EXTENDED BLOCK COUNT CHECKING

Data Facilities Area (DFA) Fields

Common Name: Data Facilities Area
Macro ID: IHADFA
DSECT Name: DFA
Eye-Catcher ID: DFAACRON
Subpool and Key: Nucleus resident and Key 0; Resident below 16MB
Size: 48 bytes
Created by: Assembled into nucleus
Pointed to by: CVTDFA field of the CVT
Serialization: N/A

Function: Maps the Data Facilities Area, which contains information that applies to DFSMS/MVS.

Figure 129 (Page 1 of 2). DFA Fields

Offset	Type	Length	Name	Description
0(0)	ADDRESS	2	DFALEN	LENGTH OF THIS TABLE
VERSION, RELEASE, AND MODIFICATION LEVEL INFORMATION FOR DFP. THE FIRST THREE DIGITS OF DFAREL REPRESENTS THE LAST LEVEL OF DFP(X'311'). THE FOURTH DIGIT OF DFAREL BEING NON-ZERO INDICATES THE LEVEL OF THIS PRODUCT IS HIGHER THAN THE INDICATED LEVEL.				
2(2)	BITSTRING	2	DFAREL	FOUR DIGITS = VERSION, RELEASE, MOD, X
THE FOLLOWING ARE THE FEATURE BYTES. WHEN A BIT IS ON, IT MEANS EITHER THAT THE CURRENT RELEASE SUPPORTS THE FEATURE OR THAT THIS PARTICULAR INSTANCE OF THE SYSTEM SUPPORTS THE FEATURE. IF THE SOFTWARE SUPPORTS A FEATURE BUT IT CANNOT BE USED, PERHAPS BECAUSE COREQUISITE SOFTWARE OR HARDWARE IS NOT AVAILABLE, THEN THE FEATURE BIT WILL BE OFF.				
4(4)	SIGNED	4	DFAFEATS(0)	ALL FEATURES BYTES–USED BY CS INSTR
4(4)	BITSTRING	1	DFAFEAT1	FEATURES BYTE 1
	1.		DFAXA	"X'80'" MVS/XA (COPY OF CVTMVSE IN CVT)
	.1.		DFALSR	"X'40'" MULTIPLE VSAM LSR POOLS SUPPORTED (1.1.0)
	. .1.		DFAEOS	"X'20'" DASDM ERASE ON SCRATCH SUPPORTED (2.1.0)
	. . .1		DFAXRF	"X'10'" EXTENDED RECOVERY FACILITY (2.1.0)
	. . . . 1. . .		DFAEXPCI	"X'08'" EXPORT BY CONTROL INTERVAL (2.1.0)
	. . . . .1. .		DFAEOSIC	"X'04'" ERASE ON SCRATCH FOR ICF (2.1.0)
	. . . . . .1.		DFASMS	"X'02'" SYSTEM MANAGED STORAGE (3.1.0)
	1		DFAPDSE	"X'01'" PDSE SUPPORT AVAILABLE ON THE SYSTEM SET BY IEAVNP26 WHEN DFP LEVEL IS 3.2.0 OR GREATER AND AN APPROPRIATE LEVEL OF SP EXISTS.
	1		DFAIPDS	"DFAPDSE" IPDS IS OLD NAME FOR PDSE
5(5)	BITSTRING	1	DFAFEAT2	FEATURES BYTE 2
	1.		DFADLS	"X'80'" RESERVED
	.1.		DFAPML	"X'40'" RESERVED
	. .1.		DFAFMS	"X'20'" FILE MANAGEMENT SERVICES SUPPORTED
	. . .1		DFACMPAC	"X'10'" INSTALLATION DEFAULT FOR COMPACTION
	. . . . 1. . .		DFABPBLD	"X'08'" BYPASS CHANNEL PROGRAM PREFIX BUILD
	. . . . .1. .		DFASSF	"X'04'" SSF SERVICES ARE AVAILABLE
	. . . . . .1.		DFAMMEXT	"X'02'" MMS SUPPORTS XTIOT
	1		DFAINDEF	"X'01'" COMPACTION DEFAULT EXPLICITLY SET BY INSTALLATION
6(6)	BITSTRING	1	DFAFEAT3	FEATURES BYTE 3
	1.		DFAVOLSN	"X'80'" VOLSER EXTRACTED FROM SENSE INFO ACCEPTABLE BY THE INSTALLATION
	.1		DFASAMEX	EXTENDED FORMAT SEQUENTIAL DATA SETS SUPPORTED
	. 1.		DFASMSEX	ALIAS FOR DFASAMEX
	. .1		DFAKSDEX	EXTENDED FORMAT KSDS SUPPORTED
	. . . 1		DFACMPCT	DFSMS/MVS ACCESS METHOD COMPRESSION SUPPORTED. BIT SET BY SMS SUB-SYSTEM INITIALIZATION.
	. . . . 1 . . .		DFARLSJ3	THE SMSVSAM SERVER HAS SUCCESSFULLY INITIALIZED ON THIS SYSTEM. THIS BIT IS USED BY SMS SCHEDULING. ONCE ON, THIS BIT REMAINS ON FOR THE LIFE OF THE IPL. THIS BIT DOES NOT INDICATE THAT THE SMSVSAM SERVER IS CURRENTLY OPERATIONAL.
	. . . . .1 . .		DFARECAL	DATA SET RECALL CAPABILITY VIA THE ARCHRCAL MACRO IS AVAILABLE.
	. . . . . .1 .		DFADEEXT	DESERV EXIT FUNCTION IS AVAILABLE
7(7)	BITSTRING	1	DFAFEAT4	FEATURES BYTE 4
	1.		DFAFDAT	RESERVED
	.1		DFANSRV	DFP NIP SERVICES CAN BE INVOKED VIA IGGSSRV MACRO
	. . 1.		DFADYNL	DYNAMIC LINKLIST IS SUPPORTED
	. . 1.		DFACIR2	THE CATALOG INFORMATION ROUTINE, IKJEHCIR, SUPPORTS A FORMAT 2 WORK AREA, I.E., FULL WORD LENGTH FIELDS
	. . .1		DFARTLS	DFSMS/MVS SUPPORT FOR RTLS IS AVAILABLE

Figure 129 (Page 2 of 2). DFA Fields

Offset	Type	Length	Name	Description
	. . . . 1. . .		DFADYLPA	DFSMS/MVS SUPPORT FOR DYNAMIC LPA IS AVAILABLE
8(8)	CHARACTER	4	DFAACRON	ACRONYM FOR THIS CONTROL BLOCK(“DFA*bb*”) (*bb* here represents a space.)
12(C)	BITSTRING	1	DFAFEAT5	FEATURES BYTE 5
	. . . . . .1 .		DFAALVER	DETERMINES INSTALLATION VERSION ISO/ANSI LEVEL: VERSION 3 OR VERSION 4 TAPE LABELS
	 1		DFAALFOR	DETERMINES WHETHER INSTALLATION ISO/ANSI VERSION LEVEL IS FORCED
13(D)	BITSTRING	1	DFAFEAT6	FEATURES BYTE 6
14(E)	BITSTRING	1	DFALPKG	LICENSED PACKAGE BYTE: IF ONE OF THE FOLLOWING BITS IS ON, THIS INSTALLATION HAS INDICATED THAT IT HAS A LICENSE FOR THAT IBM PACKAGE
	1.		DFAHSM	DFSMShsm IS ALLOWED TO RUN
	. 1.		DFADSS	DFSMSdss IS ALLOWED TO RUN
	. . 1.		DFARMM	DFSMSrmm IS ALLOWED TO RUN
	. . .1		DFAOPT	OPTIMIZER IS ALLOWED TO RUN
	. . . . 1. . .		DFADFP0	RESERVED
	. . . . .1 . .		DFAADSM	ADSTAR DISTRIBUTED STORAGE MANAGER IS ALLOWED TO RUN
	. . . . . .1 .		DFASORT	DFSORT IS ALLOWED TO RUN
15(F)	BITSTRING	1	DFALPKG2	SECOND LICENSED PACKAGE BYTE
	1.		DFANFS	OS/390 NFS IS ALLOWED TO RUN
PRODUCT, VERSION, RELEASE, AND MODIFICATION LEVEL INFORMATION. BYTE 0 DEFINES A PRODUCT CODE, BYTES 1-3 DEFINE THE VERSION, RELEASE, AND MODIFICATION LEVELS OF THIS PRODUCT. FOR DFSMS/MVS THE PRODUCT BYTE IS X'01', SUBSEQUENT PRODUCTS WILL INCREMENT THIS PRODUCT NUMBER SO THAT RELEASE IDENTIFIERS CAN BE COMPARED NUMERICALLY INDEPENDENT OF PRODUCT NAMES. THE DFP PRODUCT IS INDICATED BY A PRODUCT BYTE OF X'00'.				
16(10)	BITSTRING	4	DFARELS	4 BYTES = PRODUCT, VERSION, REL, MOD
16(10)	BITSTRING	1	DFAPROD	PRODUCT BYTE
	X'00'		DFADFP	"X'00'" PRODUCT CODE FOR DFP
	X'01'		DFADFSMS	"X'01'" PRODUCT CODE FOR DFSMS/MVS
17(11)	BITSTRING	1	DFAVER	VERSION BYTE
18(12)	BITSTRING	1	DFARLSE	RELEASE BYTE
19(13)	BITSTRING	1	DFAMOD	MODIFICATION BYTE
20(14)	SIGNED	2	DFAMSMDE	MAXIMUM LENGTH OF THE SMDE IN THIS RELEASE WITH AN 8-BYTE ALIAS NAME
22(16)	BITSTRING	2		RESERVED
24(18)	ADDRESS	4	DFACSSVT	CALLABLE SYSTEM SERVICES VECTOR TABLE ADDRESS
28(1C)	BITSTRING	16		RESERVED
44(2C)	ADDRESS	4	DFADFVAD	FOR SYSTEM USE
END OF DFA AS IT WAS WHEN IT WAS FIRST SHIPPED IN MVS/XA DFP VERSION 2 RELEASE 1 MODIFICATION LEVEL 0. FUTURE ADDITIONS WILL BE INSERTED IMMEDIATELY BEFORE THIS LINE.				
	X'30'		DFAEND	"*" ALWAYS END OF DFA

Glossary

The following terms are defined as they are used in the DFSMS/MVS Library. If you do not find the term you are looking for, see the IBM Software Glossary:

`http://www.networking.ibm.com/nsg/nsg`

This glossary is an ever-evolving document that defines technical terms used in the documentation for many IBM software products.

A

access method control block (ACB). A control block that links an application program to VSAM or VTAM programs.

access method services. A multifunction service program that manages VSAM and non-VSAM data sets, as well as integrated catalog facility (ICF) and VSAM catalogs. Access method services provides the following functions:

- defines and allocates space for VSAM data sets, VSAM catalogs, and ICF catalogs
- converts indexed-sequential data sets to key-sequenced data sets
- modifies data set attributes in the catalog
- reorganizes data sets
- facilitates data portability among operating systems
- creates backup copies of data sets
- assists in making inaccessible data sets accessible
- lists the records of data sets and catalogs
- defines and builds alternate indexes
- converts CVOLS and VSAM catalogs to ICF catalogs

B

backup-while-open (BWO). This makes a backup copy of a data set while the data set is open for update. The backup copy can contain partial updates.

basic catalog structure (BCS). The name of the catalog structure in the integrated catalog facility environment. See also *integrated catalog facility catalog*.

binder. The DFSMS/MVS program that processes the output of language translators and compilers into an executable program (load module or program object). It replaces the linkage editor and batch loader in MVS.

C

compress. (1) To reduce the amount of storage required for a given data set by having the system replace identical words or phrases with a shorter token associated with the word or phrase. (2) To reclaim the unused and unavailable space in a partitioned data set that results from deleting or modifying members by moving all unused space to the end of the data set.

compressed format. A particular type of extended-format data set specified with the (COMPACTION) parameter of data class. VSAM can compress individual records in a compressed-format data set. SAM can compress individual blocks in a compressed-format data set. See *compress*.

D

DASD volume. A DASD space identified by a common label and accessed by a set of related addresses. See also *volume.*

data class. A collection of allocation and space attributes, defined by the storage administrator, that are used to create a data set.

data set control block (DSCB). A data set label for a data set in direct access storage.

Device Support Facilities (ICKDSF). A program used for initialization of DASD volumes and track recovery.

DFSMShsm. A DFSMS/MVS functional component or base element of OS/390, used for backing up and recovering data, and managing space on volumes in the storage hierarchy.

DFSMS/MVS. An IBM System/390 licensed program that provides storage, data, and device management functions. When combined with MVS/ESA SP Version 5 it composes the base MVS/ESA operating environment. DFSMS/MVS consists of DFSMSdfp, DFSMSdss, DFSMShsm, and DFSMSrmm.

DFSMS/MVS Network File System. See *OS/390 Network File System*.

© Copyright IBM Corp. 1979, 1999

E

extended format. The format of a data set that has a data set name type (DSNTYPE) of EXTENDED. The data set is structured logically the same as a data set that is not in extended format but the physical format is different. See also *striped data set* and *compressed-format*.

I

indexed VTOC. A volume table of contents with an index that contains a list of data set names and free space information, which allows data sets to be located more efficiently.

integrated catalog facility catalog. A catalog that is composed of a basic catalog structure (BCS) and its related volume tables of contents (VTOCs) and VSAM volume data sets (VVDSs). See also *basic catalog structure* and *VSAM volume data set*.

M

MANAGEMENT CLASS. A collection of management attributes, defined by the storage administrator, used to control the release of allocated but unused space; to control the retention, migration, and backup of data sets; to control the retention and backup of aggregate groups, and to control the retention, backup, and class transition of objects.

MVS/ESA. An MVS operating system environment that supports ESA/390.

MVS/ESA SP. An IBM licensed program used to control the MVS operating system. MVS/ESA SP together with DFSMS/MVS compose the base MVS/ESA operating environment. See also *OS/390*.

O

OpenEdition MVS. See *OS/390 UNIX System Services.*

optical volume. Storage space on an optical disk, identified by a volume label. See also *volume*.

OS/390. OS/390 is a network computing-ready, integrated operating system consisting of more than 50 base elements and integrated optional features delivered as a configured, tested system. See also *MVS/ESA SP.*

OS/390 Network File System. A base element of OS/390 that allows remote access to MVS host processor data from workstations, personal computers, or any other system on a TCP/IP network that is using client software for the Network File System protocol.

OS/390 UNIX System Services (OS/390 UNIX). The set of functions provided by the shell and utilities, kernel, debugger, file system, C/C++ Run-Time Library, Language Environment, and other elements of the OS/390 operating system that allow users to write and run application programs that conform to UNIX standards.

P

partitioned data set. A data set on direct access storage that is divided into partitions, called members, each of which can contain a program, part of a program, or data.

partitioned data set extended (PDSE). A system-managed data set that contains an indexed directory and members that are similar to the directory and members of partitioned data sets. A PDSE can be used instead of a partitioned data set.

program object. All or part of a computer program in a form suitable for loading into virtual storage for execution. Program objects are stored in PDSE program libraries and have fewer restrictions than load modules. Program objects are produced by the binder.

R

Resource Access Control Facility (RACF). An IBM-licensed program or a base element of OS/390, that provides for access control by identifying and verifying the users to the system, authorizing access to protected resources, logging the detected unauthorized attempts to enter the system, and logging the detected accesses to protected resources.

S

storage class. A collection of storage attributes that identify performance goals and availability requirements, defined by the storage administrator, used to select a device that can meet those goals and requirements.

storage control. The component in a storage subsystem that handles interaction between processor channel and storage devices, runs channel commands, and controls storage devices.

storage location. A location physically separate from the removable media library where volumes are stored for disaster recovery, backup, and vital records management.

storage management. The activities of data set allocation, placement, monitoring, migration, backup, recall, recovery, and deletion. These can be done either manually or by using automated processes. The Storage Management Subsystem automates these processes for you, while optimizing storage resources. See also *Storage Management Subsystem*.

Storage Management Subsystem (SMS). A DFSMS/MVS facility used to automate and centralize the management of storage. Using SMS, a storage administrator describes data allocation characteristics, performance and availability goals, backup and retention requirements, and storage requirements to the system through data class, storage class, management class, storage group, and ACS routine definitions.

stripe. In DFSMS/MVS, the portion of a striped data set that resides on one volume. The records in that portion are not always logically consecutive. The system distributes records among the stripes such that the volumes can be read from or written to simultaneously to gain better performance. Whether it is striped is not apparent to the application program.

striping. A software implementation of a disk array that distributes a data set across multiple volumes to improve performance.

striped data set. In DFSMS/MVS, an extended-format data set consisting of two or more stripes. SMS determines the number of stripes to use based on the value of the SUSTAINED DATA RATE in the storage class. Striped data sets can take advantage of the sequential data striping access technique. See *stripe* and *striping*.

system data. The data sets required by MVS or its subsystems for initialization and control.

T

tape subsystem. A magnetic tape subsystem consisting of a controller and devices, which allows for the storage of user data on tape cartridges. Examples of tape subsystems include the IBM 3490 and 3490E Magnetic Tape Subsystems.

tape volume. Storage space on tape, identified by a volume label, which contains data sets or objects and available free space. A tape volume is the recording space on a single tape cartridge or reel. See also *volume*.

U

use attribute. (1) The attribute assigned to a DASD volume that controls when the volume can be used to allocate new data sets; use attributes are *public*, *private*, and *storage.* (2) For system-managed tape volumes, use attributes are *scratch* and *private*.

UTC. Coordinated Universal Time

V

volume. The storage space on DASD, tape, or optical devices, which is identified by a volume label. See also *DASD volume*, *optical volume*, and *tape volume*.

volume status. In the Storage Management Subsystem, indicates whether the volume is fully available for system management:

- “Initial” indicates that the volume is not ready for system management because it contains data sets that are ineligible for system management.
- “Converted” indicates that all of the data sets on a volume have an associated storage class and are cataloged in an integrated catalog facility catalog.
- “Non-system-managed” indicates that the volume does not contain any system-managed data sets and has not been initialized as system-managed.

volume table of contents (VTOC). (1) A table on a direct access volume that describes each data set on the volume. (2) An area on a disk or diskette that describes the location, size, and other characteristics of each file and library on the disk or diskette.

VSAM volume data set (VVDS). A data set that describes the characteristics of VSAM and system-managed data sets residing on a given DASD volume; part of an integrated catalog facility catalog. See also *basic catalog structure* and *integrated catalog facility catalog*.

Index

Numerics

1403 printer
  UCS images 297
31-bit IDAW (indirect addressing word) 96
3203 Model 5 printer
  JES support 296
  UCS images 297
3211 printer
  indexing feature 314, 317
  JES support 317
  UCS images 297
3262 Model 5 printer
  alias names 302
  default FCB image 311
  print bands 306
  UCS image table 302, 305
3480/3490 Magnetic Tape Subsystem
  messages, displaying 211
  MSGDISP macro 211
  tape recording technique 119
3800 Printing Subsystem
  standard FCB image 312
4245 printer
  alias names 302
  default FCB image 311
  UCS image table 302
  UCS image table, contents 303
4248 printer
  alias names 302
  default FCB image 311
  image table, contents 304
  UCS image table 302
6262 Model 14 printer
  alias names 302
  default FCB image 311
  UCS image table 302, 305

A

ABE appendage
  conditions 134
  XENDA operand 113
actual track address to relative track address conversion routine 137
address space
  V=R, using EXCP 99
  V=V, using EXCP 99
alias
  retrieving catalog information 74
allocation
  DASD space 13
allocation *(continued)*
  retrieval area format 188
  retrieval list format 187
ANTRQST macro 363, 417
  environment 363
  LEVEL request 372
  parameters 371
  PDELPAIR request 373
  PDELPATH request 374
  PESTPAIR request 376
  PESTPATH request 378
  PFREEZE request 381
  PQUERY request 382
  PRECOVER request 384
  programming requirements 364
  PRUN request 387
  PSUSPEND request 388
  register Information 364
  return and reason codes 407
  SDVCINFO request 390
  SQRYDVCS request 393
  SQRYSSYS request 397
  SRELEASE request 399
  SSNAP request 400
  syntax 365
  XCONTIME request 403
APF (authorized program facility)
  EXCP 99, 100, 106
  EXCPVR 108
  VTOC operations 57
API 225, 363
appendages
  ABE (abnormal end) 134
  CHE (channel end) 136
  description 126
  entry points 127, 128
  EOE (end-of-extent) 134
  installing 128
  list, authorized 128, 134
  naming convention 129
  PCI (program controlled interruption) 133
  PGFX (page fix) 130
  programming restrictions 127
  returns 127, 128
  SIO (start I/O) 130, 131
  SYS1.PARMLIB listing 128
  work areas available 128
authorization
  RACF, DASDVOL 57, 58, 60

© Copyright IBM Corp. 1979, 1999

B

backing up a data set 231
binder (program management binder)
  defaults 245
  description 245
bit maps, allocated DSCBs, VIRs 15
BLDL 247
block ID, high-speed cartridge tape positioning 192
buffer
  data transfer rate with EXCP 115
  installing 115
  lists
    creating 43
    entries 44
    format 45
    function 43
    header format 44
    reading DSCBs 339
    releasing 47, 323
  operating system overhead 115
  performance considerations with BUFNO 115
  releasing 47, 323
BWO (backup-while-open) facility
  indicator bits 231

C

callable services
  data set attribute retrieval 229
  data set backup-while-open support 231
  description 225
  DFSMSdfp share attribute retrieval 237
  invoking 226
  nonreentrant program 225
  return and reason codes 239
  system level determination 226
CAMLST macro
  BLDA operand 83
  BLDG operand 80
  BLDX operand
    adding levels 78
    building 78
  BLOCK operand 76
  CAT(BX) operand 88
  DLTA operand 84
  DLTX operand 82
  DRPX operand 86
  format 24, 26
  LNKX operand 85
  RECAT operand 91
  RENAME operand 64, 68
  SCRATCH operand 59, 64
  SEARCH operand 24, 25
  SEEK operand 26, 27
  UNCAT operand 90
cartridge tape
  high-speed positioning 192
catalog
  entry
    non-VSAM data set 88
    reading 70
  indexes 78
  information retrieval
    alias 74
    data set name 71
    GDS name 73
  maintenance 71
  management
    information retrieval 70
    macro functions 69
  non-VSAM data sets
    example 89
    macro specifications 88
    return codes 93
  search order 70
CATALOG macro
  CAT(BX) operand 88
  RECAT operand 91
  UNCAT operand 90
CCW (channel command word) xv, 99
  See also channel programs
CDRA 225, 226
chaining check 134
channel
  control check 134
  data check 134
channel programs
  appendages 126
  command chaining 97
  completion codes 98, 101
  control information 98
  data set characteristics 98
  EXCP
    initiation and execution 98, 101
    modification restrictions 99
    related 102
    translating virtual addresses to central storage addresses 136
  problem program
    communication 98
    executing 96
  processing requirements 98
  real storage 108
  seek address 123
  translation
    V=R address space 131, 132
    V=V address space 100
    virtual addresses to central storage addresses 131
  XDAP 142, 146

CHE appendage
  CENDA operand 113
  description 136
CLOSE macro
  EXCP 108
  XDAP 143
command retry 100
communication vector table (CVT) 136, 139
completion codes xv
  See also return codes
  EXCP I/O 125
  XDAP I/O 146
control block
  data facilities area (DFA) 425
  DCBE (data control block extension) 120
  DEB-EXCP(VR) 419
  DFA 425
  EXCP 120
    DCB 98, 110, 120
    DCBE 98
    DEB 98, 126, 419
    ECB 98, 124
    IEDB 98, 124
    IOB 98, 120, 124
    IOBE 98, 123
  fields used with EXCP 120
  fields-EXCP(VR) 419
  purged I/O restore list 197
  XDAP 145
control password 152
control-volume pointer 85
conversion
  actual track address to relative track address
    register usage 137
  relative track address to actual track address
    procedure 136
    register usage 136
    return codes 137
  sector value, RPS devices 138, 146
  virtual channel program to real channel
    program 108, 131
  VTOC 57
CSW (channel status word) 62, 100
CVAF (common VTOC access facility)
  addressing mode 40
  buffer 323
    freeing 324
  filter service
    control blocks required 50
    invoking 337
    reading sets of DSCBs 49
  macros
    coding 319
    uses and format 319
    VTOC access 40, 55
  parameter list xv
    See also CVPL
CVAF (common VTOC access facility) *(continued)*
  return codes
    PARTREL 32
    RENAME 66
    SCRATCH 62
  serialization 40
  status codes 62, 66
  tracing calls 56
  volume identification 41
CVAFDIR macro
  coding 319
  examples 325, 330
  parameters 320, 325
  return codes 325
  using 40, 46
CVAFDSM macro
  format 330
  parameters 331, 336
  return codes 336
  using 40, 330
CVAFFILT macro
  control block address resolution 338
  DSCBs, reading 49
  entry 341
  example 342
  filter criteria list
    entry format 52
    header format 50
  format 337
  forms 341
  invocation sequences 53, 55
  parameters 339, 341
  partially-qualified names, examples 342
  RESUME capability 49
  return codes 342
  RLSE function 55
  using 40, 337
CVAFSEQ macro
  DSCB or DSN access 47
  examples 352, 359
  format 347
  parameters 348, 352
  return codes 352
  using 40, 347
CVAFTST macro
  format 359
  return codes 360
  using 40, 359
CVFCTN field of CVPL 43
CVOL
  assigning alias to index 83
  block 76
  catalog management macros 69
  connecting 85
  disconnecting 85
  indexes
    adding levels 78

CVOL *(continued)*
indexes *(continued)*
alias assignment 83
alias deletion 84
building 78
deleting levels 82
generation index 80
return codes 87
names 69
non-VSAM data sets 88
parameter list 69
reading a block 76
return codes 94
volume serial in CAMLST 69
CVPL (CVAF parameter list)
creation 41
format 42, 43
function 41, 43
initializing 337
mapping 337
CVSTAT codes 32, 361

D

DADSM (direct access device space management) 28
allocate routine 13
return codes 39
definition 1
non-VSAM data sets, deleting 59
OBTAIN routine
reading by absolute device address 26
reading by data set name 24
return codes 25, 27
RENAME macro
return codes 66
status codes 67
return codes 22
SCRATCH macro
return codes 62
status codes 63
DASD (direct access storage device)
block size 115
buffer allocation 115
data set, creating 34
data transfer rate 115
password protected data sets 150
reading and writing to, with XDAP macro 141, 145
releasing space 13
track capacity calculations 198
volume
fragmentation information 16
initializing 58
restoring from tape 58
space allocation 15
space information 16, 24
VTOC status 16
DASDVOL
RACF, class 57, 58, 60
data
areas, fixing with EXCPVR 109
class
name 229
control block xv
See also DCB
extent block xv
See also DEB
data control block extension (DCBE)
used with EXCP 120
Data Extent Block (see DEB) 419
data facilities area (DFA)
control block 425
data set
allocation
absolute 34
movable 34
attribute retrieval 229
backup 231
control block xv
See also DSCB
defining 2
deleting
macro instructions 59, 64
shared cylinders 60, 61
stored across devices 60
VIO processed 60
device, assigning 116
DSCB formats 2
expiration date 60
ISAM index 8
non-VSAM
recataloging 91
organization 114
password protection 147, 150
renaming
password protection 65
VTOC 64
WRITE protection mode indicator 65
repositioning on tape 113
security
access types 148
concatenation 151
space allocation, releasing 27
space, releasing 16
user label extent 34
data set attribute retrieval 229
DCB (data control block)
address 98
buffer parameters 114
device dependent parameters (EXCP) 116, 120
device-dependent format 114
EXCP 98, 120
fields 110

DCB (data control block) *(continued)*
format after OPEN (EXCP) 112
generating 98
initializing 120
macro instruction
XDAP 142
OPEN installation exit 115
parameters
device dependent 116
EXCP 110
foundation block 113
restoring 108
DCBD mapping macro for EXCP 120
DCBE (data control block extension) 97, 120
EXCP 98
DCBFDAD field 118
DCBOFLGS field 106, 108
DCBTRBAL field 118
DDR (dynamic device reconfiguration)
repositioning tape data sets 113
DEB (data extent block)
EXCP 98
fields 126
layout (EXCP and EXCPVR) 419
obtaining 41
validating with DEBCHK 159, 163
DEBCHK macro
functions 159, 163
list form 163
register contents 162
return codes 163
specification 160, 163
DELETE
as functional replacement 247
DEQ macro, tape volume demount facility 190
DESERV Exit
DELETE 247
GET 247
global exit 249
interaction between task and global exits 249
PUT 247
RENAME 247
SVC Screening 247
SVCUPDTE 247
task level exit 248
UPDATE 247
determining DFSMSdfp release level 226
determining release level 207
DEVD
DCB operands 119
parameters 116
device
characteristics table entry (DCTE) 200
characteristics, I/O 163
dependent parameters 116
end error recovery procedures 101
DEVTYPE macro
examples 177
execute form 168
INFO form 170
INFOLIST type 165
list form 165, 169
output 174, 176
return, reason codes 176
RPS devices 166
specification 164, 165
UCBLIST type 165
DFA
fields 425
mapping of 207
DFADEEXT 253
DFSMS/MVS level determination 207
DFSMSdfp callable services
data set
attribute retrieval 229
backup-while-open support 231
description 225
DFSMSdfp share attribute retrieval 237
invoking 226
reason codes 239
return codes 239
system attribute call 226
See also system level determination
system level determination 226
DFSMSdfp level determination 207, 226
DFSMSdfp share attribute retrieval 237
DSCB (data set control block) 4, 13
absolute track address 24
access
direct 46
indexed 48
physical sequential 48
sequential 47
buffer generation 325
chains 54, 337
format sequence 2
format-7 13
identifier 4
ISAM data set index 8
mapping 325
nonindexed VTOC 12
qualified data set name 337
reading
buffer list 339
by absolute device address 26
by address 319
by data set name 24, 319
directly by data set name 46
directly by DSCB location 47
sequentially 47, 355
sets, CVAF filter service 49
VTOC 327

DSCB (data set control block) *(continued)*
retrieving 339
VSAM data space extents 9
VTOC contents 10
VTOC types 2
writing
by address 319
by data set name 319
directly by data set name 46
directly by DSCB location 47
DSN order, accessing DSNs, DSCBs 48
DSSIZE value 36
DSTYPE data set attribute 229
dynamic device reconfiguration 113

E

ECB (event control block)
completion code 66
completion codes 62
EXCP 98, 124
XDAP 146
end-of-data-set
condition 143
routine 114
EOE (end-of-extent) appendage
description 134
EOEA operand 113
EOV (end-of-volume)
erasing data 59
EXCP 106, 108
macro 108
end-of-data-set routine, user 106
EXCP 106
XDAP 143
magnetic tape 106
error
CVAF VTOC index 360
I/O recovery procedures (EXCP) 102
routines, ignoring 114
VTOC or index
processing 55
recovery from system or user errors 56
event control block xv
See also ECB
examples
ANTRQST macro 416
DEVTYPE macro 177
exception conditions 114
EXCP (execute channel program)
ABE appendage 134
appendages 126
CHE appendage 136
command chaining 97
completion processing 101
control blocks 120
DCB 98, 110, 120
EXCP (execute channel program) *(continued)*
control blocks *(continued)*
DCBE 98
DEB 98, 126
ECB 98, 124
error recovery 121
IOB 98, 120, 124
DCBE 120
device-end error recovery procedures 101
EOE appendage 134
I/O error handling 102
indirect addressing word requirements 96
initiation and execution 98, 101
interface parameters 113
macro instruction 110
macros
CLOSE 108
EOV 106, 108
format 106
OPEN 105, 106
WAIT 96
modifying 99
multivolume data set requirement 105
overhead 115
PCI appendage 133
PDSE 95
problem programs 95
programming 95
considerations 100
interfaces 96
real
channel programs 108
storage 108
return codes 62
SIO appendage 130
status information 124
translation by system 131
V=R address space 99
EXCP(VR)
DEB layout 419
executing your own channel programs 95
EXCPVR macro
format 108
using 108
expiration date, overriding 60, 61
extent area
ICVEDT02 mapping macro 45
extents
allocating 34
available 16
control information 98
end of, appendage 113
free space 13
tracking 4
VSAM data space 9

F

FCB (forms control buffer) image
  JES support 317
  standard image STD1 312
  standard image STD2 313
  standard images 311
  SYS1.IMAGELIB
    adding image 313, 315
    function 311, 316
    modifying image 316
FCL (filter criteria list)
  entry format 52
  header format 50
file mark, writing 117
filter criteria list xv
  See also FCL
format 0-6 DSCBs, overview 2
format-1 DSCB, reading from VTOC 24
formats 13
  free VTOC record 4
forms control buffer image xv
  See also FCB image
free space
  DASD volume 16, 24
  reestablishing 56
free VTOC record 4

G

GDS (generation data set)
  retrieving by name 73
generation
  index 80, 82
  name
    absolute 73
    relative 73
  number 73
GET
  DESERV 247
global exit (DESERV) 249
GTF trace of CVAF processing 56

H

HFS data set 1, 5, 230
HFS file
  device characteristics information 173
  DEVTYPE returning UCBTYP 172
  RENAME macro 65
high-speed cartridge tape positioning 192

I

I/O
  device characteristics 163
  efficiency, improving 108
I/O *(continued)*
  purged restore list 197
  requests
    purging 194
    reprocessing 198
    restoring 194
ICVEDT02 mapping macro 45
IDAL (indirect data address list) 131
IDAW (indirect addressing word) 96
IEAAPP00 128
IEBUPDTE program
  SYS1.PARMLIB appendage listing 128
IEC502E message 190
IECDPPL macro 195
IECPDSCB macro 35
IECSDSL1 macro 3, 325
IEDB (input/output error data block)
  EXCP 98
  status information 104
  VIO considerations 105
IEEE
  POSIX 1
IEHLIST program 56
IEWPMAR macro 278, 293
IEZDEB
  fields-EXCP(VR) 419
IGGUCSIT macro 307
IGWABWO call statement
  format 231
  return and reason codes 239
  using 234
IGWARLS call statement
  format 240
  return codes 242
IGWASMS call statement
  format 229
  return and reason codes 239
IGWASYS call statement
  format 226
  return and reason codes 239
IGWDES macro 252, 293
IGWLSHR call statement
  format 237
  return and reason codes 239
IGWSMDE macro 278, 293
IHAARL macro 186
IHADFA
  control block 425
  fields 425
  macro 207
IHADVA macro 163
IHAPDS macro 257
image
  alias name 297
  identifier 313
  length 298

image *(continued)*
library
adding FCB image 313
IMAGELIB macro 316
maintaining 295, 316
modifying an FCB image 316
printer information 296
table
entries, adding or modifying 307
object module 307
verification 298, 310
IMGLIB macro 316
INDEX macro
building and deleting indexes 83
DRPX operand 86
LNKX operand 85
indexed VTOC xv
See also VTOC
buffer
disposition 323
contents 10
conversion 57
DSCB
access 48
reading 319
writing 319
initializing 57
listings 56
modifying 319
password protection 57
protecting 57
records 319
structure 15
system
error 56
support 359
unauthorized user 56
volume
restoring 58
updating 58
volume update 58
indexing feature for 3211 314, 317
indirect
addressing word (IDAW) 96
data address list (IDAL) 131
INFO form
DEVTYPE macro 170
INFOLIST parameter
DEVTYPE macro 165
intercept condition 134
interface control check 134
interruption handling procedures 102
IOB (input/output block)
chain modification 197
EXCP 98
fields used with EXCP 120, 124
IOB (input/output block) *(continued)*
sense bytes 62, 66
XDAP 145
XDAP fields 145
IOBE (input/output block common extension)
EXCP 98
requesting extended error information 103
IPL volume 149
ISAM (indexed sequential access method)
data set
allocating 34
counting 352
DSCB 2
indexes 8
VTOCs 2

J

JES (job entry subsystem)
processing printed output 302
JFCB (job file control block) xv
See also RDJFCB macro
macros used with
IHAARL 186, 190
OPEN 193
RDJFCB 180, 185
modifying
functions 178, 194
precautions 178
type 07 exit list entry 182
JOBCAT/STEPCAT DD statement
catalog search order 70

K

KEYLEN operand 119

L

labels, standard 106
level
DFSMSdfp or DFSMS/MVS 207, 226
library
callable services 225
FCB images 316
maintenance 316
printer control information 295
SYS1.IMAGELIB data set 295
list form, DEVTYPE macro 169
LOCATE macro
alias name 74
data set name 71
generation name 73
relative block address 76
retrieving catalog information 71
return codes 77

locate-record CCW (EXCP) 100
LSPACE macro 21
data return area 23
description 24
format 16
data return area 23
expanded message return area 22
message return area 22
parameter list 20
message return area 23

M

macro
data management
CATALOG 88
LOCATE 70, 77
MSGDISP 211, 224
macro, mapping
ICVEDT02 45
macros system
DEVTYPE macro 168
macros, data management
ANTRQST 363, 417
CAMLST
RENAME operand 64, 68
SCRATCH operand 59, 64
SEARCH operand 24, 25
SEEK operand 26, 27
CLOSE
EXCP 108
XDAP 143
CVAF VTOC access 40, 55
CVAFDIR 46, 319
CVAFDSM 330
CVAFFILT 49, 337
CVAFSEQ 47, 347
CVAFTST 359
DCB 120
EXCP 110
XDAP 142
DEBCHK 160, 163
DEVTYPE 163, 178
EOV
EXCP 106, 108
XDAP 143
EXCP 106, 108, 110, 120
EXCPVR 108
LSPACE 16, 24
OBTAIN 24, 27
OPEN
EXCP 105, 106
modified JFCB 193
XDAP 142
PARTREL 27, 33
PROTECT 152, 158
macros, data management *(continued)*
PURGE 194
RDJFCB 178, 185
REALLOC 34, 40
RENAME 64, 68
RESTORE 198
SCRATCH 59
TRKCALC 198, 206
using XDAP 142, 143
VTOC access
DADSM 15, 40
XDAP 141, 145
management class
name 229
mapping macros
IECSDSL1 325
IHADFA 207
messages
CVAF VTOC index error 360
displaying 211
displaying on 3480/3490
options 221
parameters 211
ready 216
resetting display 220
verify volume 214
volume demount 217
volume mounting 211
IEC502E 190
LSPACE macro 22
MSGDISP macro 211
MINAU parameter 36
MSGDISP macro
parameters
demount message 217
display options 221
ready message 216
reset message display 220
verify volume 214
volume mounting 211
return codes 224
multivolume data set
processing with EXCP 105

N

name
VTOC index 14
nocapture option 107, 422
non-VSAM
data set
cataloging 88
recataloging 91
uncataloging 90
nonindexed VTOC
available space 12

nonindexed VTOC *(continued)*
structure 15
NOPWREAD protection-mode indicator 152
NOWRITE protection-mode indicator 152

O

OBTAIN macro 24, 27
OBTAIN macro, VIO data sets, 24
OBTAIN macro, VSAM data sets, 24
OPEN macro
DEQ at demount facility, tape volumes 190
EXCP
dummy data set restriction 105
procedures performed 105
modified JFCB 193
TYPE=J
example 184
invoking 190
specification 194
XDAP 142
OPENJ (OPEN, TYPE=J) macro 193
out-of-extent error 134
overhead, system 115
overwriting data 59
See also EOV (end-of-volume), erasing data

P

page
boundary, CCW data address crossing 131
fix
appendage 130
list processing 130
formatting 314
paging exceptions 130
partial DSCB 35
partitioned data set
space
releasing 27, 33
PARTREL macro
description 27, 33
DSECT form 31
list form 31
return codes 32
password xv
See also PROTECT macro
control 152
counter maintenance 152
data set concatenation 151
deleting protected data set 151
parameter list
ADD record 153
DELETE record 156
LIST record 157
REPLACE record 155
password *(continued)*
protection
data sets 147, 158
JFCB modifications 183
mode indicator 152
tape data sets 150
volume switching 151
VTOC indexes 57
record 150
renaming protected data set 151
secondary 152
PASSWORD data set
creating 150
maintenance 152
requirements 149
PCI (program controlled interruption)
appendage 113, 133
CCW modification 100
description 133
PCIA operand 113
PD1SCALO flag byte 36
PDDIRQTY (number of directory blocks) 35
PDPRIQTY (primary space request in tracks) 35
PDSE (partitioned data set extended)
attribute retrieval 229
sharing protocol 237
space
releasing 27, 33
releasing unused 16
TRKCALC macro 198
PGFX appendage 130
POSIX 1
posting completion code in ECB
EXCP I/O 124
printer
bands
3262 Model 5 306
alias names 302
national band IDs 305
special order (RPQ) 306
storing 302
character set images 295
control information 295
default image
print position indexing feature 313
specifying defaults 313
standard FCB images 311
image xv
See also UCS image
image library 296
page layout 314
UCS image table 297
program check 134
PROTECT macro xv
See also password
format 153

PROTECT macro *(continued)*
functions 152
parameter list
ADD function 153
DELETE function 156
LIST function 157
REPLACE function 155
PASSWORD data set 147, 157
protection-mode indicator 152
return codes 157
protection
check 134
mode indicator 152
VTOC index 57
PURGE macro
parameter list 195
return codes 197
specification 195
purge routines 194
purged I/O restore list 197
PUT
as functional replacement 247
PWREAD protection-mode indicator 152
PWWRITE protection-mode indicator 152

R

RACF
DASDVOL class 57, 58, 60
RACF (Resource Access Control Facility)
renaming a data set 64
return codes 62, 66
scratching a data set 59
VTOCs and VTOC indexes 57
RDJFCB macro
DCB exit list entry
type '13' 186
type 07 182
description 180
example 181
format
allocation retrieval area 188
allocation retrieval list 187
invoking DEQ at demount 190
retrieving allocation information
DCB exit list entry 186
example 188
return codes 182
security 183
specification 180
use by authorized programs 183
REALLOC macro
description 34, 40
DSECT form 38
execute form 35
list form 38
REALLOC macro *(continued)*
return codes 39
reason codes
DEVTYPE macro 176
record
current location 117
register
contents 126
conversion routines usage
actual to relative 138
relative to actual 136
related requests 102, 122
relative
block address 76
generation number 73
to actual track address conversion routine 136
release, determining level of 207
release, determining level of DFSMSdfp 226
RENAME
as functional replacement 247
RENAME macro
example 66
return codes 66
SMS-managed volumes 65
specification 64
status codes 67
requesting different levels of ERP processing 104
requesting extended error information 103
Resource Access Control Facility xv
See also RACF
RESTORE macro 198
restore routines 195
retrieving data set attributes 229
retrieving DFSMSdfp share attributes 237
return codes
CATALOG macro 93
CVAF VTOC index error message 360
CVAFDIR macro 325
CVAFDSM macro 336
CVAFFILT macro 342
CVAFSEQ macro 352
CVAFTST macro 360
DADSM allocation 39
DEBCHK macro 163
DEVTYPE macro 176
IGWABWO call statement 239
IGWARLS call statement 242
IGWASMS call statement 239
IGWASYS call statement 239
IGWLSHR call statement 239
INDEX macro 87
LOCATE macro 77
LSPACE macro 21
MSGDISP macro 224
OBTAIN macro
reading from VTOC by absolute device address 27

return codes *(continued)*
OBTAIN macro *(continued)*
reading from VTOC by data set name 25
PARTREL macro 32
RDJFCB macro 182
REALLOC macro 39
RENAME macro 66
SCRATCH macro 62
track address convert routine 137
TRKCALC macro 206
rotational position sensing 166
RPS (rotational position sensing)
device sector number 138, 146
parameter 166

S

SCRATCH macro
description 59
example 61
return codes 62
status codes 63
scratching a data set 59, 64
secondary passwords 152
sector
address
(RPS device) 138, 146
XDAP macro 144
conversion routine 138, 146
number (RPS device) 138, 146
sensitive data, erasing 59
sequential data set
space, releasing 27
serializing CVAF requests 40
share attributes, retrieval of DFSMSdfp 237
SIO (start I/O) appendage
entry points 130
EXCP (execute channel program) 130
SIOA operand 113
SMF (System Management Facilities)
volume information 19
SMS (Storage Management Subsystem)
class names 229
data set, deleting 59
indexed VTOC 13
release level 226
status 226
version 226
VTOC
deleting data sets 59
space allocation
releasing unused 27
space map 15
start I/O appendage 130
status codes
RENAME macro 67
status codes *(continued)*
SCRATCH macro 63
storage class. name 229
STOW 247
SUFFIX, block 172
SVC Screening 247
SVCUPDTE 247
replacing SVC routine 247
switching volumes, password protection 151
syntax diagrams, how to read xviii
SYS1.CSSLIB data set 225
SYS1.IMAGELIB data set
adding
FCB image 313, 315
UCS image 297
alias name 302
maintaining 295, 316
modifying FCB image 316
UCS image tables 302
SYSCTLG data set 69
system
attribute call 226
See also system level determination
macro instructions 159
system data mover API 363
system level determination 226
system residence volume 149

T

tape
block count field 123
end-of-volume 106
recording
density 119
technique 119
reduced error recovery 113
reflective spot 106
repositioning data sets 113
tapemark 106
volumes
DEQ at demount facility 190
password protected data sets 150
tape, cartridge
high-speed positioning 192
task level exit (DESERV) 248, 249
track
balance 199, 201
calculating capacity 198, 206
deleting a record 201
fixed-length records 200
TRKCALC output 205
TRKCALC macro
description 198, 206
DSECT form 204
examples 206

TRKCALC macro *(continued)*
execute form 202
inconsistency with PDSE 198
list form 204
output 205
parameter list
creating 204
empty, in-line 204
initialization 202
remote 202
storage definition 202
symbolic expansion 204
return codes 206
standard form 198
TRKCALC, macro
for extended sequential d.s. 172
TRTCH operand 119
TYPE operand
DEBCHK macro 161
OPEN macro 194
TYPE=J (OPEN macro) 193

U

UCB (unit control block)
index 123
mapping macro 173
track address 201
UCBLIST parameter
DEVTYPE macro 165
UCS (universal character set)
image
adding, JCL 299
alias name 302
creating 297
SYS1.IMAGELIB addition 297
SYS1.IMAGELIB, examples 299, 301
verifying 310, 311
image table
adding data 309
aliases 307
entry format 303
image names 307
modifying entries 307
object module 307
UCS5 contents 303
UCS6 contents 304
system image library maintenance 295
VERIFY parameters 311
UE (unit exception) 136
unit
check 134
control block xv
See also UCB
exception 134
universal character set xv
See also UCS
UPDATE
as functional replacement 247

V

V=R address space 99
V=V address space 100
validating the DEB (DEBCHK) 159, 163
VCB (volume control block) 71
VIER (VTOC index entry record)
characteristics 14
description 13
index search 319
reading 319
retaining in virtual storage 323
VIO data sets
EXCP 105
OBTAIN macro 24
VIR (VTOC index record) 13
buffer, releasing 320
reading 319
releasing 319
virtual
IDAW (indirect addressing word) 96
VIXM (VTOC index map)
bit maps
allocated DSCBs, VIRs 15
description 15
maps of allocated space
for VIRs 15
volume
copying (VTOC) 58
indexed VTOC 58
list 70, 71
description 59
renaming 64
list entry
scratch status code 62, 66
secondary status code 62, 66
nonindexed VTOC 58
restoring from tape 58
space allocation 15
swapping with DDR 113
switching
during EOV 106
multivolume data sets 151, 184
password protection 151
table of contents xv
See also VTOC
tape, protecting 150
volume, IPL 149
VPSM (VTOC pack space map)
allocated cylinders and tracks 15
description 15

VSAM (virtual storage access method)
catalog
search order 70
data space, defining 2
VSAM data sets, OBTAIN macro 24
VTOC (volume table of contents)
access 46
CVAF macros 55
DADSM macros 15, 40
DSCB directly, with CVAFDIR 46
DSCB sequentially, with CVAFSEQ 47
DSN sequentially, with CVAFSEQ 47
access macros
CVAFDIR 46, 319
CVAFDSM 330
CVAFFILT 49, 337
CVAFSEQ 47, 347
CVAFTST 359
LSPACE 16
PARTREL 27
REALLOC 34
SCRATCH 59
APF authorization 57
contents 10
conversion 57
data set renaming 64
deleting
non-VSAM data sets 59
temporary VSAM data sets 59
description 1
DSCB
defining data sets 2
formats 2
map xv
nonindexed 2
reading 319
writing 319
index
contents 14
creating 57
description 13
entry record (VIER) 14
error messages 360
listing 56
maintaining 59
map (VIXM) 15
name 14
password protection 57
RACF protection 57
record 13
relationship 14
renaming 14
structure 14
testing 359
volumes 58
initializing 57
VTOC (volume table of contents) *(continued)*
locating 1
map of DSCBs (VMDS) 13
modifying 59, 319
nonindexed 12
pack space map (VPSM) 15
protection 57
reading
by data set name 24, 27
DSN order 347
physical sequential order 347
recording facility 15
records 319
structure 15
system support 359
updating 41
using 1
volume space allocation 15

W

WAIT macro 96
WLR (wrong-length record) 136
wrong-length indication 134

X

XDAP (execute direct access program)
channel program 146
control blocks
DCB 142
ECB 146
IOB 145
macros
CLOSE 143
DCB 142
EOV 143
OPEN 142
specification 143, 145
requirements 141, 143

File Number: S370/S390-30
Program Number: 5695-DF1
5647-A01

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC26-4921-04

Spine information:

IBM DFSMS/MVS Version 1 Release 5 DFSMSdfp Advanced Services