Note
Before using this information and the product it supports, read the information in "Notices" on page 499.

This edition applies to Version 2 Release 1 of z/OS (5650-ZOS) and to all subsequent releases and modifications until otherwise indicated in new editions.

© Copyright IBM Corporation 1979, 2014.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
# Chapter 4. Executing Your Own

## Channel Programs

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comparing EXCP and EXCPVR</td>
</tr>
<tr>
<td>Using EXCP and EXCPVR</td>
</tr>
<tr>
<td>Allocating the Data Set or Device</td>
</tr>
<tr>
<td>Opening the Data Set</td>
</tr>
<tr>
<td>Direct Data Set Considerations</td>
</tr>
<tr>
<td>VSAM Data Set Considerations</td>
</tr>
<tr>
<td>Creating the Channel Program</td>
</tr>
<tr>
<td>CCW Channel Program</td>
</tr>
<tr>
<td>zHPF Channel Program</td>
</tr>
<tr>
<td>Comparing CCW and zHPF channel programs</td>
</tr>
<tr>
<td>EXCP 64-bit Storage Considerations</td>
</tr>
<tr>
<td>IDAW Requirements for EXCP Requests</td>
</tr>
<tr>
<td>IDAW Requirements for EXCPVR Requests</td>
</tr>
<tr>
<td>MIDAW Requirements</td>
</tr>
<tr>
<td>TIDAW requirements for EXCP requests</td>
</tr>
<tr>
<td>Determining Whether zHPF is Supported for a Device</td>
</tr>
<tr>
<td>Modifying a Channel Program During Execution</td>
</tr>
<tr>
<td>VIO Considerations</td>
</tr>
<tr>
<td>Creating the EXCP-Related Control Blocks</td>
</tr>
<tr>
<td>Input/Output Block (IOB)</td>
</tr>
<tr>
<td>Input/Output Block Common Extension (IOBE)</td>
</tr>
<tr>
<td>Event Control Block (ECB)</td>
</tr>
<tr>
<td>Input/Output Error Data Block (IEDB)</td>
</tr>
<tr>
<td>Data Control Block (DCB)</td>
</tr>
<tr>
<td>Data Control Block Extension (DCBE)</td>
</tr>
<tr>
<td>Data Extent Block (DEB)</td>
</tr>
<tr>
<td>Executing the Channel Program</td>
</tr>
<tr>
<td>Using the EXCP macro instruction</td>
</tr>
<tr>
<td>Using the EXCPVR macro instruction</td>
</tr>
<tr>
<td>Initiating the Channel Program</td>
</tr>
<tr>
<td>Translating the Channel Program</td>
</tr>
<tr>
<td>DASD Channel Program Prefix CCW Commands</td>
</tr>
<tr>
<td>DASD Rotational Positioning Sensing</td>
</tr>
<tr>
<td>Command Retry Considerations</td>
</tr>
<tr>
<td>Magnetic Tape Considerations</td>
</tr>
<tr>
<td>Lost Data Condition on IBM 3800</td>
</tr>
<tr>
<td>Processing the I/O Completion Status</td>
</tr>
<tr>
<td>Interruption Handling and Error Recovery Procedures</td>
</tr>
<tr>
<td>Handling End of Volume and End-Of-Data-Set Conditions</td>
</tr>
<tr>
<td>Closing the Data Set</td>
</tr>
<tr>
<td>Control Block Fields</td>
</tr>
<tr>
<td>Data Control Block (DCB) Fields</td>
</tr>
<tr>
<td>Data Control Block Extension (DCBE) Fields</td>
</tr>
<tr>
<td>Input/Output Error Data Block (IEDB) Fields</td>
</tr>
<tr>
<td>Input/Output Block (IOB) Fields</td>
</tr>
<tr>
<td>Input/Output Block Common Extension (IOBE) Fields</td>
</tr>
<tr>
<td>Event Control Block (ECB) Fields</td>
</tr>
<tr>
<td>Data Extent Block (DEB) Fields</td>
</tr>
<tr>
<td>EXCP and EXCPVR Appendages</td>
</tr>
<tr>
<td>Making Appendages Available to the System</td>
</tr>
<tr>
<td>The Authorized Appendage List (IEAAP00)</td>
</tr>
<tr>
<td>Start-I/O Appendage</td>
</tr>
<tr>
<td>Page Fix and EXCPVR Start I/O Appendage</td>
</tr>
<tr>
<td>Program-Controlled Interruption Appendage</td>
</tr>
<tr>
<td>End-of-Extent Appendage</td>
</tr>
<tr>
<td>Abnormal-End Appendage</td>
</tr>
<tr>
<td>Channel-End Appendage</td>
</tr>
<tr>
<td>Converting a Relative Track Address to an Actual Track Address</td>
</tr>
<tr>
<td>Return Codes from the Relative to Actual Conversion Routine</td>
</tr>
<tr>
<td>Converting an Actual Track Address to a Relative Track Address</td>
</tr>
<tr>
<td>Return Codes from the Conversion Routine</td>
</tr>
<tr>
<td>Using the IECTRKAD Callable Service or the TRKADDR Macro</td>
</tr>
<tr>
<td>Obtaining the Sector Number of a Block on an RPS Device</td>
</tr>
</tbody>
</table>

# Chapter 5. Using XDAP to Read and Write to Direct Access Devices

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Using XDAP</td>
</tr>
<tr>
<td>Macro Instructions Used with XDAP</td>
</tr>
<tr>
<td>Defining a Data Control Block (DCB)</td>
</tr>
<tr>
<td>Initializing a Data ControlBlock (OPEN)</td>
</tr>
<tr>
<td>End of Volume (EOV)</td>
</tr>
<tr>
<td>Restoring a Data Control Block (CLOSE)</td>
</tr>
<tr>
<td>Executing Direct Access Programs</td>
</tr>
<tr>
<td>Control Blocks Used with XDAP</td>
</tr>
<tr>
<td>Input/Output Block</td>
</tr>
<tr>
<td>Event Control Block</td>
</tr>
<tr>
<td>Direct Access Channel Program</td>
</tr>
<tr>
<td>RPS Device Sector Numbers</td>
</tr>
</tbody>
</table>

# Chapter 6. Using Password Protected Data Sets

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Providing Data Set Security</td>
</tr>
<tr>
<td>PASSWORD Data Set Characteristics</td>
</tr>
<tr>
<td>Creating Protected Data Sets</td>
</tr>
<tr>
<td>Protection Feature Operating Characteristics</td>
</tr>
<tr>
<td>Maintaining the PASSWORD Data Set Using PROTECT</td>
</tr>
<tr>
<td>Record Format</td>
</tr>
<tr>
<td>Protection-Mode Indicator</td>
</tr>
<tr>
<td>PROTECT Macro Specification</td>
</tr>
</tbody>
</table>

# Chapter 7. Using System Macro Instructions

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ensuring Data Security by Validating the Data Extent Block (DEBCHECK macro)</td>
</tr>
<tr>
<td>DEBCHECK Macro Specification</td>
</tr>
<tr>
<td>Obtaining I/O Device Characteristics (DEVTYPE macro)</td>
</tr>
<tr>
<td>DEVTYPE Macro Specification</td>
</tr>
<tr>
<td>HHA DVA Mapping macro</td>
</tr>
<tr>
<td>Reading and Modifying a Job File Control Block (RDJFCB Macro)</td>
</tr>
<tr>
<td>RDJFCB Macro Specification</td>
</tr>
<tr>
<td>DEQ at Demount Facility for Tape Volumes</td>
</tr>
<tr>
<td>High-Speed Cartridge Tape Positioning</td>
</tr>
<tr>
<td>OPEN - Initialize Data Control Block for Processing the JFCB</td>
</tr>
</tbody>
</table>
Purging and Restoring I/O Requests (PURGE and RESTORE macros) ........................................ 304
PURGE Macro Specification ........................................ 304
RESTORE Macro Specification ...................................... 307
Performing Track Calculations (TRKCALC macro) .... 307
Using TRKCALC ...................................................... 308
Restrictions .............................................................. 309
TRKCALC Macro Specification ................................. 309
Perform calculations and conversions with 28-bit cylinder addresses (TRKADDR macro) .......... 317
Calculate the relative track number on the volume (TRKADDR ABSTOREL) ............... 317
Compare two track addresses (TRKADDR COMPARE) ............................................ 318
Extract 28-bit cylinder number (TRKADDR EXTRACTCYL) .............................. 318
Extract 4-bit track number (TRKADDR EXTRACTTRK) ...................................... 318
Increment track address (TRKADDR NEXTTRACK) ............................................. 319
Normalize cylinder number (TRKADDR NORMALIZE) ......................................... 319
Convert a relative track number to a 28-bit cylinder address (TRKADDR RELTOABS) .... 320
Set cylinder number from register (TRKADDR SETCYL) ...................................... 320
Convert normalized track address into an absolute 28-bit track address (TRKADDR NORMTOABS) .................. 321
Determining Level and Name of DFSMS ................................................. 321
Determining Version, Release, and Modification Level of DFSMS .......................... 322
Determining Name of DFSMS ........................................... 323
Determining DFARELS During Assembler Macro Phase ........................................... 323
Chapter 8. Displaying Messages on Cartridge Magnetic Tape Subsystems (MSGDISP macro) ........................................ 325
MSGDISP—Displaying a Mount Message .............................................. 326
MSGDISP—Displaying a Verify Message .............................................. 328
MSGDISP—Displaying a Ready Message ............................................... 330
MSGDISP—Displaying a Demount Message ........................................... 331
MSGDISP—Resetting the Message Display ............................................. 334
MSGDISP—Providing the Full Range of Display Options ..................................... 336
Return Codes from MSGDISP ........................................... 338
Chapter 9. Using DFSMSdfp Callable Services ................................................. 341
Call for DFSMS Level Determination ........................................... 342
Format ................................................................. 342
Parameters ............................................................. 343
Return Codes .......................................................... 344
Example ................................................................. 344
Call for Data Set Backup-While-Open Support ............................................... 347
Format ................................................................. 347
Parameters ............................................................. 347
Return Codes .......................................................... 349
Example ................................................................. 349
Using the Backup-While-Open Facility .................................................. 350
Call for DFSMSdfp Share Attributes ................................................................. 353
Format ................................................................. 353
Parameters ............................................................. 353
Return Codes .......................................................... 354
IGWASY, IGWASY, IGWABWO, IGWLSHR Return and Reason Codes .............. 354
Call for Record-Level Sharing Query (IGWARLS) ........................................... 355
Format ................................................................. 355
Parameters ............................................................. 356
Return Codes .......................................................... 357
Example ................................................................. 358
Call for converting and comparing 28-bit cylinder addresses (IECTRKA) .................. 359
Format ................................................................. 359
Parameters ............................................................. 359
Character Data Representation Architecture (CDRA) APIs .................................. 362
Chapter 10. Using the DESERV Exit .................................................. 363
Task Level Exit ...................................................... 364
Global Exit ............................................................. 365
Interactions Between the Task Level and Global Exits ........................................... 365
Establishing Multiple Task level or Multiple Global Exits ..................................... 366
Issuing DESERV FUNC=EXIT (invocation environment) ................................... 366
Invocation Syntax .................................................... 367
Installing or Replacing the DESERV Exit .................................................. 369
Deleting the DESERV Exit .................................................. 370
Determining If a DESERV Exit Is Active .................................................. 371
Writing the DESERV Exit .................................................. 371
Parameters Related to the GET Function .................................................. 372
Parameters Related to the PUT Function .................................................. 384
PUT Return and Reason Codes .................................................. 386
Parameters Related to the DELETE Function .............................................. 389
Parameters Related to the RENAME Function .............................................. 390
Parameters Related to the UPDATE Function .............................................. 392
Entry Environment for Exit Routine .................................................. 394
Exit Environment for Exit routine .................................................. 394
Registers on Entry to the DESERV Exit .................................................. 395
Registers on Return from the DESERV Exit .................................................. 395
DESERV Exit Return and Reason Codes .................................................. 395
DESERV FUNC=EXIT Return and Reason Codes .............................................. 395
Example of the DESERV Exit .................................................. 398
Chapter 11. Managing Hierarchical File System Data Sets ........................................... 405
Creating Hierarchical File System Data Sets .................................................. 405
Creating and Mounting the Root File System .............................................. 406
Creating Additional File Systems and Directories ........................................... 406
Adding and Mounting File Systems to the Root .......................... 407
Managing File System Size .................................................. 407
Managing File System Activity .............................................. 408
Accessing HFS Data Set Attributes ........................................ 408
Transporting a File System ................................................... 408
Removing (Deleting) a File System ........................................... 409
Migrating a File System ...................................................... 409
Backing Up File Systems ...................................................... 409
Recovering a Backed-Up File System ...................................... 410
HFS Deferred File System Synchronization .............................. 410
How to specify a SYNC value ................................................ 411
Using pfsctl (BPX1PCT) Physical File System Control for HFS ......... 411
DisplayBufferLimits Command ............................................. 412
ChangeBufferLimits Command ............................................. 413
DisplayGlobalStats Command ............................................. 413
DisplayFSStats Command ................................................... 414
ExtendFS Command .......................................................... 414
BPX1PCT Return and Reason Codes ....................................... 415

Register 1 Parameter List ..................................................... 421
Passed Argument List -- SSGARGL ........................................ 421

Appendix A. Control Blocks ..................... 443
Data Extent Block (DEB) Fields ............................................ 443
Data Facilities Area (DFA) Fields ......................................... 449

Appendix B. Maintaining the System Image Library .................. 455
UCS Images in SYS1.IMAGELIB ............................................. 457
Examples of UCS Image Coding .......................................... 458
UCS Image Alias Names .................................................... 462
UCS Image Tables in SYS1.IMAGELIB ..................................... 462
Alias Names in UCS Image Tables ......................................... 462
Adding or Modifying a UCS Image Table Entry ......................... 466
Verifying the UCS Image .................................................... 470
FCB Images in SYS1.IMAGELIB ............................................. 471
Adding an FCB Image to the Image Library ............................. 473

Modifying an FCB Image .................................................... 475
JES Support for the 3211 Indexing Feature ............................. 476

Appendix C. Using the extended address volume (EAV) migration assistance tracker ............. 479
Information conventions for the EAV migration assistance tracker ........................................ 480
Tracking information ....................................................... 480
Tracking value ............................................................. 481
DFSMS instances tracked by the EAV migration assistance tracker ........................................... 481
DCB Open of a VTOC ....................................................... 492
DCB Open of EAS eligible data set ....................................... 492
Other Sample exclusion list ............................................... 493
Summary of DFSMS instances ............................................. 494

Appendix D. Accessibility .......................... 495
Accessibility features ...................................................... 495
Using assistive technologies ............................................. 495
Keyboard navigation of the user interface ................................ 495
Dotted decimal syntax diagrams ........................................ 495

Notices ......................................................... 499
Policy for unsupported hardware ........................................ 500
Minimum supported hardware ......................................... 501
Trademarks ............................................................ 501

Index ................................................. 503
Figures

1. Locating the volume table of contents .......................... 1
2. Contents of VTOC - DSBCs Describing Data Sets on Volume That Has No VTOC Index .......... 3
3. Contents of VTOC on an extended address volume - DSBCs Describing Data Sets on Volume That Has No VTOC Index ..... 4
4. Example of the Relationship of a VTOC to Its Index ................. 18
5. DADSM LSPACE Free Space Information
   MF=(D,MSG) ........................................... 32
6. DADSM LSPACE Free Space Information Format, MF=(D,EXPMSG) ........ 34
7. Control Blocks Required for CVAF Filter Services .................. 68
8. Example of CVAFDIR Macro with VTOC Part 1 of 2 .................. 82
9. Example of CVAFDIR Macro with VTOC Part 2 of 2 ................. 83
10. zHPF channel program .............................................. 175
11. How EXCP translates an EXCP request with a single 16K storage area .......... 181
12. How EXCP translates an EXCP request with storage areas crossing page boundaries .......... 182
13. Using IOSPHPF to determine if zHPF is supported by a processor and device .......... 183
14. Data Control Block Format for EXCP (After OPEN) .................. 198
15. Device-dependent portion of the DCB with DEVD=DA and DSORG=PS (or DSORG=PO) ........ 204
16. Device-dependent portion of the DCB with DEVD=DA and DSORG=DA ................. 205
17. Device-dependent portion of the DCB with DEVD=TA and DSORG=PS .................. 205
18. Device-dependent portion of the DCB with DEVD=PR and DSORG=PS .................. 205
19. Device-dependent portion of the DCB with DEVD=PC or RD and DSORG=PS .......... 206
20. Format of an IEDB, Mapped by the IOSDIEDB Macro ............... 210
21. Input/Output Block (IOB) Format .................................. 212
22. IOBFLAG3 and IOBCSW fields for format 0 channel program .......... 215
23. IOBFLAG3 and IOBCSW fields for format 1 channel program .......... 215
24. IOBFLAG3 and IOBCSW fields for zHPF channel program .......... 215
25. Format of an IOBE, mapped by the IOSDIIOBE macro .................. 217
26. Event Control Block after Posting of Completion Code .............. 220
27. Parameter List for ADD Function ................................. 254
28. Parameter List for REPLACE Function ............................. 255
29. Parameter List for DELETE Function ............................... 256
30. Parameter List for LIST Function ................................ 257
31. Examples of Standard Form of the RDJFEB macro .................... 288
32. Example Code Using RDJFEB Macro .................................. 289
33. Processing a Multivolume Data Set with EXCP ....................... 293
34. Sample Code Retrieving Allocation Information Part 1 of 2 .......... 298
35. Sample Code Retrieving Allocation Information Part 2 of 2 .......... 299
36. Examples of Standard Form of the OPEN TYPE=J Macro ............. 303
37. The IOB Chain .................................................. 307
38. Sample &IHADFAPELS Program ..................................... 324
39. Example of Determining Symbol Definition .......................... 324
40. Example of an IGWASYS Call Statement ........................... 345
41. Example of an IGWASMRT Call LINK Statement .................... 347
42. Example of IGGABWO Using LOAD and CALL Statements .......... 350
43. Example of the IGWARLS Query Call Using LOAD and CALL Statements .......... 358
44. Exit Routine Call Sequence ........................................ 366
45. PUT Return and Reason Codes ..................................... 387
46. Establishing and Deleting a Task Level
   DESERV Exit Part 1 of 2 ..................................... 399
47. Establishing and Deleting a Task Level
   DESERV Exit Part 2 of 2 ..................................... 400
48. Sample DESERV Exit Routine Part 1 of 3 ........................... 401
49. Sample DESERV Exit Routine Part 2 of 3 ........................... 402
50. Sample DESERV Exit Routine Part 3 of 3 ........................... 403
51. Code to Add a 1403 UCS Image to SYS1.IMAGELIB .............. 459
52. Code to Add a 3210 UCS Image to SYS1.IMAGELIB .................. 460
53. Sample Code to Add a 3211 UCS Image to SYS1.IMAGELIB .......... 461
54. UCS Image Table Entry Format .................................... 463
55. Adding a New Band ID to the 4245 UCS Image Table (UCS5) .......... 469
56. Adding a New Default Entry to the 4248 UCS Image Table (UCS8) .......... 470
57. Format of the Standard STD1 FCB Image .......................... 472
58. Format of the Standard STD2 FCB Image .......................... 473
59. Sample Code to Assemble and Add an FCB Load Module to SYS1.IMAGELIB .......... 475
## Tables

1. Referenced Publications ........................................... xi
2. DSCB Format-1 or Format-8 ........................................ 6
3. DSCB Format-3 ..................................................... 11
4. DSCB Format-4 ..................................................... 12
5. DSCB Format-5 ..................................................... 14
6. Format-9 DSCB ..................................................... 15
7. Format of the LSPACE Parameter List (MF=D) .............. 30
8. DADSM LSPACE Message Return Area Contents ............ 33
9. LSPACE Data Return Area Format ............................... 34
10. DADSM OBTAIN Return Codes .................................. 38
11. DADSM OBTAIN Return Codes .................................. 41
12. DADSM PARTREL Return Codes ................................. 47
13. REALLOC Parameter List .......................................... 53
14. DADSM CREATE Return Codes .................................. 55
15. CVAF Parameter List - ICVAFPL ............................... 58
16. CVFCTN Field of CVPL—Contents and Definitions .......... 60
17. Format of a Buffer List Header .................................. 61
18. Format of a Buffer List Entry ................................... 62
19. Format of ICVEDT02 Mapping Macro .......................... 62
20. Format of a Filter Criteria List Header ......................... 69
21. Format of a Filter Criteria List Entry ......................... 70
22. SCRATCH Parameter List ......................................... 148
23. SCRATCH Return Codes .......................................... 149
24. SCRATCH Status Codes .......................................... 150
25. Secondary Status Codes ......................................... 150
26. RENAME Parameter List generated by CAMLIST ............ 154
27. DADSM RENAME Return Codes .................................. 155
28. RENAME Status Codes .......................................... 155
29. LOCATE Return Codes .......................................... 162
30. CATALOG Return Codes .......................................... 167
31. Summary of the differences between EXCP, EXCPVR, and EXCP V=R .................................................. 170
32. Storage area locations for CCW channel program components .................................................. 174
33. Storage area locations for zHPF channel program components .................................................. 176
34. Comparing CCW and zHPF channel programs .................. 176
35. Maximum Number of IDAWs for CCW byte-counts ............ 180
36. DCBOFLGS Usage ................................................ 194
37. Bits that EXCP uses in DCBIFLGS after the DCB is OPEN .................................................. 199
38. DCB bits to signify presence of DCBE .......................... 203
39. DCB DEVD options ............................................. 203
40. IEDB Structure Mapping .......................................... 210
41. IOBE Structure Mapping .......................................... 217
42. EXCP Appendages ............................................... 222
43. Entry Points, Returns, and Available Work Registers for Appendages .................................................. 223
44. Registers and Their Use for Converting Relative to Actual .................................................. 233
45. Relative to Actual Conversion Routine Return Codes ........ 234
46. Registers and Their Use for Converting Actual to Relative .................................................. 235
47. Actual to Relative Conversion Routine Return Codes ........ 235
48. Registers and Their Use for A Sector Convert Routine .................................................. 237
49. ........................................................... 242
50. Channel Program Command Words Used by XDAP ............ 244
51. PROTECT Return Codes .......................................... 257
52. Minimum size of area ............................................ 265
53. INFO=AMCAP 32-byte return data ............................ 271
54. Optimum and Maximum Block Size Supported When Using EXCP or the Access Method Large Block Interface .................................................. 271
55. Simulated Device Characteristics Information ............... 273
56. Simulated Device Characteristics Information ............... 274
57. Output from DEVTYPE Macro .................................. 276
58. Output from DEVTYPE Macro — DASD Devices ............. 277
59. Return Codes from the RDJFCB Macro ......................... 289
60. ........................................................... 290
61. Format of the Type 13 JFCB Exit List Entry ................. 293
62. Format of the Allocation Retrieval List (mapped by the IHAARL macro) .................................................. 294
63. Format of the Allocation Retrieval Area (mapped by the IHAARA macro) .................................................. 295
64. Backup-While-Open Indicators .................................. 353
65. IGWASYS, IGWASMS, IGWABWO, IGWLSHR Return and Reason Codes .................................................. 354
66. IGWARLS Return and Reason Codes ......................... 357
67. Installing or Replacing the DESERV Exit ..................... 369
68. DESERV Screen Table Structure ................................ 370
69. Deleting the DESERV Exit ....................................... 370
70. Determining If a DESERV Exit Is Active ....................... 371
71. DESX Structure Mapping DESERV Exit Parameter List ........ 371
72. Structure of DESP for DESERV GET Invocations .......... 373
73. DESL Structure ............................................... 375
74. DESN Parameter List .......................................... 375
75. DESB Parameter List .......................................... 376
76. SMDE Format ................................................ 376
77. Directory Entry Name Section .................................. 378
78. Directory Entry Notelist Section (PDS Only) ............... 378
79. Directory Entry Token Section ................................ 378
80. Directory Entry Primary Name Section ....................... 379
81. Directory Entry Name Section ................................ 379
82. LSLoader Attributes Unique to Program Objects ............ 381
83. Attributes Unique to Load Modules (PDS only) ............. 383
84. Alias in Unformatted Form ..................................... 383
85. DESERV PUT DESP Fields ..................................... 384
<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>86</td>
<td>DESD Parameter List</td>
<td>388</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>87</td>
<td>DESERV DELETE DESP Fields</td>
<td>389</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>88</td>
<td>DESERV RENAME DESP Fields</td>
<td>390</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>89</td>
<td>DESERV UPDATE DESP Fields</td>
<td>392</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>90</td>
<td>Return and Reason Codes for the Exit DESERV Function</td>
<td>396</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>91</td>
<td>BPX1PCT - Return Codes and Reason Codes</td>
<td>415</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>92</td>
<td>Structure for the DisplayBufferLimits and ChangeBufferLimits Commands (GFUMPCTL)</td>
<td>416</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>93</td>
<td>Structure for the DisplayGlobalStats Command (GFUMPCTL)</td>
<td>417</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>94</td>
<td>Structure for the DisplayFSStats Command (GFUMPCTL)</td>
<td>418</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>95</td>
<td>Structure for the ExtendFS Command (GFUMPCTL)</td>
<td>419</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>96</td>
<td>Constants for Extend Units Supported</td>
<td>419</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>97</td>
<td>Partial Listing of DEB Fields</td>
<td>443</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>98</td>
<td>DFA Fields</td>
<td>449</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>99</td>
<td>DFA Element Name</td>
<td>452</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>SYS1.IMAGELIB Contents</td>
<td>456</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>UCS5 Image Table Contents</td>
<td>463</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>102</td>
<td>UCS6 Image Table Contents</td>
<td>464</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>103</td>
<td>3262 Model 5 Print Bands</td>
<td>465</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
About This Document

This document 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.

For information about the accessibility features of z/OS®, for users who have a physical disability, see Appendix D, “Accessibility,” on page 495.

Required product knowledge

To use this document 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 z/OS DFSMSdfp Storage Administration SC23-6860.

Referenced documents

The following publications are referenced in this document:

<table>
<thead>
<tr>
<th>Publication Title</th>
<th>Order Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM High Level Assembler/MVS &amp; VM &amp; VSE Programmer’s Guide</td>
<td>SC26-4941</td>
</tr>
<tr>
<td>IBM High Level Assembler/MVS &amp; VM &amp; VSE Language Reference</td>
<td>SC26-4940</td>
</tr>
<tr>
<td>z/OS MVS Programming: Assembler Services Guide</td>
<td>SA23-1368</td>
</tr>
<tr>
<td>z/OS MVS Programming: Authorized Assembler Services Reference ABE-HSF</td>
<td>SA23-1369</td>
</tr>
<tr>
<td>z/OS MVS Programming: Authorized Assembler Services Guide</td>
<td>SA23-1371</td>
</tr>
<tr>
<td>z/OS MVS Programming: Authorized Assembler Services Reference ALE-DYN</td>
<td>SA23-1372</td>
</tr>
<tr>
<td>z/OS MVS Programming: Authorized Assembler Services Reference EDF-IXG</td>
<td>SA23-1373</td>
</tr>
<tr>
<td>z/OS MVS Programming: Authorized Assembler Services Reference LLA-SDL</td>
<td>SA23-1374</td>
</tr>
<tr>
<td>z/OS MVS Programming: Authorized Assembler Services Reference SET-WTO</td>
<td>SA23-1375</td>
</tr>
<tr>
<td>z/OS MVS Planning: Global Resource Serialization</td>
<td>SA23-1389</td>
</tr>
<tr>
<td>z/OS MVS Program Management: User’s Guide and Reference</td>
<td>SA23-1393</td>
</tr>
<tr>
<td>z/OS MVS Program Management: Advanced Facilities</td>
<td>SA23-1392</td>
</tr>
<tr>
<td>z/Architecture Principles of Operation</td>
<td>SA22-7832</td>
</tr>
<tr>
<td>Publication Title</td>
<td>Order Number</td>
</tr>
<tr>
<td>----------------------------------------------------------------------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>IBM 2821 Control Unit Component Description</td>
<td>GA24-3312</td>
</tr>
<tr>
<td>IBM 3203 Printer Component Description and Operator’s Guide</td>
<td>GA33-1515</td>
</tr>
<tr>
<td>IBM 3211 Printer, 3216 Interchangeable Train Cartridge, and 3811 Printer Control Unit Component Description and Operator’s Guide</td>
<td>GA24-3543</td>
</tr>
<tr>
<td>IBM 3262 Model 5 Printer Product Description</td>
<td>GA24-3936</td>
</tr>
<tr>
<td>IBM 3800 Printing Subsystem Programmer’s Guide</td>
<td>GC26-3846</td>
</tr>
<tr>
<td>IBM 3800 Printing Subsystem Model 3 Programmer’s Guide: Compatibility</td>
<td>SH35-0051</td>
</tr>
<tr>
<td>IBM 3800 Printing Subsystem Models 3 and 8 Programmer’s Guide</td>
<td>SH35-0061</td>
</tr>
<tr>
<td>3900 Advanced Function Printer Product Description</td>
<td>GA32-0135</td>
</tr>
<tr>
<td>IBM 4245 Printer Model 1 Component Description and Operator’s Guide</td>
<td>GA33-1541</td>
</tr>
<tr>
<td>IBM 4248 Printer Model 1 Description</td>
<td>GA24-3927</td>
</tr>
<tr>
<td>IBM 6262 Printer Model 014 Product Description</td>
<td>GA24-4134</td>
</tr>
<tr>
<td>IBM 6262 Printer Model 014 User’s Guide</td>
<td>GA24-4132</td>
</tr>
<tr>
<td>IBM 6262 Printer Print Band Manual</td>
<td>GA24-4131</td>
</tr>
<tr>
<td>z/OS MVS JCL Reference</td>
<td>SA23-1385</td>
</tr>
<tr>
<td>z/OS MVS JCL User’s Guide</td>
<td>SA23-1386</td>
</tr>
<tr>
<td>z/OS JES2 Initialization and Tuning Guide</td>
<td>SA32-0991</td>
</tr>
<tr>
<td>z/OS JES3 Initialization and Tuning Guide</td>
<td>SA32-1003</td>
</tr>
<tr>
<td>z/OS DFSMS Access Method Services Commands</td>
<td>SC26-7326</td>
</tr>
<tr>
<td>z/OS DFSMSdps Diagnosis</td>
<td>SC23-6863</td>
</tr>
<tr>
<td>z/OS DFSMS Installation Exits</td>
<td>SC23-6850</td>
</tr>
<tr>
<td>z/OS DFSMS Macro Instructions for Data Sets</td>
<td>SC23-6852</td>
</tr>
<tr>
<td>z/OS DFSMS Using Data Sets</td>
<td>SC23-6855</td>
</tr>
<tr>
<td>z/OS DFSMS Using Magnetic Tapes</td>
<td>SC23-6858</td>
</tr>
<tr>
<td>z/OS DFSMSdps Utilities</td>
<td>SC23-6864</td>
</tr>
<tr>
<td>MVS/ESA SML: Managing Data</td>
<td>SC26-3124</td>
</tr>
<tr>
<td>z/OS DFSMS Implementing System-Managed Storage</td>
<td>SC23-6849</td>
</tr>
<tr>
<td>z/OS MVS Initialization and Tuning Guide</td>
<td>SA23-1379</td>
</tr>
<tr>
<td>z/OS HCD Planning</td>
<td>GA32-0907</td>
</tr>
<tr>
<td>z/OS MVS System Messages, Vol 1 (ABA-AOM)</td>
<td>SA38-0668</td>
</tr>
<tr>
<td>z/OS MVS System Messages, Vol 2 (ARC-ASA)</td>
<td>SA38-0669</td>
</tr>
<tr>
<td>z/OS MVS System Messages, Vol 3 (ASB-BPX)</td>
<td>SA38-0670</td>
</tr>
<tr>
<td>z/OS MVS System Messages, Vol 4 (CBD-DMO)</td>
<td>SA38-0671</td>
</tr>
<tr>
<td>z/OS MVS System Messages, Vol 5 (EDG-GFS)</td>
<td>SA38-0672</td>
</tr>
<tr>
<td>z/OS MVS System Messages, Vol 6 (GOS-IEA)</td>
<td>SA38-0673</td>
</tr>
<tr>
<td>z/OS MVS System Messages, Vol 7 (IEB-JEI)</td>
<td>SA38-0674</td>
</tr>
<tr>
<td>z/OS MVS System Messages, Vol 8 (IEF-JGD)</td>
<td>SA38-0675</td>
</tr>
</tbody>
</table>
Table 1. Referenced Publications (continued)

<table>
<thead>
<tr>
<th>Publication Title</th>
<th>Order Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>z/OS MVS System Messages, Vol 9 (IGF-IWM)</td>
<td>SA38-0676</td>
</tr>
<tr>
<td>z/OS MVS System Messages, Vol 10 (IXC-IZP)</td>
<td>SA38-0677</td>
</tr>
<tr>
<td>z/OS TSO/E Command Reference</td>
<td>SA32-0975</td>
</tr>
</tbody>
</table>

**z/OS information**

This information explains how z/OS references information in other documents and on the web.

When possible, this information uses cross document links that go directly to the topic in reference using shortened versions of the document title. For complete titles and order numbers of the documents for all products that are part of z/OS, see [z/OS Information Roadmap](https://www.ibm.com/systems/z/os/zos/bkserv/).

To find the complete z/OS library, including the z/OS Information Center, go to the [z/OS Internet library](http://www.ibm.com/systems/z/os/zos/bkserv/)

**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.

  ```plaintext
  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.

  ```plaintext
  First Line
  Second Line
  Last Line
  ```

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

  ```plaintext
  REQUIRED_KEYWORD
  ```

If several mutually exclusive required keywords or values exist, they are stacked vertically in alphanumeric order.
- Optional keywords and values appear below the main path line. You can choose not to code optional keywords and values.

- 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.

- 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.

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

- 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)**.

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

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

Notes:
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.

Syntax Fragment:

Address and Register Conventions

The notation used to code an operand appears in the following format:

**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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Operation</th>
<th>Operand</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALPHA1</td>
<td>L</td>
<td>1,39(4,10)</td>
</tr>
<tr>
<td>ALPHA2</td>
<td>L</td>
<td>REG1,39(4,TEN)</td>
</tr>
<tr>
<td>BETA1</td>
<td>L</td>
<td>2,ZETA(4)</td>
</tr>
<tr>
<td>BETA2</td>
<td>L</td>
<td>REG2,ZETA(REG4)</td>
</tr>
<tr>
<td>GAMMA1</td>
<td>L</td>
<td>2,ZETA</td>
</tr>
<tr>
<td>GAMMA2</td>
<td>L</td>
<td>REG2,ZETA</td>
</tr>
<tr>
<td>GAMMA3</td>
<td>L</td>
<td>2,=F'1000'</td>
</tr>
<tr>
<td>LAMBDA1</td>
<td>L</td>
<td>3,20(5)</td>
</tr>
</tbody>
</table>

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.
How to send your comments to IBM

We appreciate your input on this publication. Feel free to comment on the clarity, accuracy, and completeness of the information or provide any other feedback that you have.

Use one of the following methods to send your comments:
1. Send an email to mhvrdfs@us.ibm.com.
2. Send an email from the "Contact us" web page for z/OS (http://www.ibm.com/systems/z/os/zos/webqs.html).
3. Mail the comments to the following address:
   IBM Corporation
   Attention: MHVRCS Reader Comments
   Department H6MA, Building 707
   2455 South Road
   Poughkeepsie, NY 12601-5400
   US
4. Fax the comments to us, as follows:
   From the United States and Canada: 1+845+432-9405
   From all other countries: Your international access code +1+845+432-9405

Include the following information:
• Your name and address.
• Your email address.
• Your telephone or fax number.
• The publication title and order number:
  z/OS V2R1.0 DFSMSdftp Advanced Services
  SC23-6861-01
• The topic and page number that is related to your comment.
• The text of your comment.

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

IBM or any other organizations use the personal information that you supply to contact you only about the issues that you submit.

If you have a technical problem

Do not use the feedback methods that are listed for sending comments. Instead, take one of the following actions:
• Contact your IBM service representative.
• Call IBM technical support.
• Visit the IBM Support Portal at z/OS support page (http://www.ibm.com/systems/z/support/)
Summary of changes for z/OS Version 2 Release 1 (V2R1) as updated March 2014

The following changes are made for z/OS Version 2 Release 1 (V2R1) as updated March 2014. In this revision, all technical changes for z/OS V2R1 are indicated by a vertical line to the left of the change.

**New**

New fields to indicate zEDC compression are added to “Data Facilities Area (DFA) Fields” on page 449.

**Note:** For more information on the zEDC compression enhancements, see [z/OS DFSMS Using the New Functions](#).

**z/OS Version 2 Release 1 summary of changes**

See the following publications for all enhancements to z/OS Version 2 Release 1 (V2R1):

- [z/OS Migration](#)
- [z/OS Planning for Installation](#)
- [z/OS Summary of Message and Interface Changes](#)
- [z/OS Introduction and Release Guide](#)
Chapter 1. Using the Volume Table of Contents

This information is intended to help you to use system macros to access and modify the volume table of contents (VTOC) and VTOC index. The direct access device storage management (DADSM) routines control space allocation on direct access volumes through the VTOC for a volume and through the VTOC index if one exists. A storage administrator uses the ICKDSF utility to build these structures. See "Creating the VTOC and VTOC Index" on page 143.

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). The volume label is described in z/OS DFSMS Using Data Sets. A VTOC consists of complete tracks. Some other System z® operating systems support VTOCs that are compatible with z/OS. Some of them allow the VTOC to begin in the middle of a track. Normal z/OS operations do not support such a VTOC. However if you do not create or extend data sets from z/OS in such a VTOC, you may be able to read data sets on z/OS that were created on the other operating system.

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 currently residing on the volume, or to contiguous, unassigned tracks on the volume.

A special type of data is in a hierarchical file system (HFS) 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. Certain linear data sets contain z/OS file system (zFS) file systems. They also meet these standards. Each data set containing an HFS or zFS file system can contain UNIX files and directories belonging to multiple users. You can access the files through z/OS UNIX System Services, BSAM, QSAM, and VSAM. See z/OS DFSMS Using Data Sets. DSCBs for data sets describe their characteristics 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 Table 2 on page 6 through Table 5 on page 14.

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.

One or more DSCBs in the VTOC define a data set on each volume on which the data set resides. The number of DSCBs needed to define a data set is determined by the number of extents that the data set occupies and by whether it has a format 9 DSCB. One or more format-3 DSCBs are required for data sets with more than three extents.

![Figure 2 on page 3](image) shows a VTOC and the DSCBs needed to define four data sets. Data set A has 29 extents, so it cannot be a basic format, direct or partitioned (PDS) data set. Because it has so many extents, it requires a format-1 DSCB and two format-3 DSCBs. Data set B has 16 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. Data sets B, C, and D could be any type of data set.

---

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.
Figure 3 on page 4 shows a VTOC and the DSCBs needed to define four data sets on an extended address volume. Data set W (in Figure 3 on page 4) has 29 extents, so it cannot be a basic format, direct or partitioned (PDS) data set. Because it has so many extents, it requires a format-1 DSCB and two format-3 DSCBs. Data set X has 16 extents and since it is an EAS eligible data set it requires a format-8 and a format-9 DSCB, plus an additional format-3 DSCB. Data sets Y and Z have three or fewer extents and need only a format-1 DSCB. Data sets X, Y, and Z could be any type of data set.

<table>
<thead>
<tr>
<th>0(0)</th>
<th>1(1)</th>
<th>2(2)</th>
<th>3(3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IOBFLAG1</td>
<td>IOBFLAG2</td>
<td>IOBSENS0</td>
<td>IOBSENS1</td>
</tr>
<tr>
<td>4(4)</td>
<td>1OBECBCC</td>
<td>IOBECBPB</td>
<td></td>
</tr>
<tr>
<td>8(8)</td>
<td></td>
<td>IOBSTRTB</td>
<td></td>
</tr>
<tr>
<td>16(10)</td>
<td>IOBSIOCC</td>
<td>IOBDCBP</td>
<td></td>
</tr>
<tr>
<td>20(14)</td>
<td>IOBFLAG4</td>
<td>IOBDCBP</td>
<td></td>
</tr>
<tr>
<td>24(18)</td>
<td>IOBRESTR</td>
<td>IOBRESTR+1</td>
<td></td>
</tr>
<tr>
<td>28(1C)</td>
<td>IOBINCAM</td>
<td>IOBERRCT</td>
<td></td>
</tr>
<tr>
<td>32(20)</td>
<td>IOBSEEK (first byte, M)</td>
<td>Direct Access, Teleprocessing, and Graphic Devices</td>
<td></td>
</tr>
<tr>
<td>33(21)</td>
<td>IOBSEEK (second through eight bytes, BBCCHHR)</td>
<td>Direct Access Storage Devices (DASD)</td>
<td></td>
</tr>
<tr>
<td>39(27)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 2. Contents of VTOC - DSCBs Describing Data Sets on Volume That Has No VTOC Index
Using the VTOC

The mapping macro for the format-1, format-2, format-3, format-4, format-5, format-8, and format-9 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.
Using the VTOC

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'.

Format-1 and Format-8 DSCBs

Note: The fields in the format 1 DSCB and the format 8 DSCB are almost identical. The small differences are noted in the field descriptions.

Name: Identifier

Function: Describes the first three extents and other information about a data set.

How Many: One for each data set on the volume, except the VTOC.

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

Determining the Type of Data Set: To learn the type of a data set, test the DS1DSORG field. Except for the DS1DSGU bit, only one bit should be on. If no bit is on, the data set is not standard and the system generally treats it like DSORG=PS. The explanation for all DS1DSORG bits being off might be:

- The user did not specify the DSORG option and no program has written data in the data set.
- The data set was created by a program that used EXCP or EXCPVR.
- The data set was created on another operating system.

All of the above conditions are normal. Normally after a program writes data, the DS1DSORG field will have a bit on. z/OS currently supports the following types of data set:

- Physical sequential (DSORG=PS). There are three types:
  - Large format if the DS1LARGE bit is on.
  - Extended format if the DS1STRP bit is on. An indicator in the catalog entry specifies whether it is striped.
  - Basic format if neither of the above two bits is on.
- Direct (DSORG=DA).
- Partitioned (DSORG=PO). This can signify one of three types of data set:
  - PDS (partitioned data set) if the DS1PDSE and DS1PDSEX bits are off.
  - PDSE (partitioned data set extended) if the DS1PDSE bit is on and DS1PDSEX bit is off.
  - HFS (hierarchical file system) if the DS1PDSE and DS1PDSEX bits are on.

Other combinations of these three bits are invalid.
- VSAM (AMORG). You can use CSI, catalog search interface, to determine the type of VSAM data set. If it is a linear data set, it might contain a DB2® tablespace or a z/OS file system (zFS) or something else.

Table 2 on page 6 shows the contents of a format-1 or format-8 DSCB.

The DS1REFD field:
1. For a VSAM data set, 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.
Using the VTOC

2. For a multivolume VSAM data set, OPEN updates the DS1REFD field only for the first volume of the data component of the base cluster.

3. For a non-VSAM multivolume data set that is not SMS managed, OPEN updates the DS1REFD field on the first volume OPEN selected to use.

4. For a non-VSAM multivolume SMS-managed data set that is not extended format, OPEN updates the DS1REFD field on the first volume OPEN selected to use, as well as the first volume of the data set.

5. For an extended format single-striped non-VSAM data set, OPEN updates the DS1REFD field on the first volume OPEN selected to use, as well as the first volume of the data set.

6. For an extended format multi-striped non-VSAM data set, OPEN updates the DS1REFD fields on all volumes of the data set.

Table 2. DSCB Format-1 or Format-8

<table>
<thead>
<tr>
<th>Offset Dec (Hex)</th>
<th>Type or Bit Mask</th>
<th>Length</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0(X'0')</td>
<td>Character</td>
<td>44</td>
<td>DS1DSNAM</td>
<td>Data set name.</td>
</tr>
<tr>
<td>44(X'2C')</td>
<td>Character</td>
<td>1</td>
<td>DS1FMTID</td>
<td>Format Identifier.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DS1IDC</td>
<td>X'F1'. This is a format-1 DSCB.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DS8IDC</td>
<td>X'F8'. This is a format-8 DSCB.</td>
</tr>
<tr>
<td>45(X'2D')</td>
<td>Character</td>
<td>6</td>
<td>DS1DSSN</td>
<td>Data set serial number (identifies the first or only volume containing the data set/space).</td>
</tr>
<tr>
<td>51(X'33')</td>
<td>Unsigned</td>
<td>2</td>
<td>DS1VOLSQ</td>
<td>Volume sequence number.</td>
</tr>
<tr>
<td>53(X'35')</td>
<td>Character</td>
<td>3</td>
<td>DS1CREDT</td>
<td>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.</td>
</tr>
<tr>
<td>56(X'38')</td>
<td>Character</td>
<td>3</td>
<td>DS1EXPDT</td>
<td>Expiration date ('YDD'), discontinuous binary. Add 1900 and the value in the Y byte to determine the year.</td>
</tr>
<tr>
<td>59(X'3B')</td>
<td>Unsigned</td>
<td>1</td>
<td>DS1NOEPV</td>
<td>Number of extents on volume.</td>
</tr>
<tr>
<td>60(X'3C')</td>
<td>Unsigned</td>
<td>1</td>
<td>DS1NOBDDB</td>
<td>Number of bytes used in last directory block.</td>
</tr>
<tr>
<td>61(X'3D')</td>
<td>Bitstring</td>
<td>1</td>
<td>DS1FLAG1</td>
<td>Flags byte</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DS1COMPR</td>
<td>Compressible format data set (DS1STRP is also 1).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DS1CPOIT</td>
<td>Checkpointed data set.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DS1EXPBY</td>
<td>VSE expiration date specified by retention period (not currently used in z/OS)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>DS1RECAL</td>
<td>Data set recalled.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>DS1LARGE</td>
<td>Large format data set.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11</td>
<td>DS1EATTR</td>
<td>Extended attribute setting as specified on the allocation request.(EATTR=)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• If 0, EATTR has not been specified. For VSAM data sets, the default behavior is equivalent to EATTR=OPT. For non-VSAM data sets, the default behavior is equivalent to EATTR=NO.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• If 1, EATTR=NO has been specified. The data set cannot have extended attributes (format 8 and 9 DSCBs) or optionally reside in EAS.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• If 2, EATTR=OPT has been specified. The data set can have extended attributes and optionally reside in EAS. This is the default behavior for VSAM data sets.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>13</td>
<td>DS1SYSCD</td>
<td>System code.</td>
</tr>
</tbody>
</table>
Table 2. DSCB Format-1 or Format-8 (continued)

<table>
<thead>
<tr>
<th>Offset Dec (Hex)</th>
<th>Type or Bit Mask</th>
<th>Length</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>75(X'4B')</td>
<td>Character</td>
<td>3</td>
<td>DS1REFD</td>
<td>Date last referenced ('YDD' or zero, if not maintained). Add 1900 and the value in the Y byte to determine the year.</td>
</tr>
<tr>
<td>78(X'4E')</td>
<td>Bitstring</td>
<td>1</td>
<td>DS1SMSFG</td>
<td>System managed storage indicators.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DS1SMSDS</td>
<td>System managed data set. IEHLIST displays this bit as the letter “S”.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DS1SMSUC</td>
<td>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”.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DS1REBLK</td>
<td>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”.</td>
</tr>
<tr>
<td>79(X'4F')</td>
<td>Bitstring</td>
<td>1</td>
<td>DS1CRSDB</td>
<td>DADSM created original block size and data set has not been opened for output. IEHLIST displays this bit as the letter “B”.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DS1PDSE</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DS1STRP</td>
<td>Sequential extended-format data set. IEHLIST displays this bit as the letter “E”. IEHLIST displays a “?” when it finds an invalid combination of bits.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DS1PDSEX</td>
<td>HFS data set (DS1PDSE must also be 1) IEHLIST displays this bit as the letter “H”.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DS1DSAE</td>
<td>Extended attributes exist in the catalog entry.</td>
</tr>
<tr>
<td>80(X'50')</td>
<td>Unsigned</td>
<td>2</td>
<td>DS1SCXTV</td>
<td>Secondary space extension value for average record length or average block length.</td>
</tr>
<tr>
<td>82(X'52')</td>
<td>Bitstring</td>
<td>2</td>
<td>DS1DSORG</td>
<td>Data set organization.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DS1DSGIS</td>
<td>Indexed sequential organization.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DS1DSGPS</td>
<td>Physical sequential organization.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DS1DSGDA</td>
<td>Direct organization.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DS1DSGCX</td>
<td>BTAM or QTAM line group.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DS1DSGPO</td>
<td>Partitioned organization.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DS1DSGU</td>
<td>Unmovable; the data contains location dependent information.</td>
</tr>
</tbody>
</table>
### Using the VTOC

**Table 2. DSCB Format-1 or Format-8 (continued).** DSCB Format-1 or Format-8

<table>
<thead>
<tr>
<th>Offset Dec (Hex)</th>
<th>Type or Bit Mask</th>
<th>Length</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Second byte of</td>
<td>DS1DSORG</td>
</tr>
<tr>
<td>100x 00xx</td>
<td></td>
<td></td>
<td>DS1DSGGS</td>
<td>Graphics organization.</td>
</tr>
<tr>
<td>010x 00xx</td>
<td></td>
<td></td>
<td>DS1DSGTX</td>
<td>TCAM line group (not supported)</td>
</tr>
<tr>
<td>001x 00xx</td>
<td></td>
<td></td>
<td>DS1DSGTQ</td>
<td>TCAM message queue (not supported).</td>
</tr>
<tr>
<td>000x 10xx</td>
<td></td>
<td></td>
<td>DS1ACBM</td>
<td>VSAM data set/space.</td>
</tr>
<tr>
<td>000x 10xx</td>
<td></td>
<td></td>
<td>DS1ORGAM</td>
<td>VSAM data set/space.</td>
</tr>
<tr>
<td>000x 01xx</td>
<td></td>
<td></td>
<td>DS1DSGTR</td>
<td>TCAM 3705 (not supported).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Reserved.</td>
</tr>
<tr>
<td>84('54')</td>
<td>Character 1</td>
<td>1</td>
<td>DS1RECFM</td>
<td>Record format.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10</td>
<td>DS1RECFF</td>
<td>Fixed length.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>01</td>
<td>DS1RECFV</td>
<td>Variable length.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11</td>
<td>DS1RECFU</td>
<td>Undefined length.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.1</td>
<td>DS1RECFT</td>
<td>Track overflow. No longer supported by current</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>hardware.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.1</td>
<td>DS1RECFB</td>
<td>Blocked; cannot occur with undefined.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.1</td>
<td>DS1RECFS</td>
<td>Fixed length: standard blocks; no truncated blocks</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>or unfilled tracks except possible the last block</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>and track. Variable length: spanned records.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.00</td>
<td>No control character.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>.10</td>
<td>DS1RECFA</td>
<td>ISO/ANSI control character.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.01</td>
<td>DS1RECMC</td>
<td>Machine control character.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.11</td>
<td>Reserved.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>.x</td>
<td>Reserved.</td>
<td></td>
</tr>
<tr>
<td>85('55')</td>
<td>Character 1</td>
<td>1</td>
<td>DS1OPTCD</td>
<td>Option Code.</td>
</tr>
</tbody>
</table>

**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).
Table 2. DSCB Format-1 or Format-8 (continued). DSCB Format-1 or Format-8

<table>
<thead>
<tr>
<th>Offset Dec (Hex)</th>
<th>Type or Bit Mask</th>
<th>Length</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Search direct.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>User label totaling.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Each record contains a table reference character.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>85(X'55')</td>
<td>Bitstring</td>
<td>1</td>
<td>DS1OPTAM</td>
<td>VSAM OPTCD settings.</td>
</tr>
<tr>
<td>VSAM OPTCD field assignments (applies only if DS1ORGAM is on):</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Reserved.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>. 1</td>
<td>DS1OPTBC</td>
<td></td>
<td></td>
<td>Data set is an integrated catalog facility catalog.</td>
</tr>
<tr>
<td>... xx xxxx</td>
<td>Reserved.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>86(X'56')</td>
<td>Binary</td>
<td>2</td>
<td>DS1BLKL</td>
<td>Block length (Type F unblocked records), or maximum block size (F blocked, U or V records).</td>
</tr>
<tr>
<td>88(X'58')</td>
<td>Binary</td>
<td>2</td>
<td>DS1LRECL</td>
<td>Logical record length: Fixed length-record length, Undefined length-zero, Variable unspanned-maximum record length, Variable spanned and &lt; 32757 bytes-maximum record length, Variable spanned and &gt; 32756 bytes-X'8000'.</td>
</tr>
<tr>
<td>90(X'5A')</td>
<td>Binary</td>
<td>1</td>
<td>DS1KEYL</td>
<td>Key length (0 to 255).</td>
</tr>
<tr>
<td>91(X'5B')</td>
<td>Binary</td>
<td>2</td>
<td>DS1RKP</td>
<td>Relative key position.</td>
</tr>
<tr>
<td>93(X'5D')</td>
<td>Character</td>
<td>1</td>
<td>DS1DSIND</td>
<td>Data set indicators.</td>
</tr>
<tr>
<td>. 1</td>
<td>DS1IND80</td>
<td></td>
<td></td>
<td>Last volume containing data in this data set.</td>
</tr>
<tr>
<td>. 1</td>
<td>DS1IND40</td>
<td></td>
<td></td>
<td>Data set is RACF™, a component of the Security Server for z/OS, defined with a discrete profile.</td>
</tr>
<tr>
<td>... 1</td>
<td>DS1IND20</td>
<td></td>
<td></td>
<td>Block length is a multiple of 8 bytes.</td>
</tr>
<tr>
<td>... 1</td>
<td>DS1IND10</td>
<td></td>
<td></td>
<td>Password is required to read or write, or both; see DS1IND04.</td>
</tr>
<tr>
<td>... 1</td>
<td>DS1IND08</td>
<td></td>
<td></td>
<td>Data set has been modified since last recall.</td>
</tr>
<tr>
<td>... 1</td>
<td>DS1IND04</td>
<td></td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>... 1</td>
<td>DS1IND02</td>
<td></td>
<td></td>
<td>Data set opened for other than input since last backup copy made.</td>
</tr>
<tr>
<td>94(X'5E')</td>
<td>Character</td>
<td>4</td>
<td>DS1SCALO</td>
<td>Secondary allocation space parameters.</td>
</tr>
<tr>
<td>94(X'5E')</td>
<td>Character</td>
<td>1</td>
<td>DS1SCAL1</td>
<td>Flag byte.</td>
</tr>
<tr>
<td>xxxx xxxx</td>
<td>DS1DSPAC</td>
<td></td>
<td></td>
<td>Space request bits, defined as follows:</td>
</tr>
<tr>
<td>00 . 0</td>
<td>DS1DSABS</td>
<td></td>
<td></td>
<td>Absolute track request.</td>
</tr>
<tr>
<td>00 . 1</td>
<td>DS1EXT</td>
<td></td>
<td></td>
<td>Extension to secondary space exists (see DS1SCEXT at offset 79 (X'4F')).</td>
</tr>
<tr>
<td>11</td>
<td>DS1CYL</td>
<td></td>
<td></td>
<td>Cylinder request.</td>
</tr>
<tr>
<td>10</td>
<td>DS1TRK</td>
<td></td>
<td></td>
<td>Track request.</td>
</tr>
<tr>
<td>01 . x</td>
<td>DS1AVR</td>
<td></td>
<td></td>
<td>Average block length request.</td>
</tr>
<tr>
<td>01 . 1</td>
<td>DS1AVRND</td>
<td></td>
<td></td>
<td>Average block and round request.</td>
</tr>
<tr>
<td>... 1</td>
<td>DS1MSGP</td>
<td></td>
<td></td>
<td>Mass storage vol group (MSVGP - no longer supported).</td>
</tr>
<tr>
<td>... 1</td>
<td>DS1CONTG</td>
<td></td>
<td></td>
<td>Contiguous request.</td>
</tr>
<tr>
<td>... 1</td>
<td>DS1MXIG</td>
<td></td>
<td></td>
<td>MXIG request.</td>
</tr>
<tr>
<td>... 1</td>
<td>DS1ALX</td>
<td></td>
<td></td>
<td>ALX request.</td>
</tr>
<tr>
<td>... 1</td>
<td>-</td>
<td></td>
<td></td>
<td>Round request.</td>
</tr>
<tr>
<td>95(X'5F')</td>
<td>Binary</td>
<td>3</td>
<td>DS1SCAL3</td>
<td>Secondary allocation quantity.</td>
</tr>
</tbody>
</table>
### Using the VTOC

#### Table 2. DSCB Format-1 or Format-8 (continued). DSCB Format-1 or Format-8

<table>
<thead>
<tr>
<th>Offset Dec (Hex)</th>
<th>Type or Bit Mask</th>
<th>Length</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>98(X'62')</td>
<td>Binary</td>
<td>3</td>
<td>DS1LSTAR</td>
<td>Last used track and block on track (TTR). Not defined for VSAM, PDSE, HFS and direct (BDAM). See bit DS1LARGE at +61 and byte DS1TTTHI at +104.</td>
</tr>
<tr>
<td>101(X'65')</td>
<td>Binary</td>
<td>2</td>
<td>DS1TRBAL</td>
<td>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.</td>
</tr>
<tr>
<td>103(X'67')</td>
<td>Character</td>
<td>1</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>104(X'68')</td>
<td>Character</td>
<td>1</td>
<td>DS1TTTHI</td>
<td>High order byte of track number in DS1LSTAR. Valid if DS1Large is on.</td>
</tr>
<tr>
<td>105(X'69')</td>
<td>Character</td>
<td>30</td>
<td>DS1EXNTS</td>
<td>Three extent fields.</td>
</tr>
<tr>
<td>105(X'69')</td>
<td>Character</td>
<td>10</td>
<td>DS1EXT1</td>
<td>First extent description.</td>
</tr>
<tr>
<td></td>
<td>Character</td>
<td>1</td>
<td>X'81'</td>
<td>Extent on cylinder boundaries.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>X'80'</td>
<td>Extent described is sharing cylinder (no longer supported).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>X'40'</td>
<td>First extent describes the user labels and is not counted in DS1NOEPV.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>X'04'</td>
<td>Index area extent (ISAM).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>X'02'</td>
<td>Overflow area extent (ISAM).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>X'01'</td>
<td>User's data block extent, or a prime area extent (ISAM).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>X'00'</td>
<td>This is not an extent.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td>Lower limit (CCHH). These are the bit definitions:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0-15</td>
<td>Low order 16 bits of the 28-bit cylinder number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>16-27</td>
<td>High order 12 bits of the 28-bit cylinder number. In a format-1 DSCB, these bits always are zero.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>28-31</td>
<td>Track number from 0 to 14. Use the TRKADDR macro or IECTRKAD routine when performing track address calculations.</td>
</tr>
<tr>
<td>115(X'73')</td>
<td>Character</td>
<td>10</td>
<td>DS1EXT2</td>
<td>Second extent description.</td>
</tr>
<tr>
<td>125(X'7D')</td>
<td>Character</td>
<td>10</td>
<td>DS1EXT3</td>
<td>Third extent description.</td>
</tr>
<tr>
<td>135(X'87')</td>
<td>Character</td>
<td>5</td>
<td>DS1PTRDS</td>
<td>In a format-1 DSCB this can be a pointer (CCHHR) to a format-2 or format-3 DSCB or be zero. In a format-8 DSCB this always is the CCHHR of a format-9 DSCB.</td>
</tr>
<tr>
<td>140(X'8C')</td>
<td>Character</td>
<td>DS1END</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

### Format-2 DSCB

This format applied only to ISAM data sets, which can no longer be created or opened.
Using the VTOC

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 10 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 are restricted to 16 extents per volume.

How Found: Chained from a format 1, format 2 or format 9 DSCB that represents the data set. In the case of a PDSE, HFS data set, sequential extended-format data set, or VSAM data set, the chain also can be from a preceding format-3 DSCB.

Table 3 shows the contents of a format-3 DSCB.

Table 3. DSCB Format-3

<table>
<thead>
<tr>
<th>Offset Dec (Hex)</th>
<th>Type</th>
<th>Length</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0(X'00')</td>
<td>Bitstring</td>
<td>4</td>
<td>-</td>
<td>Key identifier (X'03030303').</td>
</tr>
<tr>
<td>4(X'04')</td>
<td>Bitstring</td>
<td>40</td>
<td>DS3EXTNT</td>
<td>Four extent descriptions.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td>Extent type indicator. (See DS1EXT1 in Table 2 on page 9)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td>Extent sequence number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td></td>
<td>Lower limit (CCHH).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td></td>
<td>Upper limit (CCHH).</td>
</tr>
<tr>
<td>44(X'2C')</td>
<td>Character</td>
<td>1</td>
<td>DS3FMTID</td>
<td>Format identifier (XF3').</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DS3IDC</td>
<td>Constant value of XF3 in DS3FMTID.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DS3IDC</td>
<td>Constant value of X'F3' in DS3FMTID.</td>
</tr>
<tr>
<td>45(X'2D')</td>
<td>Bitstring</td>
<td>90</td>
<td>DS3ADEXT</td>
<td>Nine additional extent descriptions.</td>
</tr>
<tr>
<td>135(X'87')</td>
<td>Bitstring</td>
<td>5</td>
<td>DS3PTRDS</td>
<td>Pointer (CCHHR) to next format-3 DSCB, or zero.</td>
</tr>
<tr>
<td>140(X'8C')</td>
<td>Bitstring</td>
<td>-</td>
<td>DS3END</td>
<td>-</td>
</tr>
</tbody>
</table>

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.

Table 4 on page 12 shows the contents of a format-4 DSCB.

Table 4 on page 12 shows the contents of a format-4 DSCB.

Exception: The format-4 DSCB has a 44-byte key of X'04' bytes not shown in Table 4 on page 12.
### Using the VTOC

**Table 4. DSCB Format-4**

<table>
<thead>
<tr>
<th>Offset (Hex)</th>
<th>Type</th>
<th>Length</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0(X'00')</td>
<td>Character</td>
<td>1</td>
<td>DS4IDFMT</td>
<td>Format identifier (X'F4').</td>
</tr>
<tr>
<td></td>
<td>DS4IDC</td>
<td></td>
<td></td>
<td>Constant value of X'F4' in DS4FMT.</td>
</tr>
<tr>
<td>1(X'01')</td>
<td>Character</td>
<td>5</td>
<td>DS4HPCHR</td>
<td>Highest address (CCHHR) of a format-1 DSCB.</td>
</tr>
<tr>
<td>6(X'06')</td>
<td>Unsigned</td>
<td>2</td>
<td>DS4DSREC</td>
<td>Number of available DSCBs.</td>
</tr>
<tr>
<td>8(X'08')</td>
<td>Character</td>
<td>4</td>
<td>DS4HCCHH</td>
<td>CCHH of next available alternate track.</td>
</tr>
<tr>
<td>12(X'0C')</td>
<td>Unsigned</td>
<td>2</td>
<td>DS4NOATK</td>
<td>Number of remaining alternate tracks.</td>
</tr>
<tr>
<td>14(X'0E')</td>
<td>Bitstring</td>
<td>1</td>
<td>DS4VTOCI</td>
<td>VTOC indicators.</td>
</tr>
<tr>
<td></td>
<td>DS4DOSBT</td>
<td></td>
<td></td>
<td>VSE bit. Either invalid format 5 DSCBs or indexed VTOC. Previously DOS(VSE) bit. See DS4IVTOC.</td>
</tr>
<tr>
<td></td>
<td>DS4DVTOC</td>
<td></td>
<td></td>
<td>Index was disabled.</td>
</tr>
<tr>
<td></td>
<td>DS4EFVLD</td>
<td></td>
<td></td>
<td>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 DS4FLVL and DS4FPTR.</td>
</tr>
<tr>
<td></td>
<td>DS4DSTKP</td>
<td></td>
<td></td>
<td>VSE stacked pack.</td>
</tr>
<tr>
<td></td>
<td>DS4DOCVT</td>
<td></td>
<td></td>
<td>VSE converted VTOC.</td>
</tr>
<tr>
<td></td>
<td>DS4DIRF</td>
<td></td>
<td></td>
<td>DIRF bit. A VTOC change is incomplete.</td>
</tr>
<tr>
<td></td>
<td>DS4DICVT</td>
<td></td>
<td></td>
<td>DIRF reclaimed.</td>
</tr>
<tr>
<td></td>
<td>DS4IVTOC</td>
<td></td>
<td></td>
<td>Volume uses an indexed VTOC.</td>
</tr>
<tr>
<td>15(X'0F')</td>
<td>Unsigned</td>
<td>1</td>
<td>DS4NOEXT</td>
<td>Number of extents in the VTOC(X'01').</td>
</tr>
<tr>
<td>16(X'10')</td>
<td>Bitstring</td>
<td>1</td>
<td>DS4SMSFG</td>
<td>System managed storage indicators.</td>
</tr>
<tr>
<td></td>
<td>DS4NTSMS</td>
<td></td>
<td></td>
<td>Non-system managed volume.</td>
</tr>
<tr>
<td></td>
<td>DS4SMSCV</td>
<td></td>
<td></td>
<td>System managed volume in initial status.</td>
</tr>
<tr>
<td></td>
<td>Reserved.</td>
<td></td>
<td></td>
<td>Reserverd.</td>
</tr>
<tr>
<td></td>
<td>DS4SM</td>
<td></td>
<td>System managed volume.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DS4SMSTS</td>
<td></td>
<td>System managed volume.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Reserverd.</td>
<td></td>
<td></td>
<td>Reserverd.</td>
</tr>
<tr>
<td>17(X'11')</td>
<td>Binary</td>
<td>1</td>
<td>DS4DEVAC</td>
<td>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.</td>
</tr>
<tr>
<td>18(X'12')</td>
<td>Character</td>
<td>14</td>
<td>DS4DEVCT</td>
<td>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 &quot;Performing Track Calculations (TRKCALC macro)&quot; on page 307).</td>
</tr>
<tr>
<td>18(X'12')</td>
<td>Character</td>
<td>4</td>
<td>DS4DEVSZ</td>
<td>Device size.</td>
</tr>
<tr>
<td>18(X'12')</td>
<td>Binary</td>
<td>2</td>
<td>DS4DSCYL</td>
<td>Number of logical cylinders including alternates, if any exist. Unsigned number. Set to X'FFFE' for devices with more than 65,520 cylinders, indicating that cylinder-managed space exists (DS4CYLMG is set on) and extended attribute DSCBs, formats 8 and 9, are allowed.</td>
</tr>
<tr>
<td>20(X'14')</td>
<td>Binary</td>
<td>2</td>
<td>DS4DSTRK</td>
<td>Number of tracks in a logical cylinder.</td>
</tr>
<tr>
<td>22(X'16')</td>
<td>Unsigned</td>
<td>2</td>
<td>DS4DEVTK</td>
<td>Device track length.</td>
</tr>
<tr>
<td>24(X'18')</td>
<td>Binary</td>
<td>2</td>
<td>DS4DEVOV</td>
<td>Keyed record overhead.</td>
</tr>
<tr>
<td>24(X'18')</td>
<td>Binary</td>
<td>1</td>
<td>DS4DEVI</td>
<td>Non-last-keyed record overhead.</td>
</tr>
<tr>
<td>25(X'19')</td>
<td>Binary</td>
<td>1</td>
<td>DS4DEVL</td>
<td>Last keyed record overhead.</td>
</tr>
<tr>
<td>26(X'1A')</td>
<td>Binary</td>
<td>1</td>
<td>DS4DEVK</td>
<td>Non-keyed record overhead differential.</td>
</tr>
<tr>
<td>27(X'1B')</td>
<td>Bitstring</td>
<td>1</td>
<td>DS4DEVFG</td>
<td>Flag byte 1.</td>
</tr>
<tr>
<td></td>
<td>DS4DEVAV</td>
<td></td>
<td></td>
<td>Value in DS4DEVAC, number of alternate cylinders is valid. Otherwise, the number is not available in this DSCB.</td>
</tr>
</tbody>
</table>
Table 4. DSCB Format-4 (continued)

<table>
<thead>
<tr>
<th>Offset Dec (Hex)</th>
<th>Type</th>
<th>Length</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>28(X’1C’)</td>
<td>Binary</td>
<td>2</td>
<td>DS4DEVTL</td>
<td>Device tolerance.</td>
</tr>
<tr>
<td>30(X’1E’)</td>
<td>Binary</td>
<td>1</td>
<td>DS4DEVDT</td>
<td>Number of DSCBs per track.</td>
</tr>
<tr>
<td>31(X’1F’)</td>
<td>Binary</td>
<td>1</td>
<td>DS4DEVDDB</td>
<td>Number of PDS directory blocks per track.</td>
</tr>
<tr>
<td>32(X’20’)</td>
<td>Binary</td>
<td>8</td>
<td>DS4AMTIM</td>
<td>VSAM time stamp.</td>
</tr>
<tr>
<td>40(X’28’)</td>
<td>Character</td>
<td>3</td>
<td>DS4AMCAT</td>
<td>VSAM catalog indicator.</td>
</tr>
<tr>
<td>40(X’28’)</td>
<td>Bitstring</td>
<td>1</td>
<td>DS4VSIND</td>
<td>VSAM indicators.</td>
</tr>
<tr>
<td>41(X’29’)</td>
<td>Unsigned</td>
<td>2</td>
<td>DS4VSCR A</td>
<td>Relative track location of the CRA.</td>
</tr>
<tr>
<td>43(X’2B’)</td>
<td>Binary</td>
<td>8</td>
<td>DS4R2TIM</td>
<td>VSAM volume/catalog match time stamp.</td>
</tr>
<tr>
<td>51(X’33’)</td>
<td>Character</td>
<td>5</td>
<td>DS4F6PTR</td>
<td>Pointer (CCHHR) to first format-6 DSCB, or zero. (No longer supported as non-zero).</td>
</tr>
<tr>
<td>56(X’38’)</td>
<td>Character</td>
<td>5</td>
<td>DS4F6PTR</td>
<td>Pointer (CCHHR) to first format-7 DSCB, or first format-5 DSCB contains free space information.</td>
</tr>
<tr>
<td>61(X’3D’)</td>
<td>Character</td>
<td>10</td>
<td>DS4VTOCE</td>
<td>VTOC extent description.</td>
</tr>
<tr>
<td>71(X’47’)</td>
<td>Character</td>
<td>10</td>
<td>DS4F6PTR</td>
<td>VTOC extent description.</td>
</tr>
<tr>
<td>81(X’51’)</td>
<td>Character</td>
<td>1</td>
<td>DS4EFLVL</td>
<td>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).</td>
</tr>
<tr>
<td>82(X’52’)</td>
<td>Character</td>
<td>5</td>
<td>DS4EFPTR</td>
<td>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.</td>
</tr>
<tr>
<td>87(X’57’)</td>
<td>Character</td>
<td>1</td>
<td>DS4MCU</td>
<td>Minimum allocation size in cylinders for cylinder-managed space. Each extent in this space must be a multiple of this value.</td>
</tr>
<tr>
<td>88(X’58’)</td>
<td>Character</td>
<td>4</td>
<td>DS4DCYL</td>
<td>Number of logical cylinders. Valid when DS4DSCYL= X’FFFE’.</td>
</tr>
<tr>
<td>92(X’5C’)</td>
<td>Character</td>
<td>2</td>
<td>DS4LCYL</td>
<td>First cylinder address/4095 where space is managed in multicylinder units. Cylinder-managed space begins at this address. Valid when DS4CYLMG is set.</td>
</tr>
</tbody>
</table>
Using the VTOC

Table 4. DSCB Format-4 (continued)

<table>
<thead>
<tr>
<th>Offset Dec (Hex)</th>
<th>Type</th>
<th>Length</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>94(X'5E')</td>
<td>Character</td>
<td>1</td>
<td>DS4DEVF2</td>
<td>Device Flags Byte 2</td>
</tr>
<tr>
<td>1... ...</td>
<td></td>
<td></td>
<td>DS4CYLMG</td>
<td>Cylinder-managed space exists on this volume and begins at DS4LCYL in multicylinder units of DS4MCU. DS4EADSCB is also set when this flag is on.</td>
</tr>
<tr>
<td>.1... ...</td>
<td></td>
<td></td>
<td>DS4EADSCB</td>
<td>Extended attribute DSCBs, Format 8 and 9 DSCBs, are allowed on this volume.</td>
</tr>
<tr>
<td>...xx</td>
<td></td>
<td></td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>xxxx</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>95(X'5F')</td>
<td>1</td>
<td></td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>96(X'60')</td>
<td>DS4END</td>
<td>-</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 (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.

Table 5 shows the contents of a format-5 DSCB.

Table 5. DSCB Format-5

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

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 z/OS DFSMSdfp Diagnosis for the layout of the Format-7 DSCB.

**Format-9 DSCB**

**Name:** Metadata and DSCB pointers.

**Function:** Contains metadata about the data set and pointers to all format 3 DSCBs for the data set.

**How Many:** One for each format 8 DSCB.

**How Found:** Chained from a format-8 DSCB that represents the data set. Data sets that have a format 1 DSCB do not have a format 9 DSCB.

Table 6 shows the contents of a format-9 DSCB.

<table>
<thead>
<tr>
<th>Offset Dec (Hex)</th>
<th>Type</th>
<th>Length</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (X'00')</td>
<td>Character</td>
<td>1</td>
<td>DS9KEYID</td>
<td>Key identifier.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DS9KEY</td>
<td>Constant value of X'09' in DS9KEYID</td>
</tr>
<tr>
<td>1 (X'01')</td>
<td>Binary</td>
<td>1</td>
<td>DS9SUBTY</td>
<td>Subtype number for format 9.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DS9SUBT1</td>
<td>Constant value of binary 1 to represent subtype 1.</td>
</tr>
</tbody>
</table>

If your program uses the content of a format 9 DSCB, then it should test the subtype field to learn the format of the DSCB. Currently only one subtype is defined.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Type</th>
<th>Length</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 (X'02')</td>
<td>Binary</td>
<td>1</td>
<td>DS9NUMF9</td>
<td>Number of format 9 DSCB's for this data set. Valid only in the first format 9 DSCB.</td>
</tr>
<tr>
<td>3 (X'03')</td>
<td>Character</td>
<td>1</td>
<td>DS9FLAG1</td>
<td>Format 9 DSCB flag byte 1.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DS9CREAT</td>
<td>Format 9 DSCB built by create.</td>
</tr>
<tr>
<td>4 (X'04')</td>
<td>Character</td>
<td>8</td>
<td>DS9JOBNAME</td>
<td>Job name used to create the data set described by its format 8 DSCB. Valid only when DS9CREAT is on (see offset 3 (X'03')).</td>
</tr>
<tr>
<td>12 (X'0C')</td>
<td>Character</td>
<td>8</td>
<td>DS9STEPNAME</td>
<td>Step name used to create the data set described by its format 8 DSCB. Valid only when DS9CREAT is on (see offset 3 (X'03')).</td>
</tr>
<tr>
<td>20 (X'12')</td>
<td>Character</td>
<td>6</td>
<td>DS9TIME</td>
<td>Number of microseconds since midnight, local time, that the data set described its format 8 DSCB was created. See creation date field, DS1CREDT (offset 53(X'35')), for the date. Valid only when DS9CREAT is on (see offset 3 (X'03')).</td>
</tr>
<tr>
<td>26 (X'20')</td>
<td>Character</td>
<td>18</td>
<td>*</td>
<td>Reserved.</td>
</tr>
<tr>
<td>44 (X'2C')</td>
<td>Character</td>
<td>1</td>
<td>DS9FMTID</td>
<td>Format identifier.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DS9IDC</td>
<td>Constant value of X'F9' in DS9FMTID.</td>
</tr>
<tr>
<td>45 (X'2D')</td>
<td>Binary</td>
<td>1</td>
<td>DS9NUMF3</td>
<td>Number of format 3 pointers that follow</td>
</tr>
<tr>
<td>46 (X'2E')</td>
<td>Binary</td>
<td>50</td>
<td>DS9F3</td>
<td>Pointers to first to tenth format-3 DSCBs</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DS9F3P</td>
<td>First pointer to a format-3 DSCB</td>
</tr>
</tbody>
</table>

Beginning of the fields that are unique to the format 9 subtype 1 DSCB.

The following five bytes occur ten times.
### Using the VTOC

**Table 6. Format-9 DSCB (continued)**

<table>
<thead>
<tr>
<th>Offset Dec (Hex)</th>
<th>Type</th>
<th>Length</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>46 (X'2E')</td>
<td>Binary</td>
<td>2</td>
<td>DS9F3CC</td>
<td>Cylinder number in pointer to format-3</td>
</tr>
<tr>
<td>48 (X'30')</td>
<td>Binary</td>
<td>2</td>
<td>DS9F3HH</td>
<td>Track number in pointer to format-3</td>
</tr>
<tr>
<td>50 (X'32')</td>
<td>Binary</td>
<td>1</td>
<td>DS9F3R</td>
<td>Record number in pointer to format-3</td>
</tr>
<tr>
<td>96 (X'60')</td>
<td>Character</td>
<td>20</td>
<td>DS9ATRV1</td>
<td>Attribute bytes available for vendor use. See <a href="#">Vendor fields in DS9ATRV1</a></td>
</tr>
<tr>
<td>116 (X'74')</td>
<td>Character</td>
<td>19</td>
<td>DS9ATRI2</td>
<td>Attribute fields, for future IBM definition</td>
</tr>
</tbody>
</table>

End of the fields that are unique to the format 9 subtype 1 DSCB.

<table>
<thead>
<tr>
<th>Offset Dec (Hex)</th>
<th>Type</th>
<th>Length</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>135 (X'87')</td>
<td>Bitstring</td>
<td>5</td>
<td>DS9PTRDS</td>
<td>Pointer (CCHHR) to next format-9 DSCB, the first format-3 DSCB or zero.</td>
</tr>
<tr>
<td>135 (X'87')</td>
<td>Binary</td>
<td>2</td>
<td>DS9CCPTR</td>
<td>Cylinder number in DSCB pointer</td>
</tr>
<tr>
<td>138 (X'89')</td>
<td>Binary</td>
<td>2</td>
<td>DS9HHPTR</td>
<td>Track number in DSCB pointer</td>
</tr>
<tr>
<td>140 (X'8B')</td>
<td>Binary</td>
<td>1</td>
<td>DS9RPTR</td>
<td>Record number in DSCB pointer</td>
</tr>
<tr>
<td>140 (X'8C')</td>
<td></td>
<td></td>
<td>DS9END</td>
<td></td>
</tr>
</tbody>
</table>

**Vendor fields in DS9ATRV1 in the format 9 DSCB**

z/OS does not enforce any rules on the content of the DS9ATRV1 field. It is available for vendor products to set. IBM provides the following recommendations:

Each vendor should store subfields in the following format beginning at the leftmost byte:

<table>
<thead>
<tr>
<th>Offset Dec (Hex)</th>
<th>Type</th>
<th>Length</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (X'0')</td>
<td>Character</td>
<td>1</td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>xxxx ....</td>
<td>Number of bytes that follow this two-byte header. Minimum value is 0000.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>.... xxxx</td>
<td></td>
</tr>
<tr>
<td>1 (X'1')</td>
<td>Character</td>
<td>1</td>
<td></td>
<td>One-byte field containing a vendor identification issued by IBM.</td>
</tr>
<tr>
<td>2 (X'2')</td>
<td>Character</td>
<td></td>
<td></td>
<td>Beginning of variable-length field up to 15 bytes.</td>
</tr>
</tbody>
</table>

### Allocating and Releasing DASD Space

DADSM allocate and extend routines assign tracks and cylinders on direct access volumes for data sets. The DADSM extend routine gets additional space for a data set 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.

Every data set occupies an integral number of tracks. Unused space on a track cannot be used for another data set unless the whole track is released. The minimum amount of space allocated to a data set is zero tracks but a PDS with no tracks can never contain a member.

After a certain place on the volume, the system rounds up each primary or secondary space request so it is a multiple of 21 cylinders. Currently that place is after the first 65520 cylinders. The space within the first 65520 cylinders is called the track-managed space, even if data sets are allocated in units of cylinders. The space after the first 65520 cylinders is called the cylinder-managed space. If the volume capacity is 65520 cylinders or less, then all of the space is track-managed.
Using the VTOC

**Note:** When a user program needs to determine the characteristics of a given volume, it must separately check each of the following characteristics and not assume that the value obtained for one characteristic implies the values for the other three:

- Whether space beyond a certain place on the volume is managed in multicylinder units.
- The location on the volume where the multicylinder units begin.
- The number of cylinders in a multicylinder unit.
- Whether the VTOC contains format 8 and 9 DSCBs.

**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.

An SMS-managed volume requires an indexed VTOC; otherwise, the VTOC index is highly recommended. For additional information about SMS-managed volumes, see [z/OS DFSMS Implementing System-Managed Storage](#).

**Note:** You can use the ICKDSF REFORMAT REFVTOC command to rebuild a VTOC index to reclaim any no longer needed index space and to possibly improve access times.

z/OS does support sharing a non-SMS-managed volume that contains a VTOC index with a non-z/OS system. If the other system updates the VTOC and turns on the DS4DOSBT, then later when z/OS is used to modify the VTOC, DADSM can detect that the index is no longer valid. The z/VSE® operating system sets this bit on.

Device Support Facilities (ICKDSF) initializes a VTOC index into 2048-byte physical blocks, or 8192-byte physical blocks on an extended address volume, named VTOC index records (VIRs). The DEVTYPE INFO=DASD macro can be used to return the actual block size or it can be determined from examining the format-1 DSCB of the index data set. 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 and format-8 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.volser’, where ‘volser’ is the serial number of the volume containing the VTOC and its index. The name must be unique within the system to avoid ENQ contention and must conform to standard data set naming conventions.

**Note:**
Using the VTOC

1. If first character of volser is numeric, you must either precede or replace the first character with the letter "V" (for example, if volser is 12345, then you should use either V12345 or V2345).
2. RMF™ statistics on the VTOC index will always show the data set name as SYS1.VTOCIX.Vvolser regardless of what the actual name is on the volume.

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 4.

Figure 4. 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 or format-8 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 space within the first 65,520 cylinders is called the track-managed space. The VPSMs for this area 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. The space after the first 65,520 cylinders is called the cylinder-managed space. The VPSMs for this area contains bit maps of only multicylinder units. Individual cylinders and tracks are not mapped in this type of VPSM. A value of one indicates that the multicylinder unit is allocated; zero, that it is unallocated.

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). On an extended address volume, the VIXM record header is enlarged to accommodate the volume size, free space statistics for the VTOC and index, free space statistics for the entire volume and from track-managed space. Fields in this expanded header define the RBA of the first VPSM record that maps multicylinder units, along with the minimum allocation unit in cylinders for the VPSMs that map cylinder-managed space.

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.)

VTOC map of DSCBs
The VMDS shows the DSCBs that have been allocated in the VTOC. The map contains a bit map of DSCBs corresponding the relative DSCB record in the VTOC. A value of one indicates that the DSCB has been has been allocated; zero indicates that it is unallocated.

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 56.) The DADSM macros and tasks covered here include:
- LSPACE provides information on volume size, free space on the volume, free space on the VTOC and INDEX, volume fragmentation, and VTOC status. Also provided is information on the size of the track-managed space and its free space statistics.
- OBTAIN reads one or more DSCBs from the VTOC.
- PARTREL releases unused space from a sequential or partitioned data set or a PDSE.
- REALLOC allocates DASD space.
To read one or more DSCBs 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. On a request to read multiple DSCBs specify the NUMBERDSCB= parameter on the OBTAIN or CAMLST macro and provide consecutive 140-byte return areas in virtual storage to contain this number of DSCBs. When you specify the name of the data set, an identifier (format-1, format-4, or format-8) DSCB is read into virtual storage. To read a DSCB other than a format-1, format-4, or format-8 DSCB, specify an absolute track address (see the example on page “Example” on page 41). Code the EADSCB=OK on the OBTAIN or CAMLST macro when your program supports DSCBs that describe data sets with format-8 and format-9 DSCBs. The extent descriptors in DSCBs for a data set described with these formats may have 28-bit cylinder track addresses. Use the TRKADDR macro or IECTRKAD service to manipulate 16-bit or 28-bit cylinder track addresses.

**Restriction:** 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.

Another way is to code the RLSE option on the SPACE keyword on the DD statement or the dynamic allocation equivalent. This technique does not require APF authorization. It requires that your program open the data set with the OUTPUT, EXTEND, OUTIN, OUTINX or INOUT option and the last operation before closing the data set not be a read or POINT macro.

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

**Requesting DASD Volume Information Using LSPACE**

LSPACE provides information on volume size, free space on the volume, free space on the VTOC and INDEX, volume fragmentation, and VTOC status. Also provided is information on the size of the track-managed space and its free space statistics. The LSPACE macro returns status information (such as LSPACE subfunction, return code, and reason code) in the parameter list. The LSPACE macro also returns the return code in register 15. For volumes that are configured with more than 9999 cylinders, you can use the EXPMSG option to create an expanded message return area that the LSPACE macro needs. For volumes that are configured with cylinder-managed space, you can use the XEXPMSG option to create an extended expanded message return area that the LSPACE macro needs. The use of XEXPMSG is recommended for all requests to return message data. The expanded data return area (EXPDATA) will return binary data of free space and total volume space information for volumes. For volumes with cylinder-managed space, this will be returned as free space for the entire volume and free space for the track-managed space. The two sets of free space data will be the same for a volume that does not have cylinder-managed space. The use of EXPDATA is recommended for all requests to return binary data. You can have LSPACE return additional information such as the format 4 DSCB, the total number of free extents on the volume or the fragmentation index. This information can be returned in the:

- Message return area
- Expanded message return area
- Extended expanded message return area
Data return area
Expanded data return area
Format-4 DSCB return area

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

If the device is not ready when you issue LSPACE and remains not ready, LSPACE eventually gives return code 4 with a timeout message. (See Table 8 on page 33.) In the current level of the system, LSPACE defaults to waiting as long as 240 seconds. You can change this amount of time by setting the byte at offset 7 in the parameter list. (See Table 7 on page 30.) You must use the list and execute forms of the macro because the macro has no parameter for this.

For more information about the LSPACE return code, subfunction code, and the subfunction return and reason codes, see Table 8 on page 33 and z/OS DFSMSdfp Diagnosis.

**LSPACE—Standard Form**
The format of the standard form of the LSPACE macro is:

```
/LSPACE

MF=I
UCB=addr(reg)
MSG=addr(reg)
EXPMSG=addr(reg)
DATA=addr(reg)
XEXPMSG=addr(reg)
EXPDATA=addr(reg)

SMF=TEST YES NONE
F4DSCB=addr(reg)
DATATYPE=FRAGINDEX INDEX VOLUME VTOC
IMPLIED_VERSION
PLISTVER=plistver MAX

```

The keywords are the same as described in the execute form of the LSPACE macro. See the execute form of the LSPACE macro for the descriptions.

**MF=I**

Specifies the standard form of the LSPACE macro.

I Specifies the standard form of the macro. This generates a standard
Using the VTOC

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.

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

PLISTVER=plistver | IMPLIED_VERSION | MAX
This keyword defines the version of the LSPACE parameter list that should be generated for the MF=I form of the LSPACE macro.

PLISTVER=plistver specifies the version of the LSPACE parameter list that should be generated, where plistver is either 1 or 2. This PLISTVER= keyword is required for any macro keys associated with version 2 or larger to be specified. The macro keys associated with each supported version of the macro are listed below:

PLISTVER=1
PLISTVER=1 is associated with the following macro keys:
DATA
EXPSMG
F4DSCB
MSG
SMF

PLISTVER=2
PLISTVER=2 is associated with the following macro keys:
XEXPMSG
EXPDATA
DATATYPE

When PLISTVER= IMPLIED_VERSION is specified the generated parameter list is the lowest version that allows all of the parameters on the invocation to be processed. When PLISTVER is omitted, the default is the lowest version of the parameter list, which is version 1.

When PLISTVER= MAX is specified, the generated parameter list is the largest size currently supported. This size may grow from release to release thus possibly affecting the amount of storage needed by your program. If your program can tolerate this, IBM recommends that you always specify MAX when creating the list form parameter list as that will ensure that the list form parameter list is always long enough to hold whatever parameters might be specified on the execute form.

LSPACE-Execute Form
The format of the execute form of the LSPACE macro is:
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 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.

**Restrictions:**
Using the VTOC

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
   z/OS HCD Planning
   for details.
2. LSPACE does not support VIO UCBs.

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

   Restriction: 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 32.

   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 34.

   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 34.

   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
Using the VTOC

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 Table 4 on page 12.

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.

XEXPMSG=addr or (reg) or 0
Specifies the address of a caller-provided 95-byte extended expanded message return area into which LSPACE returns either a free space message or, for unsuccessful requests, status information. Specify this keyword if you wish to obtain free space information in the message return area for volumes that are configured with cylinder-managed space. The returned free space will include space for the total volume and space from the track-managed space on a volume. The two sets of free space message data will be the same for a volume that does not have cylinder-managed space. The use of XEXPMSG is recommended for all requests to return message data. See "LSPACE Information Return Areas" on page 31 for a description of the message return area.

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.

EXPDATA=addr or (reg) or 0
Specifies the address of a caller-provided EXPANDED data return area into which LSPACE returns expanded free space and volume information. Specify this keyword if you wish to obtain free space information and total volume space in the LSPACE data return area for volumes. The returned free space will include space for the total volume and space from the track managed space on a volume. For volumes with cylinder-managed space this data will be returned as free space for the entire volume and free space for the track-managed space. The two sets of free space data will be the same for a volume that does not
have cylinder-managed space. The use of EXPDATA is recommended for all requests to return binary data. See Table 9 on page 34 for a description of the expanded data return area.

addr – RX-type address
  Specifies the address of the expanded data return area.

(reg) – (2-12)
  Specifies a register containing the address of the expanded data return area.

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

DATATYPE= ALL or FRAGINDEX or INDEX or VOLUME or VTOC
  This keyword is allowed only when the DATA or EXPDATA keyword is specified. Only the information specified will be returned to the caller. DATATYPE is valid for both non-EAV and EAV. This keyword will eliminate unnecessary I/O required to retrieve free space information that is not be required by the caller. DATATYPE=ALL is the default.

  ALL  Provide all available LSPACE statistics. This is the default

  FRAGINDEX
    Provide the fragmentation index

  INDEX
    Provide free space information related to the VTOC INDEX

  VOLUME
    Provide free space information for the VOLUME

  VTOC  Provide free space information related to the VTOC

PLISTVER
  This keyword, if specified on the MF=E form of the LSPACE macro, will generate an MNOTE.

LSPACE—List Form
  The format of the list form of the LSPACE macro is:
MF=L or (L,MSG) or (L,EXPMSG) or (L,EXPMSG) or (L,DATA) or (L,EXPDATA)

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,XEXPMSG)**
Generates the required extended expanded message return area constants (95-bytes) in the calling program. No other parameters are allowed. Use this keyword to obtain the message data area for all volume sizes including ones with cylinder-managed space.

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

**(L,EXPDATA)**
Generates the required expanded data return area constants in the calling program. No other parameters are allowed. Use the keyword to obtain the data area for all volume sizes including ones with cylinder-managed space.

**DATATYPE=ALL or FRAGINDEX or INDEX or VOLUME or VTOC**

This keyword is allowed only when the DATA or EXPDATA keyword is specified. Only the information specified will be returned to the caller. DATATYPE is valid for both non-EAV and EAV. This keyword will eliminate unnecessary I/O required to retrieve free space information that is not be required by the caller. DATATYPE=ALL is the default.

**ALL**
Provide all available LSPACE statistics. This is the default.
Using the VTOC

FRAGINDEX
Provide the fragmentation index

INDEX
Provide free space information related to the VTOC INDEX

VOLUME
Provide free space information for the VOLUME

VTOC
Provide free space information related to the VTOC

PLISTVER=plistver | IMPLIED_VERSION | MAX
This keyword defines the version of the LSPACE parameter list that should be generated for the MF=I form of the LSPACE macro.

PLISTVER=plistver specifies the version of the LSPACE parameter list that should be generated, where `plistver` is either 1 or 2. This PLISTVER= keyword is required for any macro keys associated with version 2 or larger to be specified. The macro keys associated with each supported version of the macro are listed below:

**PLISTVER=1**
PLISTVER=1 is associated with the following macro keys:
- DATA
- EXPMSG
- F4DSCB
- MSG
- SMF

**PLISTVER=2**
PLISTVER=2 is associated with the following macro keys:
- XEXPMSG
- EXPDATA
- DATATYPE

When PLISTVER= IMPLIED_VERSION is specified the generated parameter list is the lowest version that allows all of the parameters on the invocation to be processed. When PLISTVER is omitted, the default is the lowest version of the parameter list, which is version 1.

When PLISTVER= MAX is specified, the generated parameter list is the largest size currently supported. This size may grow from release to release thus possibly affecting the amount of storage needed by your program. If your program can tolerate this, IBM recommends that you always specify MAX when creating the list form parameter list as that will ensure that the list form parameter list is always long enough to hold whatever parameters might be specified on the execute form.

**LSPACE–DSECT Form**
The format of the DSECT form of the LSPACE macro is:

```plaintext
LSPACE

MF=0

0,MSG

0,EXPMSG

0,XEXPMSG

0,DATA

0,EXPDATA

label

PLISTVER=plistver

MAX
```
MF=D or (D,MSG) or (D,EXPM SG) or (D,XEXPM SG) or (D,DATA) or (D,EXPDATA)

Specifies the DSECT form of the LSPACE macro.

D  Generates a DSECT that maps the LSPACE parameter list. No other parameters are allowed. See Table 7 on page 30 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 32.

(D,EXPM SG)  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 34.

(D,XEXPM SG)  Generates a DSECT that maps the extended expanded message return area (95-bytes). No other parameters are allowed. Use this keyword to obtain the message data area for all volume sizes including ones with cylinder-managed space. For the format of the area, see “Expanded Message Return Area” on page 34.

(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 34.

(D,EXPDATA)  Generates a DSECT that maps the base and expanded data return area. No other parameters are allowed. Use the keyword to obtain the data area for all volume sizes including ones with cylinder-managed space. For the format of the area, see “Data Return Area” on page 34.

PLISTVER=plistver | MAX
This keyword defines the version of the LSPACE parameter list that should be generated for the MF=D form of the LSPACE macro.

PLISTVER=plistver specifies the version of the LSPACE parameter list that should be generated, where plistver is either 1 or 2. This PLISTVER= keyword is required for any macro keys associated with version 2 or larger to be specified. The macro keys associated with each supported version of the macro are listed below:

PLISTVER=1
PLISTVER=1 is associated with the following macro keys:
  DATA
  EXPMSG
  F4DSCB
  MSG
  SMF

PLISTVER=2
PLISTVER=2 is associated with the following macro keys:
  XEXPM SG
  EXPDATA
  DATATYPE

When PLISTVER= MAX is specified, the generated parameter list is the largest size currently supported. This size may grow from release to release thus possibly affecting the amount of storage needed by your program. If
Using the VTOC

your program can tolerate this, IBM recommends that you always specify MAX when creating the list form parameter list as that will ensure that the list form parameter list is always long enough to hold whatever parameters might be specified on the execute form.

When PLISTVER is omitted, the default is the lowest version of the parameter list mapping.

### Table 7. Format of the LSPACE Parameter List (MF=D)

<table>
<thead>
<tr>
<th>Name</th>
<th>Offset</th>
<th>Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSPAPL</td>
<td>0(X'00')</td>
<td>4</td>
<td>EBCDIC 'LSPA'.</td>
</tr>
<tr>
<td>LSPAPLID</td>
<td>0(X'00')</td>
<td>4</td>
<td>EBCDIC 'LSPA'.</td>
</tr>
<tr>
<td>LSPANGTH</td>
<td>4(X'04')</td>
<td>2</td>
<td>Length of parameter list.</td>
</tr>
<tr>
<td>LSPAFLAG</td>
<td>6(X'06')</td>
<td>1</td>
<td>Parameter flag byte.</td>
</tr>
<tr>
<td>LPSASMF</td>
<td></td>
<td>10</td>
<td>SMF=YES.</td>
</tr>
<tr>
<td>LPSASMFT</td>
<td></td>
<td>01</td>
<td>SMF=TEST.</td>
</tr>
<tr>
<td>LSPADATA</td>
<td></td>
<td>100</td>
<td>Free space data request.</td>
</tr>
<tr>
<td>LSPAMSG</td>
<td></td>
<td>10</td>
<td>Message data returned.</td>
</tr>
<tr>
<td>LSPAEMSG</td>
<td></td>
<td>10</td>
<td>Expanded message data requested.</td>
</tr>
<tr>
<td>LSPAESLP</td>
<td></td>
<td>10</td>
<td>Expanded LSPACE parameter input list provided</td>
</tr>
<tr>
<td>LSPASMT</td>
<td></td>
<td>10</td>
<td>Set on when expanded data (EXPDATA) or extended expanded message (XEXPMSG) information is returned by LSPACE processing. Systems prior to z/OS V1R10 do not set this flag. A program assembled (on a z/OS V1R10 or later system) with the LSPACE macro using the extended EXPMSG information respectively when the program is run on a system prior to z/OS V1R10.</td>
</tr>
<tr>
<td>LSPAFRES</td>
<td></td>
<td>10</td>
<td>Reserved.</td>
</tr>
<tr>
<td>LSPAXTIM</td>
<td></td>
<td>1</td>
<td>Input/output (I/O) timeout value in seconds for first LSPACE channel program (default: 240 seconds)</td>
</tr>
<tr>
<td>LSPAERCD</td>
<td></td>
<td>1</td>
<td>LSPACE return code. See Figure 5 on page 32.</td>
</tr>
<tr>
<td>LSPASFID</td>
<td></td>
<td>1</td>
<td>LSPACE subfunction code to further describe the LSPACE result.</td>
</tr>
<tr>
<td>LSPASFPC</td>
<td></td>
<td>1</td>
<td>Processing complete.</td>
</tr>
<tr>
<td>LSPASFVP</td>
<td></td>
<td>1</td>
<td>Validate parameters.</td>
</tr>
<tr>
<td>LSPASFUS</td>
<td></td>
<td>1</td>
<td>Check UCB status.</td>
</tr>
<tr>
<td>LSPASFQ</td>
<td></td>
<td>1</td>
<td>ENQ on SYSZDMNT.</td>
</tr>
<tr>
<td>LSPASF45</td>
<td></td>
<td>1</td>
<td>Read F4 and first F5 DSCB (EXCP).</td>
</tr>
<tr>
<td>LSPASFN5</td>
<td></td>
<td>1</td>
<td>Read next F5 DSCB (EXCP).</td>
</tr>
<tr>
<td>LSPASF07</td>
<td></td>
<td>1</td>
<td>Read volume label with Timeout.</td>
</tr>
<tr>
<td>LSPASF4X</td>
<td></td>
<td>1</td>
<td>Read F4 and maps (CVAFDIR).</td>
</tr>
<tr>
<td>LSPASFEX</td>
<td></td>
<td>1</td>
<td>Get free extents (CVAFD5M).</td>
</tr>
<tr>
<td>LSPASF00</td>
<td></td>
<td>1</td>
<td>Get F0 count (CVAFD5M).</td>
</tr>
<tr>
<td>LSPASFVR</td>
<td></td>
<td>1</td>
<td>Get VIR count (CVAFD5M).</td>
</tr>
<tr>
<td>LSPASFVD</td>
<td></td>
<td>1</td>
<td>Check for VRF (CVAFVRF).</td>
</tr>
<tr>
<td>LSPASF85</td>
<td></td>
<td>1</td>
<td>ESTAE routine entered. Processing error in LSPACE.</td>
</tr>
<tr>
<td>LSPASF86</td>
<td></td>
<td>1</td>
<td>LSPACE STIMERM timeout</td>
</tr>
<tr>
<td>LSPASFRT</td>
<td></td>
<td>1</td>
<td>Subfunction return code.</td>
</tr>
<tr>
<td>LSPASFRS</td>
<td></td>
<td>1</td>
<td>Subfunction reason code.</td>
</tr>
<tr>
<td>LSPARS01</td>
<td></td>
<td>1</td>
<td>Invalid parameter list storage key.</td>
</tr>
<tr>
<td>LSPARS02</td>
<td></td>
<td>1</td>
<td>Invalid parameter list ID.</td>
</tr>
</tbody>
</table>
Table 7. Format of the LSPACE Parameter List (MF=D) (continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>Offset</th>
<th>Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSPARS03</td>
<td>X'03'</td>
<td></td>
<td>Invalid LSPACE flag.</td>
</tr>
<tr>
<td>LSPARS04</td>
<td>X'04'</td>
<td></td>
<td>Invalid authorization for System Management Facility (SMF) flag.</td>
</tr>
<tr>
<td>LSPARS05</td>
<td>X'05'</td>
<td></td>
<td>Invalid message or data return area storage key.</td>
</tr>
<tr>
<td>LSPARS06</td>
<td>X'06'</td>
<td></td>
<td>Invalid format-4 DSCB return area storage key.</td>
</tr>
<tr>
<td>LSPARS07</td>
<td>X'07'</td>
<td></td>
<td>Invalid UCB address.</td>
</tr>
<tr>
<td>LSPARS08</td>
<td>X'08'</td>
<td></td>
<td>Invalid virtual UCB address.</td>
</tr>
<tr>
<td>LSPARS09</td>
<td>X'09'</td>
<td></td>
<td>Invalid VTOC pointer (UCBVTOC). Either UCBVTOC is zero, or it does not match the volume label.</td>
</tr>
<tr>
<td>LSPARS0B</td>
<td>X'0B'</td>
<td></td>
<td>ESTAE return code nonzero</td>
</tr>
<tr>
<td>LSPAUCB</td>
<td>12(X'0C')</td>
<td>4</td>
<td>UCB address.</td>
</tr>
<tr>
<td>LSPAFRSP</td>
<td>16(X'10')</td>
<td>4</td>
<td>Address of message or data return area.</td>
</tr>
<tr>
<td>LSPAFMT4</td>
<td>20(X'14')</td>
<td>4</td>
<td>Address of format-4 DSCB.</td>
</tr>
<tr>
<td>LSPAEXPL</td>
<td>24(X'18')</td>
<td>24</td>
<td>Expanded parm list area</td>
</tr>
<tr>
<td>LSPAFLAG2</td>
<td>1</td>
<td></td>
<td>Parameter flag byte 2</td>
</tr>
<tr>
<td>LSPAXMSG</td>
<td>1... ...</td>
<td></td>
<td>Extended expanded message area requested (XEXPMSG)</td>
</tr>
<tr>
<td>LSPAEDAT</td>
<td>.1... ...</td>
<td></td>
<td>Expanded output data area requested (EXPDATA)</td>
</tr>
<tr>
<td>*</td>
<td>.x. ...</td>
<td></td>
<td>Unused</td>
</tr>
</tbody>
</table>

The next five flags pertain to the DATATYPE keyword values:

<table>
<thead>
<tr>
<th>Name</th>
<th>Offset</th>
<th>Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSPAFRSI</td>
<td>...1 ...</td>
<td></td>
<td>Volume is specified, the return of volume free space information is requested</td>
</tr>
<tr>
<td>LSPAFRVT</td>
<td>.... 1...</td>
<td></td>
<td>Vtoc is specified, the return of vtoc free space information is requested</td>
</tr>
<tr>
<td>LSPAFRVX</td>
<td>.... .1..</td>
<td></td>
<td>Index is specified, the return of index free space information is requested</td>
</tr>
<tr>
<td>LSPAFRFI</td>
<td>.... ..1.</td>
<td></td>
<td>Fragindex is specified, the return of the fragmentation index is requested</td>
</tr>
<tr>
<td>LSPAFALL</td>
<td>.... ...1</td>
<td></td>
<td>All is specified or defaulted, the return of all free space information is requested, including the frag index</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Offset</th>
<th>Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSPARES2</td>
<td>25(X'19')</td>
<td>3</td>
<td>Reserved</td>
</tr>
<tr>
<td>LSPARES3</td>
<td>26(X'1A')</td>
<td>20</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

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

See Table 8 on page 33 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 z/OS DFSMSdfp Diagnosis 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 following caller requested return areas.
Requests for the MSG, EXPMMSG, XEXPMMSG, DATA, and EXPDATA 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.

**Message Return Area:** LSPACE returns information to a 30-byte message return area (Figure 5). If you provide a message return area with the MSG option, LSPACE returns EBCDIC text, qualified by return codes as shown in Table 8 on page 33.

---

**Figure 5. DADSM LSPACE Free Space Information MF=(D,MSG)**

Using the VTOC
Table 8. DADSM LSPACE Message Return Area Contents

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
</tr>
</thead>
</table>

0(X'00')

For MSG keyword (30 byte area):

Text: SPACE=aaaa,bbbb, cccc/dddd,eeee where:

Free space statistics from the entire volume:
- 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

For aaaa, bbbb, and cccc, the maximum value indicated is 9999 even if the actual value is greater. Use EXPMSG to avoid use of the maximum 9999.

For EXPMSG keyword (40 byte area):

Text: SPACE=aaaaaa,bbbbbb, cccccc/dddddd,eeeee where:

Free space statistics from the entire volume:
- aaaaaa = Total number of free cylinders
- bbbbbbb = Total number of additional free tracks
- cccccc = Total number of free extents
- ddddddd = Number of cylinders in largest free extent
- eeeee = Number of additional tracks in largest free extent

For aaaaaa, bbbbbbb, cccccc, and ddddddd, the maximum value indicated is 99999 even if the actual value is greater. Use XEXPMSG to avoid use of the maximum 999999.

For XEXPMSG keyword (95 byte area):

Text: SPACE=aaaaaaaaa,bbbbbbbbbb, ccccccddddeeeeee where:

Free space statistics from the entire volume:
- aaaaaaaaaa = Total number of free cylinders
- bbbbbbbbbbb = Total number of additional free tracks
- ccccccdddde = Total number of free extents
- dddddddddd = Number of cylinders in largest free extent
- eee = Number of additional tracks in largest free extent

Free space statistics from the track-managed space of the volume. For a volume without cylinder-managed space, these statistics will be equivalent to the total volume statistics above:

- hhhhhhhhhhh = Total number of free cylinders
- sssssssssss = Total number of additional free tracks
- iiiiiiiiiiiiiii = Number of cylinders in largest free extent
- jjj = Number of additional tracks in largest free extent

Leading zeroes for each value will be set to 'blanks' in the returned message area.
Using the VTOC

Table 8. DADSM LSPACE Message Return Area Contents (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4(X'04')</td>
<td>Text: LSPACE—PERMANENT I/O ERROR</td>
</tr>
<tr>
<td>4(X'04')</td>
<td>Text: LSPACE—I/O TIMEOUT ERROR</td>
</tr>
<tr>
<td>8(X'08')</td>
<td>Text: LSPACE—NON-STANDARD OS VOLUME. The volume does not have a VTOC index and free space information is not available. Either an operating system other than z/OS has allocated or freed space on the volume or the volume does not have a format 5 or 7 DSCB chain.</td>
</tr>
<tr>
<td>12(X'0C')</td>
<td>Text: LSPACE—UCB NOT READY</td>
</tr>
<tr>
<td></td>
<td>Text: LSPACE—UCBVTOC IS ZERO</td>
</tr>
<tr>
<td></td>
<td>Text: LSPACE—INVALID PARAMETER</td>
</tr>
<tr>
<td></td>
<td>Text: LSPACE—NOT A DIRECT ACCESS VOL</td>
</tr>
<tr>
<td>16(X'10')</td>
<td>No text returned (invalid parameter list or SMF indicator) This return code indicates a parameter list error, which can be a bad parameter list storage key, parameter list ID is invalid (not set to ‘LSPA’), or the parameter list size is not sufficient.</td>
</tr>
<tr>
<td>20 (X'14')</td>
<td>No text returned (processing error in LSPACE)</td>
</tr>
</tbody>
</table>

Expanded Message Return Area: LSPACE returns information to a 40-byte expanded message return area (Figure 6). By providing an expanded message return area with the EXPMSG option, LSPACE returns EBCDIC text, qualified by return codes as shown in Table 8 on page 33.

Even though the expanded message return area displays the same return code information as the standard message return area, the space information provided for return code zero consists of six-digit values (aaaaaa instead of aaaa).

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

Figure 6. DADSM LSPACE Free Space Information Format, MF=(D,EXPMSG)

Data Return Area: If you provide a data return area with the DATA or EXPDATA keywords, LSPACE returns the information shown below. The expanded data return area is provided only when EXPDATA is specified. EXPDATA is the recommended keyword for returning binary data from LSPACE.

Table 9. LSPACE Data Return Area Format

<table>
<thead>
<tr>
<th>Name</th>
<th>Offset</th>
<th>Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSPDRETN</td>
<td>0(X'00')</td>
<td>1</td>
<td>Return area status byte</td>
</tr>
<tr>
<td>LSPDSPAC</td>
<td></td>
<td></td>
<td>Returned space information</td>
</tr>
<tr>
<td>LSPDF0CN</td>
<td>1</td>
<td></td>
<td>Returned F0 DSCB count</td>
</tr>
<tr>
<td>LSPDVRCN</td>
<td>.1</td>
<td></td>
<td>Returned free VIR count</td>
</tr>
<tr>
<td>LSPDFRGI</td>
<td>..1</td>
<td></td>
<td>Returned fragmentation index</td>
</tr>
<tr>
<td>LSPDCYLM</td>
<td>....1</td>
<td></td>
<td>Returned data is for a volume with cyl-managed space</td>
</tr>
<tr>
<td>LSPDRRES</td>
<td>. . . x x x</td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>LSPDSTAT</td>
<td>1(X'01')</td>
<td>1</td>
<td>Volume status byte</td>
</tr>
<tr>
<td>LSPDIXDS</td>
<td>1</td>
<td></td>
<td>Index exists for VTOC</td>
</tr>
<tr>
<td>LSPDIXAC</td>
<td>.1</td>
<td></td>
<td>Index VTOC active</td>
</tr>
<tr>
<td>LSPDSRES</td>
<td>. x x x x x x</td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>LSPDRSV1</td>
<td>2(X'02')</td>
<td>2</td>
<td>Reserved</td>
</tr>
</tbody>
</table>
Table 9. LSPACE Data Return Area Format (continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>Offset</th>
<th>Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Begin of the base data return area: For DATA and EXPDATA requests, the following 32 bytes if requested or defaulted by the DATATYPE keyword will be returned. They represent statistics that describe the entire volume. LSPDFS and LSPDVIRS are not applicable to volume statistics. LSPDEVFS: Free space statistics from the entire volume. For volumes with cylinder-managed space (LSPDCYLM = '1') these statistics represent space from both the track and cylinder-managed space on the volume. See LSPDTMFS for statistics from the track-managed space on the volume.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LSPDEVFS</td>
<td>0X04</td>
<td>20</td>
<td>Total volume free space</td>
</tr>
<tr>
<td>LSPDNEXT</td>
<td>0X04</td>
<td>4</td>
<td>Number of free extents</td>
</tr>
<tr>
<td>LSPDTCYL</td>
<td>0X08</td>
<td>4</td>
<td>Total free cylinders</td>
</tr>
<tr>
<td>LSPDTRK</td>
<td>0X10C</td>
<td>4</td>
<td>Total addition free tracks</td>
</tr>
<tr>
<td>LSPDLCYL</td>
<td>0X10</td>
<td>4</td>
<td>Number of cylinders in largest free extent</td>
</tr>
<tr>
<td>LSPDLTRK</td>
<td>0X10C'</td>
<td>4</td>
<td>Number of additional tracks in largest free extent</td>
</tr>
<tr>
<td>LSPDFOS</td>
<td>0X18</td>
<td>4</td>
<td>Format 0 count</td>
</tr>
<tr>
<td>LSPDVIRS</td>
<td>0X1C</td>
<td>4</td>
<td>Free VIR count</td>
</tr>
<tr>
<td>LSPDFRAG</td>
<td>0X20</td>
<td>4</td>
<td>Fragmentation count</td>
</tr>
<tr>
<td>End of the base data return area.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Beginning of the expanded data return area: For EXPDATA requests, the following 32 bytes of statistics if requested or defaulted by the DATATYPE keyword will be returned. LSPDTMFS: Free space statistics from track-managed space on a volume for a volume with cylinder-managed space (LSPDCYLM = '1'). For volumes with no cylinder-managed space (LSPDCYLM = '0') than these statistics are equivalent to the total volume statistics described above.

<table>
<thead>
<tr>
<th>Name</th>
<th>Offset</th>
<th>Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSPDTMFS</td>
<td>0X24</td>
<td>24</td>
<td>Track-managed free space</td>
</tr>
<tr>
<td>LSPDVNXT</td>
<td>0X24</td>
<td>4</td>
<td>Number of free extents</td>
</tr>
<tr>
<td>LSPDVCTL</td>
<td>0X28</td>
<td>4</td>
<td>Total free cylinders</td>
</tr>
<tr>
<td>LSPDVTK</td>
<td>0X2C</td>
<td>4</td>
<td>Total additional free cylinders</td>
</tr>
<tr>
<td>LSPDVLCL</td>
<td>0X30</td>
<td>4</td>
<td>Number of cylinders in the largest free extent</td>
</tr>
<tr>
<td>LSPDVLTK</td>
<td>0X34</td>
<td>4</td>
<td>Number of additional free tracks in the largest free extent</td>
</tr>
<tr>
<td>LSPDVFRG</td>
<td>0X38</td>
<td>4</td>
<td>Fragmentation index</td>
</tr>
<tr>
<td>Volume size information</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LSPDVOLI</td>
<td>0X3C</td>
<td>8</td>
<td>Volume size information</td>
</tr>
<tr>
<td>LSPDTRKS</td>
<td>0X3C</td>
<td>4</td>
<td>Total number of tracks on volume</td>
</tr>
<tr>
<td>LSPDTRKM</td>
<td>0X40</td>
<td>4</td>
<td>Total number of tracks in track-managed space when LSPDCYLM='1'. Set to the value of LSPDTRKS otherwise. When LSPDCYLM='1', this value is also the first track address where cylinder managed space begins.</td>
</tr>
<tr>
<td>LSPDRSV8</td>
<td>0X44</td>
<td>60</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

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 Table 4 on page 12.
**Using the VTOC**

**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 z/OS UNIX files. You will receive unpredictable results if you issue OBTAIN for a z/OS UNIX 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 VIO data sets.

The format of the OBTAIN and CAMLST macros is:

```
<table>
<thead>
<tr>
<th>OBTAINT</th>
<th>listname_addrx</th>
<th>NUMBERDSCB=number_dscbs</th>
</tr>
</thead>
<tbody>
<tr>
<td>EADSCB</td>
<td>NOTOK</td>
<td>OK</td>
</tr>
</tbody>
</table>
```

```
| listname—CAMLST—SEARCH—,dsname_relexp,—vol_relexp,—wkarea_relexp— |
|-------------------------|-------------------------|
| NUMBERDSCB=number_dscbs | EADSCB=OK             |
```

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

**SEARCH**
Code this operand as shown.
Using the VTOC

**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.

**Tip:** 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.

**NUMBERDSCB**
Specifies a value between 0 and 255 that designates the number of consecutive 140-byte return areas that are provided in wkarea_relexp. The system treats a value of 0 as a 1. Currently the system does not support a chain of more than 12 DSCBs for one data set, but it is valid for you to provide an area that is longer than currently needed. The system verifies that the provided area is valid. When you provide an area that is long enough to contain more than one DSCB, obtain processing will return DSCBs for the requested data set name in logical VTOC order until all the 140-byte return areas are used. The logical VTOC order is a format-1 DSCB, followed by zero or more format-3 DSCBs or a format-8 DSCB, followed by one or more format-9 DSCBs, followed by zero or more format-3 DSCBs. No absolute maximum number of DSCBs for a data set should be assumed. The actual number of DSCBs are returned in a field located in the first 140-byte return area.

On the OBTAIN macro you can code a register number or symbol for a register number in parentheses. It means that the specified register contains the number of DSCBs that can fit in the return area. If you code the NUMBERDSCB parameter on OBTAIN, the macro execution stores the value in the CAMLST area. You cannot code a register on the CAMLST macro.

Note that for programs run on a pre-z/OS R10 system that do not support this keyword, the NUMBERDSCB value will be treated as if it were 1.

**EADSCB**
Specifies whether this program supports data sets with format-8 and format-9 DSCBs. Such data sets can appear on extended address volumes.

**EADSCB=OK**
Code **EADSCB=NOTOK** when your program does not support data sets that have format-8 and format-9 DSCBs. The extent descriptors in DSCBs for a data set described with these formats may have track addresses that contain cylinder addresses 65,520 or larger. **EADSCB=OK** is accepted for data sets described by all DSCB types, including format-1 DSCBs, regardless of the volume size where the data set resides. Your program can also run on an older level of the system that does not support this keyword. In these cases, **EADSCB=OK** is ignored. **EADSCB=OK** sets byte 2 bit 4 in the OBTAIN parameter list to on.

**EADSCB=NOTOK**
Code **EADSCB=NOTOK** when your program does not support DSCBs that describe data sets with format-8 and format-9 DSCBs. **EADSCB=NOTOK** is the default when the EADSCB keyword is not specified.
Using the VTOC

When \texttt{EADSCB=NOTOK} is coded or assumed by default, \texttt{OBTAIN} will set return code 24 if the target of the \texttt{OBTAIN} request has a format-8 DSCB.

\texttt{EADSCB=NOTOK} sets byte 2 bit 4 in the \texttt{OBTAIN} parameter list to zero.

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

\begin{verbatim}
OBTAIN   DSCBABC          READ DSCB FOR DATA
*        SET A.B.C INTO DATA
*        AREA NAMED WORKAREA
\end{verbatim}

\begin{verbatim}
DSCBABC CAMLIST 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
\end{verbatim}

**Recommendation:** Check the return codes.

The \texttt{OBTAIN} macro instruction points to the \texttt{CAMLST} parameter list. \texttt{SEARCH}, the first operand of \texttt{CAMLST}, specifies that a DSCB be read into virtual storage using the data set name at the address indicated in the second operand. \texttt{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. \texttt{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. \texttt{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 \texttt{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 Table 10.

**Return Codes from OBTAIN (Reading by Data Set Name)**

\textit{Table 10. DADSM OBTAIN Return Codes}

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0(X'00')</td>
<td>Successful completion of \texttt{OBTAIN} routine.</td>
</tr>
<tr>
<td>4(X'04')</td>
<td>The required volume was not mounted.</td>
</tr>
<tr>
<td>8(X'08')</td>
<td>The format-1 DSCB was not found in the VTOC of the specified volume.</td>
</tr>
<tr>
<td>12(X'0C')</td>
<td>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).</td>
</tr>
<tr>
<td>16(X'10')</td>
<td>Invalid work area pointer.</td>
</tr>
<tr>
<td>24(X'18')</td>
<td>Data set has a format-8 DSCB and \texttt{EADSCB=NOTOK} is in effect</td>
</tr>
<tr>
<td>24(X'1C')</td>
<td>Internal error with \texttt{EADSCB=NOTOK} in effect.</td>
</tr>
</tbody>
</table>

On return from \texttt{SEARCH} requests, the first 103 bytes of the first 140 byte return area will contain:

- The 96 byte data portion of the format-1, format-4, or format-8 DSCB
Followed by 5 bytes that contain the absolute track address (CCHHR) of this DSCB. For VSAM object names that appear in the catalog, but not in the VTOC or VIO data sets, these 5 bytes contain zeros.

Followed by a 2 byte count of the total number of DSCBs associated with the data set DSCB, even if there are insufficient return areas into which to read them all. This count includes all DSCB types that could describe a data set. Set to zero for DSCBs constructed from catalog when DSCBs are not present in the VTOC.

If the request specified multiple DSCBs, only the format-1 or format-8 pseudo DSCB will be returned when the DSCB was not found in the VTOC and had to be constructed from Catalog. No forward chain pointer to the next DSCB will be present in this returned DSCB. The remaining 140 byte return areas will contain the entire 140 byte key and data portions of the associated DSCB.

If the request did not specify multiple DSCBs, a format-1 or format-8 pseudo DSCB will be returned when the DSCB was not found in the VTOC and had to be constructed from Catalog. No forward chain pointer to the next DSCB will be present in this returned DSCB. 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:

```
/SM590000/SM590000
OBTAIN
label
NUMBERDSCB=number_dscbs
/SM590000

/SM590000/SM590000
CAMLST
SEEK
listname
label
dsname_relexp
cchhr_relexp
vol_relexp
/SM590000

/SM590000/SM590000

/SM590000

/SM590000

listname
Label of the CAMLST macro instruction.
SEARCH
Code this operand as shown when calling obtain to read by the passed data set name.
```
Using the VTOC

SEEK
Code this operand as shown when calling obtain to read by the passed
CCHHR device address

dsnname_relexp
For SEARCH requests, specifies the virtual storage location of a fully-qualified
data set name. The area that contains the name must be 44-bytes long.

cchhr_relexp
For SEEK requests, 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
For all requests, specifies the virtual storage location of a 140-byte work area,
or larger, that must be defined.

NUMBERDSCB
Specifies a value between 0 and 255 that designates the number of consecutive
140-byte return areas that are provided in wkarea_relexp. The system treats a
value of 0 as a 1. Currently the system does not support a chain of more than
12 DSCBs for one data set, but it is valid for you to provide an area that is
longer than currently needed. The system verifies that the provided area is
valid. When you provide an area that is long enough to contain more than one
DSCB, OBTAIN processing will return DSCBs for the requested data set name
in logical VTOC order until all the 140-byte return areas are used. The logical
VTOC order is a format-1 DSCB, followed by zero or more format-3 DSCBs or
a format-8 DSCB, followed by one or more format-9 DSCBs, followed by zero
or more format-3 DSCBs. No absolute maximum number of DSCBs for a data
set should be assumed. When the target of the seek operation is not a format-1
or format-8 DSCB, the NUMBERDSCB value is treated as if it were 1 and only
that single DSCB will be returned.

On the OBTAIN macro, you can code a register number or symbol for a
register number in parentheses. This means that the specified register contains
the number of DSCBs that can fit in the return area. If you code the
NUMBERDSCB parameter on OBTAIN, the macro execution stores the value in
the CAMLST area. You cannot code a register on the CAMLST macro.

Note that for programs run on a pre-z/OS R10 system that do not support this
keyword, the NUMBERDSCB value will be treated as if it were 1.

EADSCB
Specifies whether this program supports data sets with format-8 and format-9
DSCBs. Such data sets can appear on extended address volumes.

EADSCB=OK
Code EADSCB=NOTOK when your program does not support data sets
that have format-8 and format-9 DSCBs. The extent descriptors in DSCBs
for a data set described with these formats may have track addresses that
contain 28-bit cylinder numbers. EADSCB=OK is accepted for data sets
described by all DSCB types, including format-1 DSCBs, regardless of the
volume size where the data set resides. Your program can also run on an
older level of the system that does not support this keyword. In these
cases, EADSCB=OK is ignored. EADSCB=OK sets byte 2 bit 4 in the
OBTAIN parameter list to on.
Using the VTOC

**EADSCB=NOTOK**

Code **EADSCB=NOTOK** when your program does *not* support DSCBs that describe data sets with format-8 and format-9 DSCBs. **EADSCB=NOTOK** is the default when the EADSCB keyword is not specified.

When **EADSCB=NOTOK** is coded or assumed by default, OBTAIN will set return code 24 if the target of the OBTAIN request has a format-8 or format-9 DSCB. OBTAIN will not check format-3 DSCB extent ranges for track addresses that contain 28-bit cylinder numbers.

**EADSCB=NOTOK** sets byte 2 bit 4 in the OBTAIN parameter list to zero.

Example: In the following example, the DSCB at actual-device address X'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
```

Recommendation: 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 Table 11.

**Return Codes from OBTAIN (Reading by Absolute Device Address)**

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0(X'00')</td>
<td>Successful completion of OBTAIN routine.</td>
</tr>
<tr>
<td>4(X'04')</td>
<td>The required volume was not mounted.</td>
</tr>
<tr>
<td>8(X'08')</td>
<td>The format-1 DSCB was not found in the VTOC of the specified volume or an ‘alien’ (non-VSAM) request was returned from Catalog.</td>
</tr>
<tr>
<td>12(X'0C')</td>
<td>A permanent I/O error was encountered or an unexpected error return code was received from CVAF.</td>
</tr>
<tr>
<td>16(X'10')</td>
<td>Invalid work area pointer.</td>
</tr>
<tr>
<td>20(X'14')</td>
<td>The SEEK option was specified and the absolute track address (CCHHR) is not within the boundaries of the VTOC.</td>
</tr>
<tr>
<td>24(X'18')</td>
<td>Data set has a format-8 or format-9 DSCB and <strong>EADSCB=NOTOK</strong> is in effect.</td>
</tr>
<tr>
<td>28(X'1C')</td>
<td>Internal error with EADSCB=NOTOK in effect.</td>
</tr>
</tbody>
</table>
On return from SEEK requests with multiple DSCBs requested, the entire 140 bytes of the first return area will contain the 140 byte key and data portions of the DSCBs read from the volume. The remaining 140 byte return areas will contain the entire 140 byte key and data portions of the DSCBs chained from this first DSCB until the chain of DSCBs has ended or until the 140 byte return areas are exhausted.

On return from SEEK requests without multiple DSCBs specified, the first return area will contain the entire 140 byte key and data portions of the DSCB read from the user’s passed device address.

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 RLESE 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.

For an extended address volume, partial release processing will release space on multicylinder unit boundaries when the last used track is in an extent in cylinder-managed space. For VSAM striped data sets where at least one stripe is on an extended address volume, partial release processing will ensure the stripes remain the same size. For these reasons, it is possible that the high allocated track after the release may be larger than the last used track or that no space could be released.

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 z/OS UNIX files. You will receive unpredictable results if you issue PARTREL for z/OS UNIX 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:

```
label PARTREL MF=(E,addr) or (E,(reg)), DSN=addr or (reg), UCB=(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.

```plaintext
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. For a VSAM file, you must specify the component name, not the cluster name.

```plaintext
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 145 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'.

When TIOT=NOENQ is specified and flag PRLTIOTX is set on, the ENQ TEST is not executed. This is a dangerous option and can cause data corruption. IBM recommends that you not set PRLTIOTX.

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, only extended format VSAM data sets are supported.

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:
Restrictions:
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
```

**Description:** The following example provides a description of the parameter list:

```
+PRPLID DS CL4 EBCDIC 'PREL' FOR PARTREL
+PRPLNGTH DS AL2 LENGTH OF PARAMETER LIST
+PRPRCDE DS H ERROR CODE RETURNED FROM
+PRPLFLG DS XL1 PARTIAL RELEASE
+PRPLPGM EQU X'00' MODE=PGM (PROBLEM PROGRAM)
+PRPLSUP EQU X'80' MODE=SUP (SUPERVISOR STATE)
+PRPTIOT EQU X'40' TIOT=NOENQ
+PRPLERASE EQU X'10' ERASE=YES
+PRPLCLA EQU 01 VSAMCLLR=VSAMCLSE
+PRPLCNV EQU X'04' VSAMCLLR=NOTVSAM
+PRPLBYPS EQU X'02' VSAMSEC=BYPASS
+PRPLNBYPS EQU X'01' VSAMSEC=NOBYPASS
+PRPLFLG2 DS XL1
+PRPLKSDS EQU X'00' VSAMTYPE=KSDS
+PRPLVSDS EQU X'40' VSAMTYPE=ESDS
+PRPLVRRDS EQU X'20' VSAMTYPE=RRDS
+PRPLVVRRD EQU X'10' VSAMTYPE=VRRDS
+PRPLLDS EQU X'08' VSAMTYPE=LDS
+PRPLSTD EQU X'04' VSAMTYPE=STD
+PRPLTDX EQU X'01' No ENQ TEST when TIOT=NOENQ is specified
+PRPLSVD DS XL2 RESERVED
+PRPLDSN DS A DATA SET NAME POINTER
+PRPLUCB DS A UCB POINTER
+PRPLTRKS DS A ADDRESS OF NUMBER OF TRACKS RETURNED
```
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.

Table 12 describes the conditions indicated by the PARTREL return code.

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

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0(X’00’)</td>
<td>Successful</td>
</tr>
<tr>
<td>2(X’02’)</td>
<td>Unable to find extent in format-1 or format-8 DSCB.</td>
</tr>
<tr>
<td>3(X’03’)</td>
<td>Exceeded maximum number of format-3 pointers in format-9 DSCB.</td>
</tr>
<tr>
<td>4(X’04’)</td>
<td>Unable to find extent in format-3 DSCB.</td>
</tr>
<tr>
<td>8(X’08’)</td>
<td>Either the required SYSZTIOT or SYSDSN ENQ failed, or an unrelated DEB indicates that another DCB is open to the data set.</td>
</tr>
<tr>
<td>12(X’0C’)</td>
<td>Invalid parameter list.</td>
</tr>
<tr>
<td>16(X’10’)</td>
<td>One of the following conditions occurred:</td>
</tr>
<tr>
<td></td>
<td>• A permanent I/O error occurred.</td>
</tr>
<tr>
<td></td>
<td>• CVAF provided an unexpected return code.</td>
</tr>
<tr>
<td></td>
<td>• An installation exit rejected the request.</td>
</tr>
<tr>
<td></td>
<td>• An I/O error occurred while the tracks to be released were being erased (for ERASE-on-SCRATCH).</td>
</tr>
<tr>
<td>20(X’14’)</td>
<td>DSN, or DSN pointer is invalid.</td>
</tr>
<tr>
<td>24(X’18’)</td>
<td>Invalid UCB pointer.</td>
</tr>
<tr>
<td>28(X’1C’)</td>
<td>Specified DSORG is not supported.</td>
</tr>
<tr>
<td>32(X’20’)</td>
<td>No room in the VTOC.</td>
</tr>
<tr>
<td>36(X’24’)</td>
<td>Invalid TIOT=NOENQ request; caller does not have exclusive use of SYSDSN.</td>
</tr>
<tr>
<td>40(X’28’)</td>
<td>An error occurred while SMS was processing the request.</td>
</tr>
<tr>
<td>44(X’2C’)</td>
<td>CLOSE is the caller, user rejected the partial release using the PREEXIT routine.</td>
</tr>
<tr>
<td>48(X’30’)</td>
<td>An error occurred during conversion from CCHH to relative track address.</td>
</tr>
</tbody>
</table>
Table 12. DADSM PARTREL Return Codes (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>52(X'34')</td>
<td>An error occurred during conversion from relative track address to CCHH.</td>
</tr>
<tr>
<td>56(X'38')</td>
<td>An error occurred during conversion from new extent descriptor table to old extent descriptor table.</td>
</tr>
<tr>
<td>60(X'3C')</td>
<td>An error occurred during sort conversion of the extent descriptor table.</td>
</tr>
<tr>
<td>64(X'40')</td>
<td>An error occurred during conversion from format-7 to extent descriptor table.</td>
</tr>
<tr>
<td>68(X'44')</td>
<td>An error occurred during conversion from format-5 to extent descriptor table.</td>
</tr>
<tr>
<td>72(X'48')</td>
<td>Input DSCB is not a format-5 or a format-7 DSCB.</td>
</tr>
<tr>
<td>76(X'4C')</td>
<td>An error occurred during conversion from extent descriptor table to format-7.</td>
</tr>
<tr>
<td>80(X'50')</td>
<td>An error occurred during conversion from extent descriptor table to format-5.</td>
</tr>
<tr>
<td>84(X'54')</td>
<td>VSAM standard format data set incorrectly specified. If the data set is VSAM, it must be extended format.</td>
</tr>
<tr>
<td>88(X'58')</td>
<td>VSAMTRKS address parameter must be specified when VSAMTYPE is specified.</td>
</tr>
<tr>
<td>92(X'5C')</td>
<td>The name of the specified VSAM data set is not a VSAM cluster data component.</td>
</tr>
<tr>
<td>96(X'60')</td>
<td>Catalog call to partial release failed.</td>
</tr>
<tr>
<td>100(X'64')</td>
<td>Request was directed to a VSAM data set defined with guaranteed space.</td>
</tr>
<tr>
<td>104(X'68')</td>
<td>Unexpected reason code received back from a call to catalog services.</td>
</tr>
<tr>
<td>108(X'6C')</td>
<td>Unable to find enough F0s to complete the PARTREL request.</td>
</tr>
<tr>
<td>112(X'70')</td>
<td>DAPCCHH passed is in cylinder-managed space, but is not on a multi-cylinder unit boundary.</td>
</tr>
</tbody>
</table>

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 includes the following requirements and options:

- 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.
In addition, 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 (MOV).

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.

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 or format-8 DSCB, and an optional format-3 DSCB, that describe the space and attributes of the desired data set. An optional format-9 DSCBs can also be provided to pass additional attributes:

- Support is provided for, but not limited to, data sets with a user label extent.
- The number of extents to be allocated, and their absolute placement on the volume, are defined by the format-1 or format-8 DSCB and one (optional) format-3 DSCB.

For an absolute request, the following checks are performed prior to writing the passed DSCBs. The type of checks depend on the volume for which the REALLOC macro is issued.

- For any volume, a passed DSCB format must not describe a format that is not supported. The supported formats in REALLOC processing are 1, 3, 8, or 9.
- For an extended address volume, a passed format-1 DSCB must not describe extents that contain cylinder addresses larger than 65,519.
- A passed format-1, format-3, or format-8 DSCB must not describe extents that contain cylinder addresses larger than the highest cylinder address of the volume.
- For an extended address volume, a passed format-8 DSCB must not describe a data set organization that is not EAS eligible.
- For an extended address volume, a passed format-3 or format-8 DSCB must not describe an extent that begins on cylinder address 65519 or lower and that ends on cylinder address 65520 or higher.
- For a volume that is not an extended address volume, a passed format-8 or format-9 DSCB is not allowed.

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.

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 that describe the attributes of the desired data set:

- 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 system managed storage indicators.
Using the VTOC

(PDISMSFG) bytes in the partial DSCB. If PD1DSORG indicates a VSAM data set organization, the maximum number of extents is 7257 across multiple volumes, 123 on a single volume.

**REALLOC—Execute Form**
The format of the execute form of the REALLOC macro is:

```
/SM590000/SM590000
REALLOC
label
MF= (E, addr)
(E, (reg))

/SM590000
REALLOC
label
MF= (E, (reg))

/SM590000
REALLOC
label
MF= (E, addr)
```

- **REALLOC**—Execute Form
  - **label**—Label
  - **MF**—Macro Form
    - **E**—Execute
    - **addr**—Address
    - **(reg)**—Register
  - **ALLOC**—Allocate
    - **ABS**—Absolute
    - **MOV**—Move
    - **NON**—Non
  - **DSSIZE**—Data Set Size
    - **addr**—Address
    - **(reg)**—Register
  - **F3DSCB**—Field 3 DSCB
    - **addr**—Address
    - **(reg)**—Register
  - **F9DSCB**—Field 9 DSCB
    - **addr**—Address
    - **(reg)**—Register
  - **MINAU**—Minimum AU
    - **addr**—Address
    - **(reg)**—Register
  - **NUMF9**—Number of F9 DSCBs
    - **number_dscbs**—Number
    - **(reg)**—Register
  - **PDSCB**—PDSCB
    - **addr**—Address
    - **(reg)**—Register
  - **PDSDIR**—PDSDIR
    - **addr**—Address
    - **(reg)**—Register
  - **UCB**—UCB
    - **(reg)**—Register

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 a REALLOC parameter list.

- **ALLOC**—ABS or MOV or NON
  - Specifies one of the following:
    - **ALLOC**—ABS
      - Specifies that the REALLOC request is for absolute extents.
    - **ALLOC**—MOV
      - Specifies that the REALLOC request is for a movable allocation.
        ALLOC=MOV is the default.
    - **ALLOC**—NON
      - Specifies that the REALLOC request is to rebuild the free space chain on an unindexed VTOC without allocating a data set. Expect a DADSM create return code X'3D'.

- **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.
  - **(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
This parameter is supported for assembly purposes, but the system no longer supports it when your program executes. This because you can no longer create indexed sequential data sets

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).

F9DSCB=addr or (reg) or 0
Specifies the address of a caller-provided contiguous partial format-9 DSCB data area where attribute information from it is to be used when creating a format-9 DSCB. Specify this keyword if you wish to pass format-9 DSCB attribute information to REALLOC processing. Only attribute information in the format-9 DSCB will be processed. Format-9 DSCBs with a subtype field with a value other than 1 will be ignored. See mapping macro, IECSDSL1. The number of contiguous partial format-9 DSCBs defined in this data area is defined in the NUMF9=number_dscb keyword or is defaulted to one.

addr—RX-type address
(reg)—(2-12)

Specifies the address of the partial format-9 DSCB data area.

0
Specifies that you do not want to pass a partial format-9 DSCB data area.

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.
**Using the VTOC**

*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.

**NUMF9=number_dscbs or (reg)**

For REALLOC with F9DSCB requests, *number_dscbs* is an absolute expression or (register) with a value between 0 and 255 that designates the number of consecutive 140-byte partial format-9 DSCB areas that are provided in the F9DSCB=addr. The system treats a value of 0 as a 1 when F9DSCB=addr is specified. Format-9 DSCBs with a subtype field with a value other than 1 are ignored.

**PDSCB=addr or (reg)**

Specifies the address of a partial DSCB (for ALLOC=MOV) or the in-storage address of a full format-1 or format-8 DSCB (for ALLOC=ABS). This DSCB is used as a model to construct the allocated data set's format-1 or format-8 DSCB.

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

Provide a value for the PDSCB parameter and initialize the PD1FMTID field to X’F1’ for a format-1 DSCB or to X’F8’ for a format-8 DSCB. However, the PDSCB attribute information determines EAS eligibility and whether the actual allocated DSCB allocated is a format-1 or format-8 DSCB.

**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.

```
label REALLOC MF=L
      MOV ALLOC=ABS
      F3DSCB=addr
      F9DSCB=addr(reg)
      NUMF9=number_dscbs(reg)
      PDSCB=addr
```

**Restrictions:**

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:

```
label REALLOC MF=D
```

**REALLOC Parameter List**

The parameter list for the DSECT form expansion is:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length or Bit Pattern</th>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0(X'00')</td>
<td>4</td>
<td>REALPL</td>
<td>DSECT name.</td>
</tr>
<tr>
<td>0(X'00')</td>
<td>4</td>
<td>RALPLID</td>
<td>EBCDIC “REAL” for “REALLOC”</td>
</tr>
<tr>
<td>4(X'04')</td>
<td>2</td>
<td>RALNGTH</td>
<td>Length of parameter list</td>
</tr>
<tr>
<td>6(X'06')</td>
<td>2</td>
<td>RAERRCDE</td>
<td>Error code from DADSM create</td>
</tr>
<tr>
<td>8(X'08')</td>
<td>1</td>
<td>RALFLAG</td>
<td>Flag byte</td>
</tr>
<tr>
<td>9(X'09')</td>
<td>1</td>
<td>RALABS</td>
<td>0 means ALLOC=MOV. 1 means ALLOC=ABS.</td>
</tr>
<tr>
<td>10(X'09')</td>
<td>1</td>
<td>RALVTOCE</td>
<td>VTOCENQD=YES</td>
</tr>
<tr>
<td>11(X'09')</td>
<td>1</td>
<td>RALPFLGS</td>
<td>Processing flag byte.</td>
</tr>
</tbody>
</table>
### Using the VTOC

**Table 13. REALLOC Parameter List (continued)**

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length or Bit Pattern</th>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1...</td>
<td>RALDUMMY</td>
<td>Dummy REALLOC parameter list is passed. Only the processing flag byte (byte 9), minimum allocation unit (bytes 16-19), and UCB address (bytes 24-27) are used. Ignore all other bytes and use the values in the JFCB or Partial DSCB interface as passed in register 0.</td>
<td></td>
</tr>
<tr>
<td>..1...</td>
<td>RALTRKAL</td>
<td>Space must be allocated from track-managed space.</td>
<td></td>
</tr>
<tr>
<td>..1...</td>
<td>RALEXREQ</td>
<td>The exact amount of space must be allocated. Applicable to EAV. The request is to be allocated using a combination of the track-managed and/or the cylinder-managed spaces. If the exact space is not available, then the request is failed.</td>
<td></td>
</tr>
<tr>
<td>...1 11...</td>
<td>* Reserved.</td>
<td>Reserve.</td>
<td></td>
</tr>
<tr>
<td>....11</td>
<td>RALEATTR</td>
<td>The extended attribute (EATTR=) value to be used. Valid when RALDUMMY is set and when the JFCB is passed in register 0.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• If 0, EATTR has not been specified. For VSAM data sets, the default behavior is equivalent to EATTR=OPT. For non-VSAM data sets the default behavior is equivalent to EATTR=NO.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• If 1, EATTR=NO has been specified. The data set cannot have extended attributes (format 8 and 9 DSCBs) or optionally reside in EAS.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• If 2, EATTR=OPT has been specified. The data set can have extended attributes and optionally reside in EAS. This is the default behavior for VSAM data sets.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• If 3, Not Used, EATTR treated as not specified.</td>
<td></td>
</tr>
<tr>
<td>10(X'0A')</td>
<td>1 RALNUMMF9</td>
<td>Number of contiguous partial format 9 DSCBs that are located at the address in bytes 32-35. The default is one.</td>
<td></td>
</tr>
<tr>
<td>11(X'0B')</td>
<td>1 Reserved</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12(X'0C')</td>
<td>4 RALDSSZ</td>
<td>Data set size</td>
<td></td>
</tr>
<tr>
<td>16(X'10')</td>
<td>4 RALMAU</td>
<td>Minimum allocation unit</td>
<td></td>
</tr>
<tr>
<td>20(X'14')</td>
<td>4 RALPDSCK</td>
<td>Address of partial DSCB</td>
<td></td>
</tr>
<tr>
<td>24(X'18')</td>
<td>4 RALUCB</td>
<td>Address of UCB</td>
<td></td>
</tr>
<tr>
<td>28(X'1C')</td>
<td>4 RALDQTY</td>
<td>PDS directory quantity</td>
<td></td>
</tr>
<tr>
<td>32(X'20')</td>
<td>4 RAL2DSCB</td>
<td>Address of format 2 DSCB. Mutually exclusive with RAL9DSCB.</td>
<td></td>
</tr>
<tr>
<td>32(X'20')</td>
<td>4 RAL9DSCB</td>
<td>Contiguous partial format 9 DSCBs pointer. RALNUMMF9 defines the number of partial format 9 DSCBs. Format 9 DSCBs with a subtype field with a value other than 1 are ignored. Only format 9 DSCB attribute data in this model will be used. Mutually exclusive with RAL2DSCB.</td>
<td></td>
</tr>
<tr>
<td>36(X'24')</td>
<td>4 RAL3DSCB</td>
<td>Address of format 3 DSCB</td>
<td></td>
</tr>
<tr>
<td>40(X'28')</td>
<td>0 RALEND</td>
<td>Byte after end of list</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>RALENGTH</td>
<td>Symbolic length of parameter list</td>
<td></td>
</tr>
</tbody>
</table>

**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 Table 14 on page 55.
REALLOC returns 4 bytes of diagnostic information in register 0. See z/OS DFSMSdfp Diagnosis.

If an error occurs, REALLOC issues message IEC614I, consisting of failure-related status information. See Table 14 for descriptions of the conditions indicated by the REALLOC return code.

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

Table 14. DADSM CREATE Return Codes

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00(X'00')</td>
<td>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 z/OS DFSMSdfp Diagnosis to determine the failure-related conditions.</td>
</tr>
<tr>
<td>04(X'04')</td>
<td>Duplicate data set name.</td>
</tr>
<tr>
<td>08(X'08')</td>
<td>No room in VTOC or VTOC index.</td>
</tr>
<tr>
<td>12(X'0C')</td>
<td>Permanent I/O error or CVAF error.</td>
</tr>
<tr>
<td>16(X'10')</td>
<td>Requested absolute track not available.</td>
</tr>
<tr>
<td>20(X'14')</td>
<td>Requested quantity not available.</td>
</tr>
<tr>
<td>24(X'18')</td>
<td>Average record length exceeds 65,535 bytes.</td>
</tr>
<tr>
<td>28(X'1C')</td>
<td>ISAM is no longer supported and the request has been failed.</td>
</tr>
<tr>
<td>32(X'20')</td>
<td>SMS configuration parmlib field USESLV was set to NO. The request to allocate on a volume with more than 65,520 cylinders is not allowed.</td>
</tr>
<tr>
<td>40(X'28')</td>
<td>The create request specified DACEXREQ, but the exact amount of space could not be returned. Because the exact amount was required, no space is returned.</td>
</tr>
<tr>
<td>48(X'30')</td>
<td>Invalid DADSM REALLOC parameter list.</td>
</tr>
<tr>
<td>52(X'34')</td>
<td>Invalid JFCB, partial DSCB pointer, or ineligible DSORG specified with F8 ID in the DSCB for an EAS request on EAV.</td>
</tr>
<tr>
<td>56(X'38')</td>
<td>Not enough space on volume for directory.</td>
</tr>
<tr>
<td>60(X'3C')</td>
<td>REALLOC ALLOC=ABS is not supported on unindexed VTOCs.</td>
</tr>
<tr>
<td>61(X'3D')</td>
<td>REALLOC ALLOC=NON requested the free space chain to be rebuilt. The data set was not created.</td>
</tr>
<tr>
<td>64(X'40')</td>
<td>Invalid user label request.</td>
</tr>
<tr>
<td>68(X'44')</td>
<td>Invalid UCB pointer. Requires the address of a UCB, not a UCB copy.</td>
</tr>
<tr>
<td>72(X'48')</td>
<td>VSE VTOC cannot be converted to an unindexed VTOC.</td>
</tr>
<tr>
<td>76(X'4C')</td>
<td>No space parameter given for a new data set or zero space requested at absolute track zero.</td>
</tr>
<tr>
<td>104(X'68')</td>
<td>Invalid space subparameter.</td>
</tr>
<tr>
<td>108(X'6C')</td>
<td>ABSTR request.</td>
</tr>
<tr>
<td>116(X'74')</td>
<td>User labels not supported.</td>
</tr>
<tr>
<td>120(X'78')</td>
<td>Invalid combination of DSIZE and MINAU in REALLOC parameter.</td>
</tr>
<tr>
<td>124(X'7C')</td>
<td>DSIZE not a multiple of MINAU.</td>
</tr>
<tr>
<td>128(X'80')</td>
<td>Directory space requested is larger than primary space.</td>
</tr>
<tr>
<td>136(X'88')</td>
<td>Invalid FMT3 DSCB pointer.</td>
</tr>
<tr>
<td>144(X'90')</td>
<td>EAV extent above 65520 cylinders is not an even integral of the multicylinder unit.</td>
</tr>
<tr>
<td>148(X'94')</td>
<td>Overlapping extents in the VTOC.</td>
</tr>
<tr>
<td>156(X'9C')</td>
<td>DADSM CREATE terminated because of possible VTOC errors.</td>
</tr>
<tr>
<td>164(X'A4')</td>
<td>Allocation terminated because of VSE stacked pack format.</td>
</tr>
<tr>
<td>168(X'A8')</td>
<td>RACDEF failed, data set already defined.</td>
</tr>
<tr>
<td>172(X'AC')</td>
<td>User not authorized to define data set.</td>
</tr>
<tr>
<td>176(X'B0')</td>
<td>Installation exit rejected this request with return code 8.</td>
</tr>
</tbody>
</table>
Using the VTOC

Table 14. DADSM CREATE Return Codes (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>180(X'B4')</td>
<td>Installation exit rejected the request with return code 4.</td>
</tr>
<tr>
<td>184(X'B8')</td>
<td>RACF define with modeling specified and model not found.</td>
</tr>
<tr>
<td>188(X'BC')</td>
<td>Invalid FMT2 DSCB pointer.</td>
</tr>
<tr>
<td>192(X'C0')</td>
<td>Requested data set creation was not allowed by SMS.</td>
</tr>
<tr>
<td>196(X'C4')</td>
<td>Requested data set creation was not possible. Register 0 contains additional diagnostic information.</td>
</tr>
<tr>
<td>200(X'C8')</td>
<td>The PDSE directory could not be built.</td>
</tr>
<tr>
<td>204(X'CC')</td>
<td>VTOC ENQ related failure.</td>
</tr>
<tr>
<td>208(X'D0')</td>
<td>I/O error occurred during the allocation of a data set. The data set will be deleted.</td>
</tr>
<tr>
<td>212(X'D4')</td>
<td>Request failed due to a presence of split cylinder data sets on the volumes.</td>
</tr>
<tr>
<td>216(X'D8')</td>
<td>For this type of data set, the primary quantity requested cannot exceed 65,535 tracks.</td>
</tr>
<tr>
<td>217(X'D9')</td>
<td>Data set could not be created DSNTYPE=LARGE not valid for this data set.</td>
</tr>
<tr>
<td>220(X'DC')</td>
<td>VTOC conversion failed because the VTOC was full.</td>
</tr>
</tbody>
</table>

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 one or more 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.

“Coding CVAF VTOC Access Macros” on page 73 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 7 on page 68 might reside above the 16 MB line. All these areas must be accessible in your program's storage key.

Note: You must supply a UCB address that matches the caller's AMODE. That is, AMODE=24 requires a 24 bit UCB address, while AMODE=31 requires a 31 bit UCB address.

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 or the format-8 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 or format-8 DSCB.

If your program holds the enqueue for the SYSVTOC resource, then no other program can later start and complete a DADSM request. This includes extending or creating a data set on that volume in your own address space. If you try to extend a data set under the same task that holds SYSVTOC, it will be abnormally terminated. If a different task requests SYSVTOC, that task will wait. If your program then requests a resource that is held by that task, the two tasks will be in a deadlock. To prevent deadlocks between tasks in the same address space when either of the tasks has previously enqueued on SYSVTOC, the secondary caller of CVAF must ensure the enqueue bit is on (CV3ENQD) in the CVPL. Other options to get around this limitation are either:

- Allocating a data set so that it does not require secondary extents.
- Requesting that the output data set be on a volume other than the one where the application holds enqueue for the SYSVTOC resource.

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 287; the OPEN macro is described under “OPEN - Initialize Data Control Block for Processing the JFCB” on page 302. For an extended address volume the DCB macro must point to a DCBE where the EADSCB=OK keyword is specified. If you do not code this option, the OPEN function will issue ABEND 113-48 and message IEC142I.

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 z/OS 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'). CVAFFILT request need a longer parm list to accommodate one extra file.

- To get the longer parameter list (X'44' bytes), specify CVPLFSA=YES on the inclusion of the ICVAFPL mapping macro.
- To get the shorter parameter list (X'40' bytes), specify the CVPLX=YES on the inclusion of the ICVAFPL mapping macro.

The ICVAFPL macro maps the CVPL. The format of the CVPL is shown in Table 15.

Note: The area starting at CVCTAR5 is generated only when the CVPLX=YES macro variable is specified on the invocation of ICVAFPL.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Type</th>
<th>Length</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0(X'00')</td>
<td>STRUCTURE</td>
<td>100</td>
<td>CVPL</td>
<td>CVAF parameter list</td>
</tr>
<tr>
<td>0(X'00')</td>
<td>CHARACTER</td>
<td>4</td>
<td>CVLBL</td>
<td>EBCDIC 'CVPL'</td>
</tr>
<tr>
<td>4(X'04')</td>
<td>SIGNED</td>
<td>2</td>
<td>CVLTH</td>
<td>Length of CVPL</td>
</tr>
<tr>
<td>6(X'06')</td>
<td>BIT(8)</td>
<td>1</td>
<td>CVFCTN</td>
<td>Function byte (see values below)</td>
</tr>
<tr>
<td>7(X'07')</td>
<td>UNSIGNED</td>
<td>1</td>
<td>CVSTAT</td>
<td>Status information (see values below)</td>
</tr>
<tr>
<td>8(X'08')</td>
<td>BIT(8)</td>
<td>1</td>
<td>CVFL1</td>
<td>First flag byte</td>
</tr>
<tr>
<td>. ....</td>
<td></td>
<td></td>
<td>CV1IVT</td>
<td>Indexed VTOC accessed</td>
</tr>
<tr>
<td>..1 ....</td>
<td></td>
<td></td>
<td>CV1IOAR</td>
<td>IOAREA=KEEP</td>
</tr>
<tr>
<td>...1 ....</td>
<td></td>
<td></td>
<td>CV1PGM</td>
<td>BRANCH=(YES,PGM)</td>
</tr>
<tr>
<td>...1 ....</td>
<td></td>
<td></td>
<td>CV1MRCDS</td>
<td>MAPRCDS=YES</td>
</tr>
<tr>
<td>... 1....</td>
<td></td>
<td></td>
<td>CV1IRCD5</td>
<td>IXRCDS=KEEP</td>
</tr>
<tr>
<td>.... 111</td>
<td></td>
<td></td>
<td>CV1MAP</td>
<td>Map bits (see next three bits)</td>
</tr>
<tr>
<td>.... 1..</td>
<td></td>
<td></td>
<td>CV1MAPIX</td>
<td>MAP=INDEX</td>
</tr>
</tbody>
</table>
**Table 15. CVAF Parameter List - ICVAFPL (continued)**

<table>
<thead>
<tr>
<th>Offset</th>
<th>Type</th>
<th>Length</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>CV1MAPVT</td>
<td>MAP=VTOC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CV1MAPVL</td>
<td>MAP=VOLUME</td>
</tr>
<tr>
<td>9(X'09')</td>
<td>BIT(8)</td>
<td>1</td>
<td>CVFL2</td>
<td>Second flag byte</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CV2HIVIE</td>
<td>HIVIER=YES</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CV2VRF</td>
<td>VRF data exists</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CV2CNT</td>
<td>COUNT=YES</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CV2RCVR</td>
<td>RECOVER=YES</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CV2SRCH</td>
<td>SEARCH=YES</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CV2DSNLY</td>
<td>DSNONLY=YES</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CV2VER</td>
<td>VERIFY=YES</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CV2NLEVL</td>
<td>Output - new highest level VIER created</td>
</tr>
<tr>
<td>10(X'0A')</td>
<td>BIT(8)</td>
<td>1</td>
<td>CVFL3</td>
<td>Third flag byte</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CV3FILT</td>
<td>FLTAREA=KEEP</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CV3IXERR</td>
<td>Index error found</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CV3PTR31</td>
<td>Address words in CVPL are valid in AMODE 31</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CV3AVT</td>
<td>Support bypass write inhibit error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CV3ENQD</td>
<td>CVAF caller has serialized with another task</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CV3NOPIN</td>
<td>Caller requested no UCBPIN</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CV3HLIXP</td>
<td>Higher level IX pruning done</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CV3RTA4B</td>
<td>CVAFDSM RTA4BYTE=YES</td>
</tr>
<tr>
<td>11(X'0B')</td>
<td>BIT(8)</td>
<td>1</td>
<td>CVFL4</td>
<td>Fourth flag byte</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CV4UCBCA</td>
<td>UCB has been captured</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CV4OBTRQ</td>
<td>Request came from OBTAIN</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CV4NMHID</td>
<td>Data set name hiding for authorized caller.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CV4EADOK</td>
<td>EADSCB=OK specified</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CV4MULTD</td>
<td>Processing of multiple DSCBs on CVAFDSM is</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CV4RSVD</td>
<td>Reserved</td>
</tr>
<tr>
<td>12(X'0C')</td>
<td>ADDRESS 4</td>
<td>4</td>
<td>CVUCB</td>
<td>UCB address</td>
</tr>
<tr>
<td>16(X'10')</td>
<td>ADDRESS 4</td>
<td>4</td>
<td>CVDSN</td>
<td>Data set name address</td>
</tr>
<tr>
<td>16(X'10')</td>
<td>ADDRESS 4</td>
<td>4</td>
<td>CVFCL</td>
<td>Filter criteria list address</td>
</tr>
<tr>
<td>20(X'14')</td>
<td>ADDRESS 4</td>
<td>4</td>
<td>CVBUFL</td>
<td>Buffer list address</td>
</tr>
<tr>
<td>24(X'18')</td>
<td>ADDRESS 4</td>
<td>4</td>
<td>CVIRCDs</td>
<td>Index VIR's buffer list pointer</td>
</tr>
<tr>
<td>28(X'1C')</td>
<td>ADDRESS 4</td>
<td>4</td>
<td>CVMRCDS</td>
<td>MAP VIR's buffer list pointer</td>
</tr>
<tr>
<td>32(X'20')</td>
<td>ADDRESS 4</td>
<td>4</td>
<td>CVIOAR</td>
<td>I/O area address</td>
</tr>
<tr>
<td>36(X'24')</td>
<td>ADDRESS 4</td>
<td>4</td>
<td>CVDEB</td>
<td>DEB address</td>
</tr>
<tr>
<td>40(X'28')</td>
<td>ADDRESS 4</td>
<td>4</td>
<td>CVARG</td>
<td>Argument address</td>
</tr>
<tr>
<td>44(X'2C')</td>
<td>ADDRESS 4</td>
<td>4</td>
<td>CVSPACE</td>
<td>Space parameter list address</td>
</tr>
</tbody>
</table>
Using the VTOC

Table 15. CVAF Parameter List - ICVAFPL (continued)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Type</th>
<th>Length</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>48(X'30')</td>
<td>ADDRESS</td>
<td>4</td>
<td>CVEXTS</td>
<td>Extent table address</td>
</tr>
<tr>
<td>52(X'34')</td>
<td>ADDRESS</td>
<td>4</td>
<td>CVBUFL2</td>
<td>New VRF VIXM Buffer list pointer</td>
</tr>
<tr>
<td>56(X'38')</td>
<td>ADDRESS</td>
<td>4</td>
<td>CVVRFDA</td>
<td>VRF data address</td>
</tr>
<tr>
<td>60(X'3C')</td>
<td>ADDRESS</td>
<td>4</td>
<td>CVCTAR</td>
<td>Count area address</td>
</tr>
<tr>
<td>64(X'40')</td>
<td>ADDRESS</td>
<td>4</td>
<td>CVFSA</td>
<td>Filter save area address</td>
</tr>
<tr>
<td>68(X'44')</td>
<td>ADDRESS</td>
<td>4</td>
<td>CVCTAR5</td>
<td>CCHHR AREA RETURN ADDRESS</td>
</tr>
<tr>
<td>72(X'48')</td>
<td>UNSIGNED</td>
<td>1</td>
<td>CVNMDSBC</td>
<td>NUM OF DSCBs</td>
</tr>
<tr>
<td>73(X'49')</td>
<td>CHARACTER</td>
<td>3</td>
<td>*</td>
<td>Reserved</td>
</tr>
<tr>
<td>76(X'4C')</td>
<td>CHARACTER</td>
<td>24</td>
<td>*</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

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

Table 16. CVFCTN Field of CVPL—Contents and Definitions

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

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. Table 17 on page 61 shows the format of a buffer list header. Table 18 on page 62 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 Table 17.

Table 17. Format of a Buffer List Header

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length or Bit Pattern</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (X’00’)</td>
<td>8</td>
<td>BFLHDR</td>
<td>Buffer list header.</td>
</tr>
<tr>
<td>0 (X’00’)</td>
<td>1</td>
<td>BFLHNOE</td>
<td>Number of entries.</td>
</tr>
<tr>
<td>1 (X’01’)</td>
<td>1</td>
<td>BFLHFL</td>
<td>Key and flag byte.</td>
</tr>
<tr>
<td>.......</td>
<td></td>
<td>BFLHKEY</td>
<td>Protect key of buffer list and buffers.</td>
</tr>
<tr>
<td>.......</td>
<td></td>
<td>BFLHVIR</td>
<td>Reserved.</td>
</tr>
<tr>
<td>.......</td>
<td></td>
<td>BFLHDSCB</td>
<td>Buffer list entries describe DSCBs.</td>
</tr>
<tr>
<td>.......</td>
<td></td>
<td>BFLHWREV</td>
<td>Write multiple DSCBs in reverse order.</td>
</tr>
<tr>
<td>2 (X’02’)</td>
<td>1</td>
<td>BFLHNOEN</td>
<td>Number of entries needed to read all DSCBs describing the DSN. Set by CVAFDIR READ when processing a READ of a format-1 or format-8 DSCB. Applicable to all volume types and set regardless of the value provided in BFLHNOE.</td>
</tr>
<tr>
<td>3 (X’03’)</td>
<td>1</td>
<td>BFLHSP</td>
<td>Identifies the subpool of buffer list and buffers.</td>
</tr>
<tr>
<td>4 (X’04’)</td>
<td>4</td>
<td>BFLHFCHN</td>
<td>Forward chain address of next buffer list.</td>
</tr>
</tbody>
</table>

Buffer List Entry

A buffer list contains one or more entries in contiguous storage. 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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCHHR</td>
<td>5-byte CCHHR</td>
</tr>
<tr>
<td>TTR</td>
<td>0TTR0</td>
</tr>
<tr>
<td>RBA</td>
<td>1 byte of 0 followed by a 4-byte relative byte address.</td>
</tr>
</tbody>
</table>
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 Table 18.

Table 18. Format of a Buffer List Entry

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length or Bit Pattern</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (X'00')</td>
<td>12</td>
<td>BFLE</td>
<td>Buffer list entry.</td>
</tr>
<tr>
<td>0 (X'00')</td>
<td>1</td>
<td>BFLEFL</td>
<td>Flag byte.</td>
</tr>
<tr>
<td>100 . . .</td>
<td>BFLERBA</td>
<td>Argument is RBA.</td>
<td></td>
</tr>
<tr>
<td>010 . . .</td>
<td>BFLECHR</td>
<td>Argument is CCHHR.</td>
<td></td>
</tr>
<tr>
<td>001 . . .</td>
<td>BFLETTR</td>
<td>Argument is TTR.</td>
<td></td>
</tr>
<tr>
<td>. . . 1 . .</td>
<td>BFLEAUPD</td>
<td>CVAF updated argument field.</td>
<td></td>
</tr>
<tr>
<td>. . . 1 . .</td>
<td>BFLEMOD</td>
<td>Data in buffer has been modified.</td>
<td></td>
</tr>
<tr>
<td>. . . 1 . .</td>
<td>BFLESKIP</td>
<td>Skip this entry.</td>
<td></td>
</tr>
<tr>
<td>. . . 1 . .</td>
<td>BFLEIOER</td>
<td>I/O error.</td>
<td></td>
</tr>
<tr>
<td>. . . 1 . .</td>
<td>BFLENOVR</td>
<td>If using CVAFDIR to write a 96-byte DSCB, bypass comparing the DSCB key to the data set name.</td>
<td></td>
</tr>
<tr>
<td>1 (X'01')</td>
<td>1</td>
<td>BFLEFLG2</td>
<td>Flag byte 2. When using CVAFDIR to write multiple DSCBs and this flag is set, CVAF will not verify that the existing DSCB is format 0 prior to writing this DSCB. This overrides VERIFY=YES on CVAFDIR (CV2VER) for this buffer list entry.</td>
</tr>
<tr>
<td>1 (X'01')</td>
<td>. . . .</td>
<td>BFLENVER</td>
<td>Reserved.</td>
</tr>
<tr>
<td>2 (X'02')</td>
<td>1</td>
<td>BFLELTH</td>
<td>Length of DSCB buffer.</td>
</tr>
<tr>
<td>3 (X'03')</td>
<td>5</td>
<td>BFLEARG</td>
<td>Argument of DSCB.</td>
</tr>
<tr>
<td>4 (X'04')</td>
<td>3</td>
<td>BFLEATTR</td>
<td>TTR of DSCB.</td>
</tr>
<tr>
<td>4 (X'04')</td>
<td>4</td>
<td>BFLEARBA</td>
<td>Reserved.</td>
</tr>
<tr>
<td>8 (X'08')</td>
<td>4</td>
<td>BFLEBUF</td>
<td>Buffer address.</td>
</tr>
</tbody>
</table>

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.

Table 19. Format of ICVEDT02 Mapping Macro

<table>
<thead>
<tr>
<th>Offset</th>
<th>Bytes</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (X'00')</td>
<td>8</td>
<td>DT2X7EYE</td>
<td>Identifier=&quot;ICVEDT02&quot;</td>
</tr>
<tr>
<td>8 (X'08')</td>
<td>4</td>
<td>DT2X7LEN</td>
<td>Length of ICVEDT02.</td>
</tr>
<tr>
<td>12 (X'C')</td>
<td>1</td>
<td>DT2X7LEV</td>
<td>Level number.</td>
</tr>
<tr>
<td>13 (X'D')</td>
<td>1</td>
<td>DT2X7FLG</td>
<td>Flag byte.</td>
</tr>
<tr>
<td>14 (X'0E')</td>
<td>2</td>
<td>DT2X7NF0</td>
<td>Number of format-0 DSCBs created.</td>
</tr>
<tr>
<td>16 (X'10')</td>
<td>5</td>
<td>DT2X7CSR</td>
<td>CCHHR cursor field. Used by the system.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Initialized to zero by caller—then do NOT modify.</td>
</tr>
<tr>
<td>21 (X'15')</td>
<td>3</td>
<td>DT2X7RES</td>
<td>Reserved.</td>
</tr>
</tbody>
</table>
Table 19. Format of ICVEDT02 Mapping Macro (continued)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Bytes</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>24 (X'18')</td>
<td>4</td>
<td>DT2X7RTA</td>
<td>Relative Track Address (RTA) cursor field. Used by the system. Initialized to zero by caller—then do NOT modify.</td>
</tr>
<tr>
<td>28 (X'1C')</td>
<td>4</td>
<td>DT2X7RE2</td>
<td>Reserved.</td>
</tr>
<tr>
<td>32 (X'20')</td>
<td>4</td>
<td>DT2X7ENT</td>
<td>Number of extent descriptor entries.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Extent descriptor array.</td>
<td></td>
</tr>
<tr>
<td>36 (X'24')</td>
<td>8</td>
<td>DT2ENTRY</td>
<td>One entry for each extent.</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>DT2RTAST</td>
<td>RTA of start of extent.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DT2RTAED</td>
<td>RTA+1 of end of extent.</td>
</tr>
</tbody>
</table>

**Accessing the DSCB Directly**

The CVAFDIR macro can be used to read or write one or more DSCBs and is described in [CVAFDIR Macro Overview and Specification](#) on page 74. After a CVAFDIR call, you can test the CV1IVT bit in the CVPL to determine whether the VTOC is indexed or nonindexed.

If the first 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 first 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 first 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 one or more DSCBs 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 first buffer. Do not use a 140-byte buffer as the first buffer. You can chain buffer list headers together, but at this time, CVAF only uses the first buffer list.

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.
Using the VTOC

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.

For CVAF calls that pass a data set name, a CVAFDIR request will fail if the EADSCB=OK indicator is not set and the DSCB associated with this data set name is a format-8 DSCB. CVAF return code of 4 with a CVAF status code (CVSTAT) of STAT082 will be set.

If you use CVAFDIR MULTIPLEDSCBS=YES ACCESS=READ, all DSCBs associated with this data set name are read and returned in the buffer list buffer (BFLEBUF) for entries without the skip bit on in logical VTOC order. The associated address of each DSCB is put in the corresponding BFLEARG field. The total number of DSCBs associated with this data set name is also returned in the BFLHNOEN field of the buffer list header. If there are not enough buffers to house all the DSCBs associated with this data set name, only those DSCBs that have BFLEs are processed. The buffer list header field BFLHNOEN will then have a number greater than the BFLHNOE field.

If you use CVAFDIR MULTIPLEDSCBS=YES ACCESS=WRITE, all BFLEs without the skip bit are used to write all DSCBs with one call. BFLARG must be correct for all entries (except the first 96-byte DSCB) and BFLEBUF must point to the correct corresponding DSCB to be written.

Specifying the DSCB Location
To read or write one or more DSCBs 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 at least one 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 at this time CVAF uses only the first buffer list.

If the first 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.

For CVAF calls that specify the DSCB location, a CVAFDIR request will fail if the EADSCB=OK indicator is not set and the DSCB associated with this location is a format-8 DSCB. CVAF return code of 4 with a CVAF status code (CVSTAT) of STAT082 will be set.

If you use CVAFDIR MULTIPLEDSCBS=YES ACCESS=READ, all DSCBs associated with the data set whose format-1 or format-8 DSCB is pointed to by the BFLEARG of the first BFLE entry are read and returned in the buffer list buffer (BFLEBUF) for entries without the skip bit on in logical VTOC order. The associated address of each DSCB is put in the corresponding BFLEARG field. The total number of DSCBs associated with this data set is also returned in the BFLHNOEN field of the buffer list header. If there are not enough buffers to house all the DSCBs associated
with this data set name, only those DSCBs that have BFLEs are processed. The buffer list header field BFLHNOEN will then have a number greater than the BFLHNOE field.

If you use CVAFDIR MULTIPLEDSCBS=YES ACCESS=WRITE, all BFLEs without the skip bit are used to write all DSCBs with one call. BFLARG must be correct for all entries and BFLEBUF must point to the correct corresponding DSCB to be written.

**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 information about the data sets:

- DSCBs in indexed (data set name) order (either format-1 and format-8 DSCBs or a format-4 DSCB)
- One or more DSCBs in physical-sequential order. (If you are unauthorized, you can request only 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 120 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, format-4, or format-8 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.

**Note:** The returned DSCB from CVAFSEQ always includes the key portion of the DSCB when the supplied length is 140 bytes for the buffer (field BFLELTH). The mapping for structure IECSDSF4 in macro IECSDSL1, however, does **not** include the key portion of the DSCB. Therefore, if you use the format-4 DSCB fields
Using the VTOC

defined in this structure (all fields starting with DS4), you must adjust the starting point of the first field in the IECSDSF4 structure to correspond to 44 bytes into the returned buffer.

For indexed access, a CVAFSEQ request will fail if the EADSCB=OK indicator is not set and the DSCB attempted to be read is a format-8 DSCB. CVAF return code of 4 with a CVAF status code (CVSTAT) of STAT082 will be set.

**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.

For physical-sequential access, a CVAFSEQ request to an extended address volume will fail if the EADSCB=OK indicator is not set. CVAF return code of 4 with a CVAF status code (CVSTAT) of STAT082 will be set.

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. See “CVAFFILT Macro Overview and Specification” on page 100 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 69 and “Partially-Qualified Names for CVAFFILT” on page 106 for further information.

- Identify a single DASD volume in the CVPL.

- For each data set that has a format-1 DSCB, the VTOC order returned by CVAFFILT is the format-1 DSCB, followed by any format 3 DSCBs.

  For each data set that has a format-8 DSCB, the VTOC order returned by CVAFFILT is the format-8 DSCB, one or more format-9 DSCBs, followed by any format-3 DSCBs. Currently, the minimum buffer list entry size needed to read all DSCBs associated with a data set is 12. This is one format-8 DSCB, one format-9 DSCB, and 10 format-3 DSCBs to reach the 123-extent limit.

- 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 72 and “Example of Using the CVAFFILT Macro” on page 106 for further information. CVAF filter service does not return a partial DSCB chain in the following cases:
  - 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” on page 68 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).
Using the VTOC

- For fully qualified data set names in the Filter Criteria List, a CVAFFILT request issued to an extended address volume will fail if the EADSCB=OK indicator is not set and the DSCB associated with the fully qualified data set name is described by a format-8 DSCB. For partially qualified data set names in the Filter Criteria List, a CVAFFILT request issued to an extended address volume will fail if the EADSCB=OK indicator is not set and a DSCB associated with a data set that matches the Filter Criteria List is described by a format-8 DSCB. In both these cases, the data set name status in the FCL (FCLDSNST) will be set to a status value of (x’06’). This status code indicates that a data set name is described by a format-8 DSCB and the caller did not specify support for it with the EADSCB=OK keyword. It will be set only when other data set name status codes are not applicable. CVAF status code (CVSTAT) of STAT086 (x’56’) will be set with these errors. This status code indicates all of the user FCL entries were processed, a resume is not required, one or more errors were found.

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 specifiedFLTAREA=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 7. Control Blocks Required for CVAF Filter Services](image)

---

Using the VTOC

- For fully qualified data set names in the Filter Criteria List, a CVAFFILT request issued to an extended address volume will fail if the EADSCB=OK indicator is not set and the DSCB associated with the fully qualified data set name is described by a format-8 DSCB. For partially qualified data set names in the Filter Criteria List, a CVAFFILT request issued to an extended address volume will fail if the EADSCB=OK indicator is not set and a DSCB associated with a data set that matches the Filter Criteria List is described by a format-8 DSCB. In both these cases, the data set name status in the FCL (FCLDSNST) will be set to a status value of (x’06’). This status code indicates that a data set name is described by a format-8 DSCB and the caller did not specify support for it with the EADSCB=OK keyword. It will be set only when other data set name status codes are not applicable. CVAF status code (CVSTAT) of STAT086 (x’56’) will be set with these errors. This status code indicates all of the user FCL entries were processed, a resume is not required, one or more errors were found.

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 7. Control Blocks Required for CVAF Filter Services](image)
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 Table 20 and Table 21 on page 70, are mapped by the ICVFCL macro. The format of the FCL header is shown in Table 20.

Table 20. Format of a Filter Criteria List Header

<table>
<thead>
<tr>
<th>Offset</th>
<th>Bytes</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (X'00')</td>
<td>4</td>
<td>FCLID</td>
<td>EBCDIC 'FCLbb' (bb here represents a blank.)</td>
</tr>
<tr>
<td>4 (X'04')</td>
<td>2</td>
<td>FCLCOUNT</td>
<td>Number of data set name entries provided in the list.</td>
</tr>
<tr>
<td>6 (X'06')</td>
<td>2</td>
<td>FCLDSCBR</td>
<td>Number of DSCBs returned.</td>
</tr>
<tr>
<td>8 (X'08')</td>
<td>1</td>
<td>FCL1FLAG</td>
<td>Request flag byte.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1....... FCL1LIST List contains fully-qualified data set names.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1....... FCL1ORDR FCL data set name order requested.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>.1...... FCL1EQF1 Return only format-1 or format-8 DSCBs.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>.1...... FCL1EQF9 Return only format-1 or format-8 and format-9 DSCBs.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>...xxxx Reserved.</td>
</tr>
<tr>
<td>9 (X'09')</td>
<td>1</td>
<td>FCL2FLAG</td>
<td>Status flag byte.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1....... FCL2SEQ CVAFFILT executed sequential VTOC access.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>.1...... FCL2SDIR CVAFFILT executed sequential VTOC access, but did at least one direct DSCB read.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>. xx xxxx Reserved.</td>
</tr>
<tr>
<td>10 (X'0A')</td>
<td>6</td>
<td>FCLDRSV</td>
<td>Reserved.</td>
</tr>
</tbody>
</table>

FCLID
Must be a 4-character EBCDIC constant of 'FCLbb'. (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 106 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).
Using the VTOC

**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:**
1. It can improve performance to allow CVAF to determine the sequence of return for format-1 DSCBs.
2. CVAF returns DSCBs for a given data set in format-1, format-3 order. For an extended address volume, the data set order for EAS eligible data sets is format-1, one or more format-9, and any format-3 DSCBs.
3. If you specify a single partially-qualified data set name, this field is not used.

**FCL1EQF1**
To have CVAF return only the format-1 or format-8 DSCBs for the data set names, set this bit to 1.

**FCL1EQF9**
To have CVAF return only the format-1 or format-8 and format-9 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 **Table 21**

**Table 21: Format of a Filter Criteria List Entry**

<table>
<thead>
<tr>
<th>Offset</th>
<th>Bytes</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>FCLDSN</td>
<td>Data set name information entry.</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>FCLDSNST</td>
<td>Data set name status.</td>
<td></td>
</tr>
<tr>
<td>X'00'</td>
<td></td>
<td>X'00'</td>
<td>Data set name not yet processed.</td>
</tr>
<tr>
<td>X'01'</td>
<td></td>
<td>X'01'</td>
<td>DSCBs returned successfully.</td>
</tr>
<tr>
<td>X'02'</td>
<td></td>
<td>X'02'</td>
<td>Data set name not found.</td>
</tr>
<tr>
<td>X'03'</td>
<td></td>
<td>X'03'</td>
<td>Error in DSCB chain. RESUME function recommended.</td>
</tr>
<tr>
<td>X'04'</td>
<td></td>
<td>X'04'</td>
<td>Error in CVAFFILT processing. RESUME not recommended.</td>
</tr>
<tr>
<td>X'05'</td>
<td></td>
<td>X'05'</td>
<td>Insufficient user buffer list elements. RESUME function recommended.</td>
</tr>
</tbody>
</table>
Using the VTOC

Table 21. Format of a Filter Criteria List Entry (continued)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Bytes</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'06'</td>
<td></td>
<td></td>
<td>Request issued to an EAV without EADSCB=OK specified and a format-8 DSCB was found.</td>
</tr>
<tr>
<td>X'07'</td>
<td></td>
<td></td>
<td>Request issued to any volume without EADSCB=OK specified where an EAS eligible data set was found.</td>
</tr>
</tbody>
</table>

1 (X'01') 1 FCLDSNLG Data set name length.
2 (X'02') 1 FCL3FLAG Flag byte.
1 . . . . . FCL3UPDT This data set name processed during this invocation.
. xxx xxxx Reserved.
3 (X'03') 1 FCLDSNRV Reserved.
4 (X'04') 4 FCLDSNA 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 Table 21 on page 70 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.
Using the VTOC

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 example below demonstrates the order that you might issue CVAFFILT macro calls to complete the following tasks:

• Request the DSCBs for a list of data sets.
• Resume CVAFFILT processing interrupted because of insufficient user buffers.
• 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 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:

<table>
<thead>
<tr>
<th>Buffer</th>
<th>Contents of Buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Format-1 DSCB, SYS1.A</td>
</tr>
<tr>
<td>2</td>
<td>Format-1 DSCB, SYS3.C</td>
</tr>
<tr>
<td>3</td>
<td>Format-3 DSCB, SYS3.C</td>
</tr>
<tr>
<td>4</td>
<td>Undefined (unused)</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>DSN</th>
<th>FCLDSNST</th>
<th>FCL3UPDT</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYS1.A</td>
<td>1</td>
<td>1</td>
<td>DSCBs returned from this call</td>
</tr>
<tr>
<td>SYS2.B</td>
<td>5</td>
<td>1</td>
<td>DSCB chain exceeds total buffers</td>
</tr>
<tr>
<td>SYS3.C</td>
<td>1</td>
<td>1</td>
<td>DSCBs returned from this call</td>
</tr>
<tr>
<td>SYS4.D</td>
<td>0</td>
<td>0</td>
<td>DSCBs can be returned by RESUME</td>
</tr>
<tr>
<td>SYS5.E</td>
<td>0</td>
<td>0</td>
<td>DSCBs can be returned by RESUME</td>
</tr>
<tr>
<td>SYS6.F</td>
<td>0</td>
<td>0</td>
<td>DSCBs can be returned by RESUME</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Buffer</th>
<th>Contents of Buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Format-1 DSCB, SYS4.D</td>
</tr>
<tr>
<td>2</td>
<td>Format-2 DSCB, SYS4.D</td>
</tr>
<tr>
<td>3</td>
<td>Format-3 DSCB, SYS4.D</td>
</tr>
<tr>
<td>4</td>
<td>Format-1 DSCB, SYS5.E</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>DSN</th>
<th>FCLDSNST</th>
<th>FCL3UPDT</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYS1.A</td>
<td>1</td>
<td>0</td>
<td>DSCBs returned from prior call</td>
</tr>
<tr>
<td>SYS2.B</td>
<td>5</td>
<td>0</td>
<td>DSCB chain exceeds total buffers</td>
</tr>
<tr>
<td>SYS3.C</td>
<td>1</td>
<td>0</td>
<td>DSCBs returned from prior call</td>
</tr>
<tr>
<td>SYS4.D</td>
<td>1</td>
<td>1</td>
<td>DSCBs returned from this call</td>
</tr>
<tr>
<td>SYS5.E</td>
<td>1</td>
<td>1</td>
<td>DSCBs returned from this call</td>
</tr>
<tr>
<td>SYS6.F</td>
<td>2</td>
<td>1</td>
<td>Data set name not found</td>
</tr>
</tbody>
</table>

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)

**Coding CVAF VTOC Access Macros**

This section 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.

- "CVAFDIR Macro Overview and Specification" on page 74
CVAF Macros

- “CVAFDSM Macro Overview and Specification” on page 94
- “CVAFFILT Macro Overview and Specification” on page 100
- “CVAFSEQ Macro Overview and Specification” on page 120
- “CVAFTST Macro Overview and Specification” on page 138
- “VTOC Index Error Message and Associated Codes” on page 139

CVAFDIR Macro Overview and Specification

For an indexed or nonindexed VTOC, you can use the CVAFDIR macro to perform the following functions:
- Read or write one or more DSCBs by specifying the name of the data set they represent.
- Read or write one or more DSCBs by specifying their addresses.

In addition, for an indexed VTOC, you can use the CVAFDIR macro to perform the following functions:
- 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 63 for additional information.

The format of the CVAFDIR macro is:
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 CVMRCD and CVIRCDS fields. If the CVMRCD or the CVIRCDS buffers are supplied in the BUFLIST field, either directly or indirectly through chaining, the keyword MAPRCD=YES, IXRCD=KEEP, or MAPRCD=(NO,0), IXRCD=(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**

```ebnf
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**

```ebnf
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**

```ebnf
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.

```ebnf
VERIFY=NO
```

CVAF does not test the key of the DSCB. This is the default.

**Restriction:** 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**

```ebnf
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.

**Recommendation:** Code the address of the UCB parameter as register (2-12). Coding an RX-Type address gives unpredictable results.
**CVAF Macros**

**Note:** You must supply a UCB address that matches the caller's AMODE. That is, AMODE=24 requires a 24 bit UCB address, while AMODE=31 requires a 31 bit UCB address.

**DEB=addr**
Supplies the address of a DEB opened to the volume table of contents (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 (neither APF nor in a system key), 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 57 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, upon completion of the CVAF request, the program should keep the CVAF I/O area associated with the CVAF parameter list. You can code IOAREA=KEEP 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 routine that calls CVAF.

When you code IOAREA=KEEP, it allows subsequent CVAF requests to be more efficient, because the program can bypass certain initialization functions. You do not need to specify either DEB or UCB when a previously obtained CVAF I/O area is supplied; you also cannot change those values.

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.
CVAF Macros

MAPRCD5: Keep or Free MAPRCD5 Buffer List and Buffers
This keyword applies to an indexed VTOC only and specifies the disposition of the MAPRCD5 buffer list and buffers.

MAPRCD5=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.

MAPRCD5=(YES, addr)
   If MAPRCD5=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 MAPRCD5 keyword.

   If MAPRCD5=YES is coded for a nonindexed VTOC, the function is performed, but an error code is returned.

MAPRCD5=NO
   If MAPRCD5=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.

MAPRCD5=(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 MAPRCD5=NO on the CVAFDIR macro that obtained the buffers
- By coding MAPRCD5=NO on a subsequent CVAF macro
- By coding CVAFDIR ACCESS=RLSE and providing the address of the buffer list in the BUFLIST keyword.

Requirement: 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.

**Requirement:** 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.

**EADSCB: Specify the support level for extended attribute DSCBs**

**EADSCB=OK**
This specification indicates that the calling program supports extended attribute DSCBs. An extended address volume may have these DSCBs.
The returned DSCBs (format-3, format-8) may contain extent descriptors described by 28-bit cylinder addresses or DSCBs (format-9) that contain additional attribute information.

For search calls where the data set name is passed (CVAFDIR ACCESS=READ,BFLEARG=0), a CVAFDIR request will fail if the EADSCB=OK indicator is not set and the DSCB associated with this data set name is a format-8 DSCB.

For seek calls where the record address is passed (CVAFDIR ACCESS=READ,BFLEARG=ccess), a CVAFDIR request issued to an EAV volume will be failed if the EADSCB=OK indicator is not set and the DSCB associated with this address is a format-8 or format-9 DSCB.

For seek calls where the record address is passed (CVAFDIR ACCESS=READ,BFLEARG=ccess), and MULTIPLEDSCBS=NO is specified or defaulted to NO, a CVAFDIR request will fail if the EADSCB=OK indicator is not set and the DSCB associated with this address is a format-3 DSCB that contain track addresses above 65,520 cylinders.

The failing error code for these cases will be reflected as follows:

- CVAF status code (CVSTAT) set to STAT082.
- Return code 4.

EADSCB=OK will set the CV4EADOK indicator in the CVPL.

For all other calls, the EADSCB=OK keyword is ignored.

**EADSCB=NOTOK**
- Indicates a calling program does not support extended attribute DSCBs. The specification of this will resolve to the CV4EADOK indicator in the CVPL to be set off. This is the default.

**MULTIPLEDSCBS: Specify whether multiple DSCBs should be processed**

**MULTIPLEDSCBS=NO**
- This specification indicates that the calling program requests that only one DSCB should be processed. This is the default for MF=L and MF=I forms of the CVAFDIR macro. When the MULTIPLEDSCBS keyword is not specified on the MF=E form, the existing setting of CV4MULTD is left unchanged. When MULTIPLEDSCBS=NO is specified or defaulted, only the first available buffer list entry is processed.

**MULTIPLEDSCBS=YES**
- This specification indicates that the calling program requests to read/write multiple DSCBs to/from a buffer list that contains more than one buffer list entry. This parameter causes an indicator in the CVPL, CV4MULTD, to be set on. Multiple DSCB processing for reads and writes is requested by specifying the MULTIPLEDSCBS=YES keyword and providing a buffer list that contains more than one buffer list entry (BFLHNOE>1).

**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.
CVAF Macros

**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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (X'00')</td>
<td>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 <a href="https://www.ibm.com/servers/zseries/dfsmsdfp/">DFSMSdfp Diagnosis</a>.)</td>
</tr>
<tr>
<td>4 (X'04')</td>
<td>An error occurred. The CVSTAT field in the CVPL contains an indication of the cause of the error. (CVSTAT code descriptions are in <a href="https://www.ibm.com/servers/zseries/dfsmsdfp/">DFSMSdfp Diagnosis</a>.)</td>
</tr>
<tr>
<td>8 (X'08')</td>
<td>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 <a href="https://www.ibm.com/servers/zseries/dfsmsdfp/">DFSMSdfp Diagnosis</a>.)</td>
</tr>
<tr>
<td>12 (X'0C')</td>
<td>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.</td>
</tr>
<tr>
<td>16 (X'10')</td>
<td>An I/O error was encountered.</td>
</tr>
</tbody>
</table>

**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.
CVAF Macros

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 BFLHL,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 BFLEL,BFLECHR CCHHR OF DSCB RETURNED BY CVAF
MVI BFLELTH,DSCLTH 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 8. Example of CVAFDIR Macro with VTOC Part 1 of 2
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 as well as the return code received back from CVAFDIR. 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 4-word entries. The format of each 4-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 fourth word contains the return code received back from CVAFDIR for 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 (SUPPLIED)
RDSN EQU 6 ADDRESS OF DATA SET NAME
RTTR EQU 7 ADDRESS OF TTR
REGIS EQU 15 RETURN CODE REGISTER 15
*
****************************************************************************************
* UCB ADDRESS SUPPLIED IN RUCB.
* READ DSCB OF DATA SET NAME SUPPLIED.
* RETURN TTR OF DSCB.
* RETURN RETURN CODE FOR CVAFDIR REQUEST FOR DSCB.
* 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
* WORD 4 OF PARAMETER LIST = RETURN CODE RETURNED FROM CVAFDIR FOR DSCB
* WORDS 1-4 CAN BE DUPLICATED FOR MULTIPLE REQUESTS
* THE HIGH ORDER BIT OF WORD 3 SET TO X'80' FOR THE LAST ENTRY ONLY.
*
****************************************************************************************
* USING LISTMAP,RLIST ADDRESSABILITY TO PARMLIST
TOLOOP EQU * LOOP FOR EACH DSCB
XC BUFLIST(BFLHLN+BFLLEN),BUFLIST ZERO BUFFER LIST
OI BFLHLN,BFLHDSCB DSCBS TO BE READ WITH BUFFER LIST
MVI BFLHNOE,1 ONE BUFFER LIST ENTRY
L 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
Example of Using the CVAFDIR macro to read multiple DSCBs

This example uses the CVAFDIR macro to read multiple DSCBs for a VSAM data set that has 123 extents using the MULTIPLEDSCBS=YES parameter. Refer to the documentation within the sample source for program logic and expected output.

The following is the sample JCL used to execute the sample source module below.

```bash
//*
//******************************************
//* JCL TO EXECUTE CVDIR027 MODULE *
//******************************************
```
The following is the CVAFDIR sample source to read multiple DSCBs.

CVDIR027 CSECT
CVDIR027 AMODE 31
CVDIR027 RMODE 24

***********************************************************************
** CVDIR027 - MODULE THAT ISSUES THE CVAFDIR MACRO WHICH RETURNS **
** THE DSCBS FOR A GIVEN DATASET USING THE KEYWORDS **
** MULTIPLEDSCBS=YES AND EADSCB=OK. **
** THIS MODULE USES A PASSED DSN (SEARCH) AND ISSUES THE **
** CVAFDIR MACRO TO PERFORM A READ OF ALL DSCBS FOR THE **
** DATASET. THE DATASET PROCESSED IS VSAM AND HAS 123 **
** EXTENTS. THE DSN IS CVAFDIR1.VSAM01.DATA. **
** THE CVAFDIR MACRO CALL WILL USE THE FOLLOWING: **
** EADSCB=OK CODED **
** MULTIPLEDSCBS=YES CODED **
** THIS MODULE HAS BEEN WRITTEN TO RUN AT THE Z/OS 1.10 **
** LEVEL AND WILL CREATE A SLIGHTLY DIFFERENT OUTPUT **
** REPORT DEPENDENT UPON DEVICE TYPE (EAV OR NON EAV). **
** THE NUMBER OF BUFFER LIST ENTRIES NEEDED WILL BE 11 **
** FOR A NON EAV DEVICE AND THE NUMBER OF BUFFER LIST **
** ENTRIES NEEDED WILL BE 12 FOR AN EAV DEVICE TO **
** ACCOUNT FOR THE FORMAT 9 DSCB. **
** THIS MODULE WILL CREATE AN OUTPUT REPORT DIRECTED TO **
** THE OUTDD DD THAT SHOULD LOOK LIKE THE FOLLOWING: **
**---------------------------------------------------------------------**
** NON EAV VOLUME **
---------- **
** CVDIR027 START OF OUTPUT MESSAGES **
** PROCESSING DSN: CVAFDIR1.VSAM01.DATA **
** CVAFDIR CALL: EADSCB=OK AND MULTIPLEDSCBS=YES CODED **
** CV4EADOK BIT SET / EADSCB=OK **
** CV4MULTD BIT SET / MULTIPLEDSCBS=YES **
** RC00 VERIFIED - THE REQUEST WAS SUCCESSFUL **
** X"00" DEC"000" 00 - CVSTAT CODE VERIFIED **
** BUFFER LIST ENTRIES PROVIDED: 12 **
** BUFFER LIST ENTRIES NEEDED : 11 **
** CVDIR027 END OF OUTPUT MESSAGES **
**---------------------------------------------------------------------**
** EAV VOLUME **
---------- **
** CVDIR027 START OF OUTPUT MESSAGES **
CVAF Macros

PROCESSING DSN: CVAFDIR1.VSAM01.DATA
CVAFDIR CALL: EADSCB=OK AND MULTIPLEDSCBS=YES CODED
CV4EADOK BIT SET / EADSCB=OK
CV4MULTD BIT SET / MULTIPLEDSCBS=YES
RC00 VERIFIED - THE REQUEST WAS SUCCESSFUL
X"00" DEC"000" 00 - CVSTAT CODE VERIFIED
BUFFER LIST ENTRIES PROVIDED: 12
BUFFER LIST ENTRIES NEEDED : 12

CVDIR027 END OF OUTPUT MESSAGES

*****************************************************************************

CVDIR027 - LOGIC NOTES

THIS MODULE WILL PERFORM THE FOLLOWING:

INITIALIZATION
- OBTAIN THE NECESSARY INFORMATION FROM THE DASD VOLUME
- OPEN AN OUTPUT FILE AND WRITE THE NECESSARY OUTPUT MESSAGES

MAINLINE
- INVOKE SETU1RTN TO SETUP BUFFER LIST FOR 12 ENTRIES
- INVOKE VSAM1RTN TO PROCESS VSAM01 DSN
- INVOKE READRTN - EADSCB=OK AND MULTIPLEDSCBS=YES CODED
- CHECK CV4FL BIT SETTINGS AFTER CVAFDIR CALL
- CHECK RC AND CVSTAT CODES RETURNED
- CHECK BUFFERS PROVIDED AND BUFFERS NEEDED FROM BUFFER LIST
- WRITE OUTPUT RECS
- ISSUE CVAFDIR TO RELEASE WORK AREAS
- WRITE OUTPUT RECS

FINALIZATION
- CLOSE THE OUTPUT FILE
- EXIT

CVDIR027 - JOB INFORMATION

NORMAL END OF JOB:
- RC=00 AND OUTDD OUTPUT AS DETAILED ABOVE

ABNORMAL END OF JOB:
- ABEND 100 - ERROR OPENING VTOC ON THE DASD VOLUME THAT IS
  ASSOCIATED WITH THE CVAFDD DD STATEMENT
- ABEND 101 - ERROR OPENING THE OUTDD DATASET
- ABEND 102 - ERROR CLOSING THE OUTDD DATASET

*****************************************************************************

HOUSEKEEPING
- SAVE CALLER'S REGISTERS AND ESTABLISH A NEW REGISTER SAVE AREA

*****************************************************************************

STM R14,R12,12(R13) STANDARD LINKAGE CONVENTION
BALR R11,0 DCL R11 AS IMPLIED BASE REG
USING BASE,R11,R12 R12 IS ALSO BASE REG
**BASE ADDR DC A(BASE+4096) ADDRESSING FOR R12**

**BASE L R12,BASEADDR SET UP ADDRESSING FOR R12**

**CV000000 DS 0H CONTINUE...**

**ST R13,SAVE+4 SAVE PTR TO CALLER'S SAVE AREA**

**LA R14,SAVE GET ADDRESS OF THE NEW SAVE AREA**

**ST R14,8(R13) CHAIN CALLER'S AREA TO OURS**

**LR R13,R14 ESTABLISH THE NEW SAVE AREA**

***********************************************************************
** INITIALIZATION **
***********************************************************************

**INITIAL DS 0H INITIALIZATION SECTION**

**BAL R14,IDVOLRTN INVOKE RTN TO IDENTIFY THE VOLUME**

**OPEN (OUTFILE,(OUTPUT)) OPEN THE OUTDD OUTPUT FILE**

**TM OUTFILE+48,X'10' TEST IF FILE IS OPEN (OUTFILE)**

**BO INIT0010 IF OPEN OK - BRANCH AROUND ABEND**

**ABEND 101 ELSE ISSUE USER ABEND 101**

**INIT0010 DS 0H FILE IS OPEN WRITE START MESSAGE**

**PUT OUTFILE,STRTMSG WRITE A RECORD TO THE OUTPUT FILE**

**PUT OUTFILE,BLNKLINE WRITE A RECORD TO THE OUTPUT FILE**

***********************************************************************
** MAINLINE **
***********************************************************************

**MAINLINE DS 0H MAINLINE SECTION**

**MAIN0010 DS 0H PROCESS CVAFDIR1.VSAM01.DATA DATASET**

**BAL R14,SETU1RTN SETUP FOR CVAFDIR - 12 ENTRIES**

**BAL R14,VSAM1RTN PROCESS VSAM01 DATASET ROUTINE**

**PUT OUTFILE,BLNKLINE WRITE A RECORD TO THE OUTPUT FILE**

***********************************************************************
** FINALIZATION **
***********************************************************************

**FINAL DS 0H FINALIZATION SECTION**

**PUT OUTFILE,ENDMSG WRITE A RECORD TO THE OUTPUT FILE**

**CLOSE (OUTFILE) CLOSE OUTPUT FILE**

**C R15,RCODE00 IF FILE CLOSE IS OK**

**BE FINL0010 BRANCH AROUND ABEND**

**FINL0010 DS 0H EXIT MODULE**

**L R13,4(R13) RESTORE REGISTER**

**LM R14,R12,12(R13) RESTORE CALLERS REGISTERS**

**LA R15,0 SET RC TO 0**

**BR R14 RETURN TO CALLER**

***********************************************************************
** IDVOLRTN **
***********************************************************************

**- OBTAIN THE NECESSARY INFORMATION FROM THE DASD VOLUME**

***********************************************************************
** IDVOLRTN DS 0H IDENTIFY VOLUME ROUTINE**

**ST R14,IDVLSAVE STORE C(R14) INTO SAVE AREA**

**RDJFCB (VT0CDBC,(INPUT)) READ JFCB / OPEN VTOC**

**MVI JFCB1,X'04' PUT IN ID FOR FORMAT 4**
Chapter 1. Using the Volume Table of Contents

CVAF Macros

MVC JFCB1+1(43),JFCB1 SETUP FOR VTOC OPEN
OPEN (VTOCDCB,(INPUT)),TYPE=J OPEN VTOC (OPEN TYPE=J)
TM VTOCDCB+48,X'10' IF OPEN OF VTOC IS OK
BO IDVL0010 BRANCH AROUND ABEND
ABEND 100 ELSE ISSUE USER ABEND 100

IDVL0010 DS 0H
SLR R4,R4 INIT REG4 FOR DEB PTR
SLR R5,R5 INIT REG5 FOR UCB PTR
ICM R4,B'0111',VTOCDCB+45 GET DEB ADDRESS
ST R4,DEBADD STORE C(R4) INTO DEBADD
ICM R5,B'0111',33(R4) GET UCB ADDRESS
ST R5,UCBADD STORE UCB ADDRESS

IDVLEXIT DS 0H EXIT FROM IDVL0RTN
L R14,IDVLSAVE LOAD C(IDVLSAVE) INTO R14
BR R14 EXIT

***********************************************************************
* SETU1RTN *
* - SETUP FOR CVAFDIR - BFLHNOE = 12 *
* - WILL SETUP 12 ENTRIES FOR CVAFDIR CALL *
* - CVAFDIR READ CALL (SEARCH) PASSED DSN *
* - 96 BYTE BUFFER - 1ST BUFFER *
* - 140 BYTE BUFFERS - REMAINING BUFFERS *
* - DSNAME=DSN WILL BE USED FOR SEARCH *
* - CCHHR = ZERO *
***********************************************************************

SETU1RTN DS 0H SETUP FOR CVAFDIR CALL
ST R14,SET1SAVE STORE C(R14) INTO SAVE AREA
LA R4,BUFLHDR GET ADDR OF BUF LIST HEADER
L R5,NBRENT LOAD R5 WITH NBR OF ENTRIES (12)
USING BFLHDR,R4 GET ADDRESSABILITY TO HEADER
MVI BFLHNOE,TWELVE INDICATE 12 ENTRIES
MVI BFLHKEY,BFLHDSCB INDICATE READ DSCB
LA R6,DSCBBUF LOAD R6 WITH ADDR OF 1ST DSCB BUFFER
LA R7,BUFLIST1 LOAD R7 WITH ADDR OF BUFLIST1
USING BFLE,R7 GET ADDRESSABILITY TO ENTRY

* INITIALIZE 1ST ENTRY
*
OI BFLEFL,BFLECHR INDICATE CCHHR TO BE READ
MVC BFLEARG(5),CCHHR0 SET ZEROES FOR ARGUMENT
MVI BFLELTH,DSCBL96 GET LENGTH OF BUFFER
ST R6,BFLEBUF PUT DSCB BUF ADDR IN ENTRY
LA R7,ENTLENG(,R7) INCREMENT ADDR TO NEXT ENTRY
LA R6,DSCBL96(,R6) INCREMENT ADDR TO NEXT DSCB BUFFER
S R5,ONE C(R5) = C(R5) - 1

* INITIALIZE REMAINING ENTRIES
*
SETU0010 DS 0H INIT REMAINING 140 BYTE BUFFERS
OI BFLEFL,BFLECHR INDICATE CCHHR TO BE READ
MVI BFLELTH,DSCBL140 GET LENGTH OF BUFFER
ST R6,BFLEBUF PUT DSCB BUF ADDR IN ENTRY
LA R7,ENTLENG(,R7) INCREMENT ADDR TO NEXT ENTRY
LA R6,DSCBL140(,R6) INCREMENT ADDR TO NEXT DSCB BUFFER
BCT R5,SETU0010 BRANCH TO SETU0010 IF MORE ENTRIES

SETIEXIT DS 0H EXIT FROM SETU1RTN
L R14,SET1SAVE LOAD C(SAVE AREA) INTO R14
BR R14 EXIT

***********************************************************************
* VSAM1RTN *
* - PROCESS VSAM01 DATASET ROUTINE *
* - CVAFDIR READ: EADSCB=OK AND MULTIPLESBCBS=YES CODED *
* - CHECK RC / CVSTAT CODES *
* - ISSUE CVAFDIR RELEASE *
***********************************************************************
VSAMRTN DS 0H PROCESS VSAM01 DATASET ROUTINE
ST R14,VSAMSAVE STORE C(R14) INTO SAVE AREA
MVC DSNNAME(44),VSAM01 DSNNAME-CVAFDIR1.VSAM01.DATA
PUT OUTFILE,VSAM1MSG WRITE A RECORD TO THE OUTPUT FILE
PUT OUTFILE,CALLMR11 WRITE A RECORD TO THE OUTPUT FILE
BAL R14,READRTN READRTN - EADSCB=OK/MULTIPLEDSCBS=YES
L R15,RETCODE LOAD R15 WITH SAVED RETURN CODE
C R15,RCODE00 IF RC FROM READ IS EQUAL TO ZERO
BE VSAM0010 BRANCH TO VSAM0010
PUT OUTFILE,UNEXPMS2 ELSE WRITE RC ERROR MESSAGE
B VSAM0020 BRANCH TO VSAM0020
VSAM0010 DS 0H
PUT OUTFILE,RC00MSG WRITE RC00 MESSAGE
VSAM0020 DS 0H
LA R9,CVAFDIR GET ADDRESS OF CVAF PARM LIST
USING CVPL,R9 GET ADDRESSABILITY TO CVPL
CLI CVSTAT,STAT000 IF CVSTAT FROM READ IS EQUAL TO ZERO
BE VSAM0030 BRANCH TO VSAM0030
PUT OUTFILE,UNEXPMS3 ELSE WRITE CVSTAT ERROR MESSAGE
B VSAM0040 BRANCH TO VSAM0040
VSAM0030 DS 0H
PUT OUTFILE,CV00MSG WRITE CV00 MESSAGE
VSAM0040 DS 0H
CLI BFLHNOE,X'0C' IF NUMBER OF BUFFERS PROVIDED = 12
BE VSAM0050 BRANCH TO VSAM0050
CLI BFLHNOEN,X'0B' IF NUMBER OF BUFFERS NEEDED = 11
CLI BFLHNOEN,X'0C' IF NUMBER OF BUFFERS NEEDED = 12
BE VSAM0090 BRANCH TO VSAM0090
PUT OUTFILE,UNEXPMS4 ELSE WRITE BUFFER ERROR MESSAGE
B VSAM0060 BRANCH TO VSAM0060
VSAM0050 DS 0H
PUT OUTFILE,BUFSMSG1 WRITE BUFFER MESSAGE
VSAM0060 DS 0H
CLI BFLHNOEN,X'0B' IF NUMBER OF BUFFERS NEEDED = 11
BE VSAM0070 BRANCH TO VSAM0070
CLI BFLHNOEN,X'0C' IF NUMBER OF BUFFERS NEEDED = 12
BE VSAM0080 BRANCH TO VSAM0080
PUT OUTFILE,UNEXPMS5 ELSE WRITE BUFFER ERROR MESSAGE
B VSAM0090 BRANCH TO VSAM0090
VSAM0070 DS 0H
PUT OUTFILE,BUFNMSG1 WRITE BUFFER MESSAGE
B VSAM0090 BRANCH TO VSAM0090
VSAM0080 DS 0H
PUT OUTFILE,BUFSMSG2 WRITE BUFFER MESSAGE
VSAM0090 DS 0H
* CVAFDIR ACCESS=RLSE,IXRCS=NOKEEP,BUFLIST=0,
  MF=(E,CVAFDIR)
C R15,RCODE00 IF RC FROM RLSE IS EQUAL TO ZERO
BE VSAM0050 BRANCH TO VSAM0050
PUT OUTFILE,UNEXPMS6 ELSE WRITE RLSE ERROR MESSAGE
* VSAMEXIT DS 0H EXIT FROM ROUTINE
DROP R9 DROP R9
L R14,VSAMSAVE LOAD C(SAVE AREA) INTO R14
BR R14 EXIT
* READDRTN *
* - CVAFDIR READ ROUTINE *
* MULTIPLEDSCBS=YES IS CODED *
* EADSCB=OK IS CODED *
* - CHECK CV4FL BIT SETTINGS AFTER READ *
* - REPORT ON CV4FL BIT SETTINGS FOR CV4EADOK AND CVMULTD *
* READDRTN DS 0H CVAFDIR READ ROUTINE
ST R14,READSAVE STORE C(R14) INTO SAVE AREA
L R2,DEBADD LOAD R2 WITH DEB ADDRESS

* CVAFDIR ACCESS=READ,DEB=(R2),BUFLIST=BUFLHDR,MAPRCDS=YES, X
  DSN=DSNAME,MULTIPLEDSCBS=YES,EDASCB=OK, X
  MF=(E,CVAFDIR)
*
ST R15,RETCODE STORE RC INTO RETCODE

* LA R9,CVAFDIR GET ADDRESS OF CVAF PARM LIST
USING CVPL,R9 GET ADDRESSABILITY TO CVPL
CLI CVFL4,CV4EADOK IF CVFL4 = X'10' CV4EADOK ONLY
BE READ0010 BRANCH TO READ0010
CLI CVFL4,CV4MULTD IF CVFL4 = X'08' CV4MULTD ONLY
BE READ0020 BRANCH TO READ0020
CLI CVFL4,BOTH IF CVFL4 = X'18' CV4MULTD/CV4EADOK
BE READ0030 BRANCH TO READ0030
PUT OUTFILE,UNEXPMS1 ELSE WRITE ERROR RECORD PL
  B READEXIT BRANCH TO EXIT ROUTINE
READ0010 DS 0H WRITE MESSAGES X'10'
PUT OUTFILE,OKMSG WRITE OK MSG RECORD
PUT OUTFILE,NOMULTMS WRITE NO MULTI MSG RECORD
  B READEXIT BRANCH TO EXIT ROUTINE
READ0020 DS 0H WRITE MESSAGES X'08'
PUT OUTFILE,NOTOKMSG WRITE NOTOK MSG RECORD
PUT OUTFILE,MULTMSG WRITE MULTI MSG RECORD
  B READEXIT BRANCH TO EXIT ROUTINE
READ0030 DS 0H WRITE MESSAGES X'18'
PUT OUTFILE,OKMSG WRITE OK MSG RECORD
PUT OUTFILE,MULTMSG WRITE MULTI MSG RECORD
PUT OUTFILE,MULTMSG WRITE MULTI MSG RECORD
READEXIT DS 0H EXIT FROM READRTN
DROP R9 DROP R9
L R14,READSAVE LOAD C(SAVE AREA) INTO R14
BR R14 EXIT

***********************************************************************
* WORKING STORAGE *
***********************************************************************
*
DS 0D
DC CL36'CVDIR027-WORKING STORAGE BEGINS HERE'
*
***********************************************************************
* EQUATES *
***********************************************************************
*
R0 EQU 0
R1 EQU 1
R2 EQU 2
R3 EQU 3
R4 EQU 4
R5 EQU 5
R6 EQU 6
R7 EQU 7
R8 EQU 8
R9 EQU 9
R10 EQU 10
R11 EQU 11
R12 EQU 12
R13 EQU 13
R14 EQU 14
R15 EQU 15
BOTH EQU X'18'
TWELVE EQU X'0C'
*
***********************************************************************
* SAVE AREAS *
***********************************************************************
**CVAF Macros**

```
***********************************************************************
* SAVE DC 18'0' MAIN PROGRAM SAVE AREA
IDVLSAVE DC F'0' IDENTIFY VOLUME ROUTINE SAVE AREA
READSAVE DC F'0' READ ROUTINE SAVE AREA
SETISAVE DC F'0' SETUP ROUTINE SAVE AREA
VSAMSAVE DC F'0' VSAM01 ROUTINE SAVE AREA
*
***********************************************************************
* CONSTANTS *
***********************************************************************
*
CCHHR0 DS XL5'00' CCHHR ARGUMENT - ZER0
NBRENT DC F'12' CONSTANT-12 NUMBER OF BUFFER ENTRIES
RCODE00 DC F'0' RETURN CODE 0
ONE DC F'1' CONSTANT - ONE
VSAM01 DC CL44'CVAFDIR1.VSAM01.DATA'
*
***********************************************************************
* PROGRAM BUFFERS *
***********************************************************************
*
BUFLIST DS 0F BUFFER LIST DECLARATIONS
BUFLHDR DC 2F'0' BUFFER LIST HEADER
BUFLIST1 DC 3F'0' BUFFER LIST ENTRY 1
ENTLENG EQU =-BUFLIST1 ENTRY LENGTH - 12
BUFLIST2 DC 3F'0' BUFFER LIST ENTRY 2
BUFLIST3 DC 3F'0' BUFFER LIST ENTRY 3
BUFLIST4 DC 3F'0' BUFFER LIST ENTRY 4
BUFLIST5 DC 3F'0' BUFFER LIST ENTRY 5
BUFLIST6 DC 3F'0' BUFFER LIST ENTRY 6
BUFLIST7 DC 3F'0' BUFFER LIST ENTRY 7
BUFLIST8 DC 3F'0' BUFFER LIST ENTRY 8
BUFLIST9 DC 3F'0' BUFFER LIST ENTRY 9
BUFLISTA DC 3F'0' BUFFER LIST ENTRY 10
BUFLISTB DC 3F'0' BUFFER LIST ENTRY 11
BUFLISTC DC 3F'0' BUFFER LIST ENTRY 12
*
DSCBBUF DS XL96 DSCB BUFFER DECLARATION 1 - 96 BYTE
DSCBL96 EQU =-DSCBBUF LENGTH - 96
DSCBBUF2 DS XL140 DSCB BUFFER DECLARATION 2 - 140 BYTE
DSCBL140 EQU =-DSCBBUF2 LENGTH - 140
DSCBBUF3 DS XL140 DSCB BUFFER DECLARATION 3 - 140 BYTE
DSCBBUF4 DS XL140 DSCB BUFFER DECLARATION 4 - 140 BYTE
DSCBBUF5 DS XL140 DSCB BUFFER DECLARATION 5 - 140 BYTE
DSCBBUF6 DS XL140 DSCB BUFFER DECLARATION 6 - 140 BYTE
DSCBBUF7 DS XL140 DSCB BUFFER DECLARATION 7 - 140 BYTE
DSCBBUF8 DS XL140 DSCB BUFFER DECLARATION 8 - 140 BYTE
DSCBBUF9 DS XL140 DSCB BUFFER DECLARATION 9 - 140 BYTE
DSCBBUFA DS XL140 DSCB BUFFER DECLARATION 10- 140 BYTE
DSCBBUB DS XL140 DSCB BUFFER DECLARATION 11- 140 BYTE
DSCBBUFC DS XL140 DSCB BUFFER DECLARATION 12- 140 BYTE
*
***********************************************************************
* PROGRAM MESSAGES *
***********************************************************************
*
BLNKLINE DC CL80' '
STRTMSG DC CL80'CVDIR027 START OF OUTPUT MESSAGES '
ENDMSG DC CL80'CVDIR027 END OF OUTPUT MESSAGES '
VSAMMSG DC CL80' PROCESSING DSN: CVAFDIR1.VSAM01.DATA '
CALLMSG DC CL80' CALL: EADSCB=OK AND MULTIPLEDSCBS=YES CODED'
OKMSG DC CL80' CV4EADOK BIT SET / EADSCB=OK '
NOTOKMSG DC CL80' CV4EADOK BIT NOT SET / EADSCB=NOTOK '
MULTMSG DC CL80' CV4MULTD BIT SET / MULTIPLEDSCBS=YES '
NOMULTMS DC CL80' CV4MULTD BIT NOT SET / MULTIPLEDSCBS=NO '
RC00MSG DC CL80' RC00 VERIFIED - THE REQUEST WAS SUCCESSFUL '
```
CVAF Macros

CV00MSG DC CL80' X"00" DECO"000" 00 - CVSTAT CODE VERIFIED
BUFSMSG1 DC CL80' BUFFER LIST ENTRIES PROVIDED: 12
BUFNMSG1 DC CL80' BUFFER LIST ENTRIES NEEDED : 11
BUFNMSG2 DC CL80' BUFFER LIST ENTRIES NEEDED : 12
UNEXPMS1 DC CL80' ERROR: UNEXPECTED BIT SETTING FOR CVFL4
UNEXPMS2 DC CL80' ERROR: UNEXPECTED RETURN CODE FROM CVAFDIR READ
UNEXPMS3 DC CL80' ERROR: UNEXPECTED CVSTAT CODE FROM CVAFDIR READ
UNEXPMS4 DC CL80' ERROR: UNEXPECTED NUMBER OF BUFFERS PROVIDED
UNEXPMS5 DC CL80' ERROR: UNEXPECTED NUMBER OF BUFFERS NEEDED
UNEXPMS6 DC CL80' ERROR: UNEXPECTED RETURN CODE FROM CVAFDIR RLSE
*
***********************************************************************
* WORK AREAS *
***********************************************************************
*
DEBADD DC F'0' DEB ADDRESS SAVE AREA
UCBADD DC F'0' UCB ADDRESS SAVE AREA
RETCODE DC F'999' RETURN CODE SAVE AREA
DSNAME DS CL44
*
***********************************************************************
* DCB - OUTPUT FILE (OUTFILE) *
***********************************************************************
*
OUTFILE DCB DDNAME=OUTDD, X
                  DSORG=PS, X
                  RECFM=FB, X
                  LRECL=80, X
                  MACRF=PM
*
***********************************************************************
* VTOC DCB AREA *
***********************************************************************
*
VTOCDB DCB DDNAME=CVAFDD,MACRF=E,EXLST=XLST1,DSORG=PS,DCBE=VTOCDCBE
XLST1 DC X'87'
       DC AL3(JFCB1)
JFCB1 DS OCL176
TESTNAME DS CL44
          DS CL8
          DS BL1
          DS CL123
VTOCDCBE DCBE EACSCB=OK
*
***********************************************************************
* CVAF DECLARATIONS *
***********************************************************************
*
CVAFDIR CVAFDIR MF=L
*
***********************************************************************
* MAPPING MACROS *
***********************************************************************
*
ICVAFPL ICVAFBFL
DSECT
IECSDSL1 (1,3,8,9)
*
*
END CVDIR027 END OF CVDIR027
/*
CVAF Macros

CVAFDSM Macro Overview and Specification

The CVAFDSM macro is used to obtain volume information for an indexed or nonindexed 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.
- A value that represents the highest allocated DSCB as determined by the VTOC INDEX.

The CVAFDSM macro can be used for a nonindexed VTOC to obtain:
- One or more extents that describe unallocated space on the volume

The format of the CVAFDSM macro is:

```
/SM590000/SM590000
CVAFDSM
label
ACCESS=MAPDATA
VOLUME
VTOC
EXTENTS=addr
/SM590000/SM590000
NO
RTA4BYTE=YES,
MAPRCDS=YES
(YES,
addr)
NO
(YES,
addr)

/SM590000/SM590000
NO
COUNT=YES,
CTAREA=addr
HADSCB=addr
/SM590000/SM590000

/SM590000/SM590000
NOKEEP,
KEEP
(KEEP,
addr)
(NOKEEP,
addr)
/SM590000/SM590000
NO
BRANCH,
YES
(YES,
SUP)
(YES,
PGM)
/SM590000/SM590000

/SM590000/SM590000
MF=(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:
CVAF Macros

- 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 the following occurs:

- Information about free extents uses mapping macro ICVEDT02 (see "Using Macro ICVEDT02 to Map the Extents Area" on page 62 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 affect 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.
CVAF Macros

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 End Of Data is reached before the provided storage area is filled with unallocated extents, CVAF will set return code 4 in register 15. If return code 0 is set in register 15, you can call CVAF again to get the remaining unallocated space information if there is any. 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 the following occurs:

- 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 (XXXYZZ) 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.

Recommendation: If you use larger volumes, specify RTA4BYTE=YES when you request extent information. If an extent is beyond the 64x1024 tracks boundary when the program specifies RTA4BYTE=NO or allows the default, the CVAF request fails with a CVSTAT of STAT075.

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 Table 19 on page 62 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      The relative track address of the first track of the extent
YY      The number of whole cylinders in the extent
Z       The number of additional tracks in the extent.

If you do not specify the RTA4BYTE parameter, the default is RTA4BYTE=NO.

MAPRCD: Keep or Free MAPRCD Buffer List and Buffers

MAPRCD=YES
Specifies that the buffer list and buffers are to be retained at the end of the function.

If MAPRCD=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.

MAPRCD=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.

MAPRCD=(YES, addr)
If MAPRCD=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 MAPRCD keyword.

MAPRCD=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.

MAPRCD=(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 MAPRCD=NO on the call that obtained the buffers
• By coding MAPRCD=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 MAPRCD is not coded, the parameter list value of MAPRCD is not changed.

Requirement: 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
CVAF Macros

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 57 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.

HADSCB: Supply a Field to Contain the CCHHR of the Highest Allocated DSCB

HADSCB=addr
Supplies the address of a 5-byte field to contain the CCHHR of the highest allocated DSCB in the VTOC when COUNT=YES and MAP=VTOC are 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.

EADSCB: Specify the support level for extended attribute DSCBs

EADSCB=OK
This specification indicates that the calling program supports RTAs that could contain tracks an extended address volume.

For calls that request unallocated space (ACCESS=MAPDATA, MAP=VOLUME, RTA4BYTE=YES and EXTENTS=address), a CVAFDSM request issued to an EAV volume will be failed if the EADSCB=OK indicator is not set.
The failing error code for these cases will be reflected as follows:

- CVAF status code (CVSTAT) set to STAT082.
- Return code 4.

EADSCB=OK will set the CV4EADOK indicator in the CVPL. All other calls to CVAFDSM are allowed and EADSCB=OK will be ignored.

EADSCB=NOTOK
Indicates a calling program does not support extended attribute DSCBs. EADSCB=NOTOK will set the CV4EADOK indicator in the CVPL to off. 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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (X'00')</td>
<td>The request was successful.</td>
</tr>
<tr>
<td>4 (X'04')</td>
<td>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 z/OS DFSMSdip Diagnosis)</td>
</tr>
<tr>
<td>8 (X'08')</td>
<td>Invalid VTOC index structure. CVSTAT contains an indication of the cause of the error. (CVSTAT code descriptions are in z/OS DFSMSdip Diagnosis)</td>
</tr>
</tbody>
</table>
| 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 Table 20 on page 69 and
Table 21 on page 70. The format of the buffer list is shown in “Using Buffer Lists”
on page 60. CVAFFILT returns a complete set of DSCBs in the order that they are
chained in the VTOC (format-1, format-2, then format-3).

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 67 for additional
information.

The format of the CVAFFILT macro is:

```
>> label
   CVAFFILT, ACCESS= READ, RESUME, BUFLIST= addr, (reg)
   , UCB= (ucbaddr), DEB= addr, FCL= addr, (reg)
   , NOKEEP, IOAREA= KEEP, (addr), (reg)
   , NOTOK, EADSCB= OK, , MF= 0
```

Restriction: 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.
CVAF Macros

- When you specify any control block’s address as \textit{(reg)}, the CVAFFILT macro assumes that the register specified contains that address.
- When you specify a “defined” control block’s address as \textit{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.
- When you specify a “referenced” control block’s address as \textit{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.\(^\leftarrow\) See the example of partially-qualified names in \textit{Partially-Qualified Names for CVAFFILT} on page 106.

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 \textit{Codes Put in the CVSTAT Field} on page 140.

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 Table 20 on page 69.

Always test the status byte of each data set name in the FCL list to ensure successful completion.\(^\leftarrow\) Some error conditions result in failure to return a data set’s DSCBs. See the FCLDSNST byte in Table 21 on page 70.

**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 Table 17 on page 61 and Table 18 on page 62 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 16 MB 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 CVAFFILT removes the DEB from the DEB table for the duration of the CVAFFILT call. See “Identifying the Volume” on page 57 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 Table 20 on page 69 and Table 21 on page 70.

**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.

**Tip:** 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.
CVAF Macros

**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.

**Tip:** 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.

**EADSCB=value: Specify the support level for extended attribute DSCBs.**

**EADSCB=OK**
- This specification indicates that the calling program supports extended attribute DSCBs. An extended address volume may have these DSCBs allocated to it. The returned DSCBs (format-3, format-8) may contain extent descriptors described by 28-bit cylinder addresses or DSCBs (format-9) that contain additional attribute information.

For fully qualified data set names in the Filter Criteria List, a CVAFFILT request fails if the EADSCB=OK indicator is not set and the DSCB associated with the fully qualified data set name is a format-8 DSCB.

For partially qualified data set names in the Filter Criteria List, a CVAFFILT request fails if the EADSCB=OK indicator is not set and a DSCB associated with a data set that matches the Filter Criteria List is a format-8 DSCB.

The failing error code for these cases will be reflected as follows:
CVAF Macros

- Data set name status in the FCL (FCLDSNST) is set to a status value of (X'06'). This status code indicates that a data set name is described by a format-8 DSCB and the caller did not specify support for an EAV with the EADSCB=OK keyword.
- Set the no resume CVAF status code (CVSTAT) of STAT072.
- Return code 4.

EADSCB=OK will set the CV4EADOK indicator in the CVPL.

EADSCB=NOTOK
Indicates a calling program does not support extended attribute DSCBs. EADSCB=NOTOK will set the CV4EADOK indicator in the CVPL to off. This is the default.

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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (X'00')</td>
<td>The request was successful.</td>
</tr>
<tr>
<td>4 (X'04')</td>
<td>Logical error; status information in CVSTAT.</td>
</tr>
<tr>
<td>8 (X'08')</td>
<td>Invalid VTOC structure.</td>
</tr>
<tr>
<td>12 (X'0C')</td>
<td>CVAFFILT parameter list in wrong key, or not valid.</td>
</tr>
<tr>
<td>16 (X'10')</td>
<td>I/O error.</td>
</tr>
</tbody>
</table>

CVSTAT in the CVAF parameter list contains the status code. See "Codes Put in the CVSTAT Field" on page 140 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 character qualifier beginning with LOAD.

Example of Using the CVAFFILT Macro

This example uses the CVAFFILT macro to read all format-1 and format-3 DSCBs for specific data sets within a given VTOC as well as for all data sets within a given VTOC. It will also calculate the number of DSCBs and print the totals and appropriate messages to an output file. Refer to the documentation within the sample source in "CVAFFILT Macro Overview and Specification" on page 100 for setup requirements, program logic, and expected output.

The CVAF parameter list, buffer list, and filter criteria list are defined in the sample source. The ICVAFPL macro generates the CVAF parameter list, the ICVAFBFL macro generates the buffer list, and the ICVFCL macro generates the filter criteria list.

Sample JCL for the CVAFFILT macro: The following is the sample JCL used to Assemble, Link, and Execute the example source. Changes will have to be made to this JCL as appropriate for each customer environment.

```
//CVAFFEXP JOB ,MSGCLASS=X,TIME=(,10),
   // NOTIFY=&SYSUID
   //*
   //STEP01 EXEC PROC=A$MACLG
   //SYSIN DD *
   /*    (INCLUDE EXAMPLE SOURCE HERE)
   */
   /*
   //L.SYSLMOD DD DSN=YOUR.AUTH.LINKLIB(CVAFFEXP),DISP=SHR
   //L.SYSIN DD *
   SETCODE AC(1)
   ENTRY CVAFFEXP
   */
   /*
   //G.SYSAEND DD SYSOUT=*  
   //SYSPRINT DD SYSOUT=* 
```
Code example of the CVAFFILT Macro:

```plaintext
CVAFFEXP TITLE 'CVAF CVAFFILT EXAMPLE'
CVAFFEXP CSECT
CVAFFEXP AMODE 24
CVAFFEXP RMODE 24

***********************************************************************
**
** CVAFFILT EXAMPLE TO BE USED IN DFSMS ADVANCED SERVICES MANUAL
**
** CVAFFILT MACRO RUN IN AMODE24/RMODE24
**
** CVAFFILT TEST DSN'S / PROCESSING USED:
**
** 1) 2 SEQUENTIAL DATASETS WITH 5 EXTENTS. (1 FMT1/1 FMT3 EACH)
** DSCBS RETURNED WILL BE COMBINED TO TOTAL 2 FMT1 / 2 FMT3'S
** DSN: CVAFFLT1.DATA01
** DSN: CVAFFLT1.DATA02
** CREATE ON THE VOLUME ASSOCIATED WITH THE CVAFDD DD
**
** 2) PDSE DATASET WITH 122 EXTENTS. (1 FMT1/10 FMT3'S)
** DSN: CVAFFLT1.PDSE01
** CREATE ON THE VOLUME ASSOCIATED WITH THE CVAFDD DD
**
** 3) 1 SEQUENTIAL DATASET WITH 5 EXTENTS (1 FMT1/1 FMT3)
** DSN: CVAFFLT1.DATA01
** CREATE ON THE VOLUME ASSOCIATED WITH THE CVAFDD DD
**
** 4) RETURN DSCB COUNT FOR ENTIRE VOLUME USING CVAFFILT RESUME
** PROCESSING.
** DSN'S ON THE VOLUME: (IN SEQUENCE ORDER)
** SYS1.VTOCIX.V39L62 (VTOC INDEX - 1 FMT1)
** CVAFFLT1.DATA01 (5 EXTENTS - 1 FMT1/1 FMT3)
** CVAFFLT1.DATA02 (5 EXTENTS - 1 FMT1/1 FMT3)
** CVAFFLT1.PDSE01 (122 EXTENTS - 1 FMT1/10 FMT3'S)
** DSN'S CREATED ON THE VOLUME ASSOCIATED WITH THE CVAFDD DD
**
** OUTPUT IN OUTDD DATASET SHOULD BE THE FOLLOWING:
**
**---------------------------------------------------------------------
**
** CVAFFEXP START OF OUTPUT MESSAGES
**
** RC00 VERIFIED - THE REQUEST WAS SUCCESSFUL
** CVAFFILT RETURNED THE FOLLOWING DSCBS FOR DSN: CVAFFLT1.DATA01
** NUMBER OF FORMAT 1 DSCBS - 0000002
** NUMBER OF FORMAT 3 DSCBS - 0000002
**
** RC00 VERIFIED - THE REQUEST WAS SUCCESSFUL
** CVAFFILT RETURNED THE FOLLOWING DSCBS FOR DSN: CVAFFLT1.PDSE01
** NUMBER OF FORMAT 1 DSCBS - 0000001
**```

Chapter 1. Using the Volume Table of Contents 107
CVAF Macros

* NUMBER OF FORMAT 3 DSCBS - 0000010
* RC09 VERIFIED - THE REQUEST WAS SUCCESSFUL
* CVAFFILT RETURNED THE FOLLOWING DSCBS FOR DSN: CVAFFLT1.DATA01
* NUMBER OF FORMAT 1 DSCBS - 0000001
* NUMBER OF FORMAT 3 DSCBS - 0000001
* RC04 VERIFIED - CVSTAT 064 RESUME IS NECESSARY
* CVAFFILT (INITIAL) RETURNED THE FOLLOWING DSCBS FOR THE VOLUME:
* NUMBER OF FORMAT 1 DSCBS - 0000003
* NUMBER OF FORMAT 3 DSCBS - 000002
* CVAFFILT (RESUME) RETURNED THE FOLLOWING DSCBS FOR THE VOLUME:
* NUMBER OF FORMAT 1 DSCBS - 0000001
* NUMBER OF FORMAT 3 DSCBS - 0000010
* CVAFFILT RESUME OPERATION COMPLETE - ALL DSCBS RETURNED
* CVAFFEXP END OF OUTPUT MESSAGES

---------------------------------------------------------------------

* NOTE: THE NUMBER OF DSCBS RETURNED WILL VARY IF THE DATASET DOES
  NOT EXIST ON THE VOLUME OR IF THE DATASET EXTENT IS NOT AS
  LISTED ABOVE OR IF THERE IS NO VTOC INDEX ON THE VOLUME.

* NOTE: WHEN CREATING A PDSE ON A SMS MANAGED VOLUME A VVDS
  WILL ALSO BE CREATED AND THE FMT1 COUNT WILL BE
  INCREASED BY ONE.

***********************************************************************

***********************************************************************

* CVAFFEXP - LOGIC NOTES

* THIS EXAMPLE WILL PERFORM THE FOLLOWING:

* INITIALIZATION
  * - OBTAIN THE NECESSARY INFORMATION FROM THE DASD VOLUME
  * - OPEN AN OUTPUT FILE AND WRITE THE NECESSARY OUTPUT MESSAGES
  * - INITIALIZE THE NECESSARY BUFFER LIST FOR CVAFFILT

* MAINLINE
  * - INITIALIZE A FCL TO READ FOR TWO SPECIFIC SEQ DATASETS
  * - ISSUE CVAFFILT READ TO READ THE DSCBS FOR THE TWO DATASETS
  * - CHECK THE RETURN CODE AND CVSTAT CODE FROM CVAFFILT
  * - COUNT THE NUMBER OF FMT1 AND FMT3 DSCBS FOR THE REQUEST
  * - FORMAT THE DSCB COUNTS AND WRITE TO THE OUTPUT DATASET
  * - ISSUE CVAFFILT RLSE TO RELEASE THE WORK AREAS USED

  * - INITIALIZE A FCL TO READ FOR ONE SPECIFIC PDSE DATASET
  * - ISSUE CVAFFILT READ TO READ THE DSCBS FOR THE PDSE DATASET
  * - CHECK THE RETURN CODE AND CVSTAT CODE FROM CVAFFILT
  * - COUNT THE NUMBER OF FMT1 AND FMT3 DSCBS FOR THE REQUEST
  * - FORMAT THE DSCB COUNTS AND WRITE TO THE OUTPUT DATASET
  * - ISSUE CVAFFILT RLSE TO RELEASE THE WORK AREAS USED

  * - INITIALIZE A FCL TO READ FOR ONE SPECIFIC SEQ DATASET
  * - ISSUE CVAFFILT READ TO READ THE DSCBS FOR THE ONE DATASET
  * - CHECK THE RETURN CODE AND CVSTAT CODE FROM CVAFFILT
  * - COUNT THE NUMBER OF FMT1 AND FMT3 DSCBS FOR THE REQUEST
  * - FORMAT THE DSCB COUNTS AND WRITE TO THE OUTPUT DATASET
  * - ISSUE CVAFFILT RLSE TO RELEASE THE WORK AREAS USED

  * - INITIALIZE A FCL TO READ THE DSCBS ON THE ENTIRE VOLUME
  * - ISSUE CVAFFILT READ TO READ THE DSCBS FOR THE VOLUME
* - CHECK THE RETURN CODE AND CVSTAT CODE FROM CVAFFILT
* - PROCESS ALL DSCBS USING CVAFFILT RESUME AS NECESSARY
* - COUNT THE NUMBER OF FMT1 AND FMT3 DSCBS FOR THE REQUEST
* - FORMAT THE DSCB COUNTS AND WRITE TO THE OUTPUT DATASET
* - ISSUE CVAFFILT RLSE TO RELEASE THE WORK AREAS USED

** FINALIZATION **
* - CLOSE THE OUTPUT FILE
* - EXIT

CVAFEXP - JOB INFORMATION

** NORMAL END OF JOB: **
* - RC=00 AND OUTDD OUTPUT AS DETAILED ABOVE

** ABNORMAL END OF JOB: **
* - ABEND 100 - ERROR OPENING VTOC ON THE DASD VOLUME THAT IS ASSOCIATED WITH THE CVAFDD DD STATEMENT
* - ABEND 101 - ERROR OPENING THE OUTDD DATASET
* - ABEND 102 - ERROR CLOSING THE OUTDD DATASET

** DASD VOLUMES USED IN THIS EXAMPLE: **
* - 339L61 - 3390 WHERE OUTDD DATASET IS DEFINED
* - 339L62 - 3390 WHERE TEST DATASETS DETAILED ABOVE ARE DEFINED

***********************************************************************
* HOUSEKEEPING *
* - SAVE CALLER'S REGISTERS AND ESTABLISH A NEW REGISTER SAVE AREA
***********************************************************************

* STM R14,R12,12(R13) STANDARD LINKAGE CONVENTION
  BALR R11,0 DCL R11 AS IMPLIED BASE REG
  USING BASE,R11,R12 R12 IS ALSO BASE REG
B
BASE L R12,BASEADDR SET UP ADDRESSING FOR R12
BASEADDR DC A(BASE+4096) ADDRESSING FOR R12
CVAFFL00 DS 0H CONTINUE...
ST R13,SAVE+4 SAVE PTR TO CALLER’S SAVE AREA
LA R14,SAVE GET ADDRESS OF THE NEW SAVE AREA
ST R14,8(.R13) CHAIN CALLER’S AREA TO OURS
LR R13,R14 ESTABLISH THE NEW SAVE AREA

***********************************************************************
* INITIALIZATION *
***********************************************************************

* DS 0H INITIALIZATION SECTION
  BAL R14,IDVOLRTN INVOKE RTN TO IDENTIFY THE VOLUME(S)
  BAL R14,OPNRNRTN INVOKE OPEN OUTPUT DATASET RTN
  MVC PDETLINE(133),STRMTMG MOVE START MSG TO LINE
  PUT OUTFILE,PDETLINE WRITE A RECORD TO THE OUTPUT FILE
  MVC PDETLINE(133),BLNKLNE MOVE BLANK LINE TO LINE
  PUT OUTFILE,PDETLINE WRITE A RECORD TO THE OUTPUT FILE
  BAL R14,BUFLRTN INVOKE RTN TO INIT BUFFER LIST (H/E)

***********************************************************************
**CVAF Macros**

* MAINLINE

**MAINLINE SECTION**

MVI DSNNBR,X’02’

SET DSNNBR FLAG TO TWO

BAL R14,FCL1RTN

INVOKE RTN TO INIT FCL (DS01/DS02)

BAL R14,CVAFRD2

INVOKE CVAFFILT READ RTN 2

BAL R14,TSTRCRTN

INVOKE TEST RC RTN

L R15,RETCODE

LOAD R15 WITH RETURN CODE

CH R15,RCODE00

IS RC = 0

BNE MAIN0010

NO - DO NOT FORMAT/PRINT LINES

MVC DS(44),DS01

ELSE MOVE DSN FOR DS01 TO MSG LINE

MVC PDETLINE(133),DSCBMSGA

MOVE MSG TO LINE

PUT OUTFILE,PDETLINE

WRITE A RECORD TO THE OUTPUT FILE

MVC PDETLINE(133),DSCBMSGA

MOVE MSG TO LINE

PUT OUTFILE,PDETLINE

WRITE A RECORD TO THE OUTPUT FILE

BAL R14,FMTOPRTN

INVOKE FORMAT OUTPUT MSG RTN

B MAIN0020

BRANCH TO MAIN0020

**MAIN0010**

DS 0H

CVAFFILT - CVSTAT LOGICAL ERROR

MVC DS3(44),DS01

MOVE DSN FOR DS01 TO MSG LINE

MVC PDETLINE(133),DSCBMSGC

MOVE MSG TO LINE

PUT OUTFILE,PDETLINE

WRITE A RECORD TO THE OUTPUT FILE

MVC PDETLINE(133),DSCBMSGC

MOVE MSG TO LINE

PUT OUTFILE,PDETLINE

WRITE A RECORD TO THE OUTPUT FILE

BAL R14,FMTOPRTN

INVOKE FORMAT OUTPUT MSG RTN

B MAIN0020

BRANCH TO MAIN0020

**MAIN0020**

DS 0H

MVC PDETLINE(133),BLNKLNE

MOVE BLANK LINE TO LINE

PUT OUTFILE,PDETLINE

WRITE A RECORD TO THE OUTPUT FILE

BAL R14,CVAFLR

INVOKE CVAFFILT RELEASE RTN

MVI DSNNBR,X’01’

SET DSNNBR FLAG TO ONE

BAL R14,FCL2RTN

INVOKE RTN TO INIT FCL (DS03)

BAL R14,CVAFRD1

INVOKE CVAFFILT READ RTN 1

BAL R14,TSTRCRTN

INVOKE TEST RC RTN

L R15,RETCODE

LOAD R15 WITH RETURN CODE

CH R15,RCODE00

IS RC = 0

BNE MAIN0030

NO - DO NOT FORMAT/PRINT LINES

MVC DS(44),DS03

ELSE MOVE DSN FOR DS03 TO MSG LINE

MVC PDETLINE(133),DSCBMSGA

MOVE MSG TO LINE

PUT OUTFILE,PDETLINE

WRITE A RECORD TO THE OUTPUT FILE

MVC PDETLINE(133),DSCBMSGA

MOVE MSG TO LINE

PUT OUTFILE,PDETLINE

WRITE A RECORD TO THE OUTPUT FILE

BAL R14,FMTOPRTN

INVOKE FORMAT OUTPUT MSG RTN

B MAIN0040

BRANCH TO MAIN0040

**MAIN0030**

DS 0H

CVAFFILT - CVSTAT LOGICAL ERROR

MVC DS3(44),DS03

MOVE DSN FOR DS03 TO MSG LINE

MVC PDETLINE(133),DSCBMSGC

MOVE MSG TO LINE

PUT OUTFILE,PDETLINE

WRITE A RECORD TO THE OUTPUT FILE

MVC PDETLINE(133),DSCBMSGC

MOVE MSG TO LINE

PUT OUTFILE,PDETLINE

WRITE A RECORD TO THE OUTPUT FILE

**MAIN0040**

DS 0H

MVC PDETLINE(133),BLNKLNE

MOVE BLANK LINE TO LINE

PUT OUTFILE,PDETLINE

WRITE A RECORD TO THE OUTPUT FILE

BAL R14,CVAFLR

INVOKE CVAFFILT RELEASE RTN

BAL R14,FCL3RTN

INVOKE RTN TO INIT FCL (DS01)

BAL R14,CVAFRD1

INVOKE CVAFFILT READ RTN

BAL R14,TSTRCRTN

INVOKE TEST RC RTN

L R15,RETCODE

LOAD R15 WITH RETURN CODE

CH R15,RCODE00

IS RC = 0

BNE MAIN0050

NO - DO NOT FORMAT/PRINT LINES

MVC DS(44),DS01

ELSE MOVE DSN FOR DS01 TO MSG LINE

MVC PDETLINE(133),DSCBMSGA

MOVE MSG TO LINE

PUT OUTFILE,PDETLINE

WRITE A RECORD TO THE OUTPUT FILE

BAL R14,FMTOPRTN

INVOKE FORMAT OUTPUT MSG RTN

B MAIN0060

BRANCH TO MAIN0060

**MAIN0050**

DS 0H

CVAFFILT - CVSTAT LOGICAL ERROR

MVC DS3(44),DS01

MOVE DSN FOR DS01 TO MSG LINE

MVC PDETLINE(133),DSCBMSGC

MOVE MSG TO LINE

PUT OUTFILE,PDETLINE

WRITE A RECORD TO THE OUTPUT FILE

**MAIN0060**

DS 0H
Finalization Section

MVC PDETLINE(133),BLNKLNE MOVE BLANK LINE TO LINE
PUT OUTFILE,PDETLINE WRITE A RECORD TO THE OUTPUT FILE
BAL R14,CVAFRL INVOKE CVAFFILT RELEASE RTN
* BAL R14,FCL4RTN INVOKE RTN TO INIT FCL (ENTIRE VOL)
BAL R14,CVAFRDG INVOKE CVAFFILT READ RTN (FOR RESUME)
LA R9,CVPLDEFA ESTABLISH ADDRESSABILITY
* USING CVPLMAP,R9 TO THE CVPL (FOR CVSTAT)
BAL R14,TSTRCRTN INVOKE TEST RC RTN

***********************************************************************
** FINALIZATION **
***********************************************************************

FINAL DS 0H FINALIZATION SECTION
MVC PDETLINE(133),BLNKLNE MOVE BLANK LINE TO LINE
PUT OUTFILE,PDETLINE WRITE A RECORD TO THE OUTPUT FILE
MVC PDETLINE(133),ENDMSG MOVE END MSG TO LINE
PUT OUTFILE,PDETLINE WRITE A RECORD TO THE OUTPUT FILE
BAL R14,CLOSERTN INVOKE CLOSE OUTPUT DATASET RTN
L R13,4(R13) RESTORE REGISTER
LM R14,R12,12(R13) RESTORE CALLERS REGISTERS
LA R15,0 SET RC TO 0
BR R14 RETURN TO CALLER

***********************************************************************
* OPENRTN *
* - ROUTINE TO OPEN OUTPUT FILE USED BY THIS MODULE *
***********************************************************************

OPENRTN DS 0H OPEN FILES ROUTINE
ST R14,OPENSBE STORE C(R14) INTO SAVE AREA
OPEN (OUTFILE,(OUTPUT)) OPEN THE OUTDD OUTPUT FILE FOR MSGS
TM OUTFILE+(DCBOFLGS-IHADCB),DCBOFOPN IS FILE OPEN?
BO OPENEXIT FILE OPEN OK - EXIT OPEN RTN
LA R1,EABN101 OUTPUT FILE NOT OPEN-USER ABEND 101
BAL R14,ABENDRTN INVOKE ABEND ROUTINE
OPENEXIT DS 0H EXIT FROM OPEN ROUTINE
L R14,OPENSBE LOAD C(OPENSBE) INTO R14
BR R14 EXIT

***********************************************************************
* CLOSERTN *
* - ROUTINE TO CLOSE OUTPUT FILE USED BY THIS MODULE *
***********************************************************************

CLOSERTN DS 0H CLOSE FILES ROUTINE
ST R14,CLOSSAVE STORE C(R14) INTO SAVE AREA
CLOSE (OUTFILE) CLOSE OUTPUT FILE
LTR R15,R15 CHECK IF CLOSED OK
BZ CLOSEEXIT IF OK BRANCH TO CLOSEEXIT
LA R1,EABN102 ELSE SETUP FOR USER ABEND 102
BAL R14,ABENDRTN INVOKE ABEND ROUTINE
CLOSEEXIT DS 0H EXIT FROM CLOSE ROUTINE
L R14,CLOSSAVE LOAD C(CLOSSAVE) INTO R14
BR R14 EXIT

***********************************************************************
* ABENDRTN *
* - FORCE AN ABEND ROUTINE *
***********************************************************************

ABENDRTN DS 0H ABEND ROUTINE
ST R14,ABENSBE STORE C(R14) INTO SAVE AREA
ABEND (R1),DUMP ISSUE USER ABEND WITH DUMP
ABENEXIT DS 0H EXIT FROM ABEND ROUTINE
CVAF Macros

L R14,ABENSAVE LOAD C(ABENSAVE) INTO R14
BR R14 EXIT
*
******************************************************************************
* IDVOLRTN *
* - OBTAIN THE NECESSARY INFORMATION FROM THE DASD VOLUME *
******************************************************************************
*
IDVOLRTN DS 0H IDENTIFY VOLUME ROUTINE
ST R14,IDVLSAVE STORE C(R14) INTO SAVE AREA
RDJFCB (VTODCB,(INPUT)) READ JFCB / OPEN VTOC
MVI JFCB1,X'04' PUT IN ID FOR FORMAT 4
MVC JFCB1+1(43),JFCB1 SETUP FOR VTOC OPEN
OPEN (VTODCB,(INPUT)),TYPE=J OPEN VTOC (OPEN TYPE=J)
TM VTODCB+(DCBOFLGS-IHADCB),DCBOFOPN
BO IDVOL010 BRANCH TO IDVOL010 - GOOD OPEN
LA R1,EABN100 ELSE SETUP FOR USER ABEND 100
BAL R14,ABENDRTN INVOKE ABEND ROUTINE
IDVOL010 DS 0H GOOD OPEN - OBTAIN VOLUME INFORMATION
SLR RDEB,RDEB INIT REG1 FOR DEB PTR
SLR RUCB,RUCB INIT REG2 FOR UCB PTR
ICM RDEB,B'0111',VTODCB+(DCBDEBA-IHADCB) GET DEB ADDRESS
ST RDEB,DEBADD SAVE DEB ADDRESS INTO R1
ICM RUCB,B'0111',(DEBBASND-DEBBASIC)+(DEBUCBA-DEBDASD)(RDEB)
ST RUCB,UCBADD SAVE UCB ADDRESS INTO R2
IDVLEXIT DS 0H EXIT FROM IDVOLRTN
L R14,IDVLSAVE LOAD C(IDVLSAVE) INTO R14
BR R14 EXIT
*
******************************************************************************
* TSTRCRTN *
* - TEST RETURN CODE FROM CVAFFILT *
* - FORMAT AND PRINT MESSAGES AS NEEDED *
******************************************************************************
*
TSTRCRTN DS 0H CHECK RETURN CODE ROUTINE
ST R14,TSTRSAVE STORE C(R14) INTO SAVE AREA
L R15,RETCODE LOAD R15 WITH RC SAVED FROM LAST CALL
CH R15,RCODE00 IF RETURN CODE = 00
BE TSTRC00 BRANCH TO PROCESS RC00
CH R15,RCODE04 IF RETURN CODE = 04
BE TSTRC04 BRANCH TO PROCESS RC04
TSTRCER DS 0H ELSE PRINT GENERAL ERROR MESSAGE
MVC PDETLINE(133),RCERMSG MOVE ERROR MSG TO PRINT LINE
PUT OUTFILE,PDETLINE WRITE A RECORD TO THE OUTPUT FILE
B TSTREXIT BRANCH TO EXIT RTN
TSTRC00 DS 0H PROCESS RETURN CODE = 00
MVC PDETLINE(133),RC00MSG MOVE RC00 MSG TO PRINT LINE
PUT OUTFILE,PDETLINE WRITE A RECORD TO THE OUTPUT FILE
BAL R14,FMTCRTCRTN BRANCH TO FMT COUNT RTN
BAL R14,FMTOPRTN INVOKE FORMAT OUTPUT MSG ROUTINE
B TSTREXIT BRANCH TO EXIT RTN
TSTRC04 DS 0H PROCESS RETURN CODE = 04
CLUD CVSTAT,STAT064 DO WE NEED RESUME?
BNE TSTR0010 NO - PRINT LOGICAL ERROR MSG
BAL R14,CVAFRS INVOKE CVAFRS RESUME ROUTINE
B TSTREXIT BRANCH TO EXIT RTN
TSTR0010 DS 0H PRINT RC04-OTHER LOGICAL ERROR/CVSTAT
MVC PDETLINE(133),RC04MSG MOVE RC04 MSG TO PRINT LINE
PUT OUTFILE,PDETLINE WRITE A RECORD TO THE OUTPUT FILE
TSTREXIT DS 0H EXIT FROM TSTRCRTN
L R14,TSTRSAVE LOAD C(TSTRSAVE) INTO R14
BR R14 EXIT
*
******************************************************************************
* FMTCTRTN *
* - COUNT THE NUMBER OF FMT1 AND FMT3 DSCBS FOR THE REQUEST *
******************************************************************************

112  z/OS V2R1.0 DFSMSdfp Advanced Services
**FMTCTRTN DS 0H COUNT DSCBS RETURNED ROUTINE**

ST R14,FMTCSAVE STORE C(R14) INTO SAVE AREA
SLR R6,R6 ZERO OUT R6
SLR R7,R7 ZERO OUT R7
CLI DSNNBR,'X'01' IF DSNNBR FLAG = 1
BE FMTC0010 BRANCH TO LOAD ADDR OF FCL
LA R4,FCLDEF ELSE LOAD R4 WITH ADDR OF FCL2
B FMTC0020 BRANCH TO SET USING

**FMTC0010 DS 0H**
LA R4,FCLDEF LOAD R4 WITH ADDR OF FCL

**FMTC0020 DS 0H CONTINUE - SET USING**
USING FCLMAP,R4 ESTABLISH ADDRESSABILITY TO FCL
SLR R5,R5 ZERO OUT R5
ICM R5,B'0011',FCLDSCBR DETERMINE IF ANY DSCBS RETURNED
BZ FMTC0060 NO - GO AND PRINT APPROPRIATE MSG
LA R4,DSCBDEF LOAD R4 WITH ADDR OF DSCB MAP
USING DSCBMAP,R4 ESTABLISH ADDRESSABILITY TO DSCB

**FMTC0030 DS 0H COUNT DSCBS BY TYPE RETURNED**
CLI DS1FMTID,X'F1' IF FORMAT1
BE FMTC0040 BRANCH TO FMTC0040
CLI DS1FMTID,X'F3' ELSE IF NOT FORMAT3
BNE FMTC0050 BRANCH TO FMTC0050
LA R7,1(R7) ADD 1 TO FORMAT3 COUNT
B FMTC0050 BRANCH TO FMTC0050

**FMTC0040 DS 0H FMTC0040 - FORMAT1 INCREMENT**
LA R6,1(R6) ADD 1 TO FORMAT1 COUNT

**FMTC0050 DS 0H FMTC0050 - PROCESS THROUGH DSCBS**
LA R4,DSCBSIZ(R4) ADD DSCBSIZ TO R4
BCT R5,FMTC0030 SUBTRACT 1 FROM DSCB COUNT AND CONT
ST R6,RETF1 STORE #FMT1'S INTO RETF1
ST R7,RETF3 STORE #FMT3'S INTO RETF3
DROP R4 DROP R4 USING
B FMTCEXIT BRANCH AROUND FMTC0060

**FMTC0060 DS 0H PRINT MSG - NO DSCB'S RETURNED**
MVC PDETLINE(133),NODSCBM MOVE MSG TO PRINT LINE
PUT OUTFILE,PDETLINE WRITE A RECORD TO THE OUTPUT FILE
SLR R6,R6 ZERO OUT R6
ST R6,RETF1 STORE ZERO INTO RETF1
SLR R7,R7 ZERO OUT R7
ST R7,RETF3 STORE ZERO INTO RETF3

**FMTCEXIT DS 0H EXIT FROM FMTCTRTN**
L R14,FMTCSAVE LOAD C(FMTCSAVE) INTO R14
BR R14 EXIT

**FMTOPRTN DS 0H**

**FMTOPRTN**

- FORMAT THE DSCB COUNTS AND WRITE TO THE OUTPUT DATASET

**FMTOPRTN DS 0H FORMAT OUTPUT ROUTINE**

ST R14,FMTOSAVE STORE C(R14) INTO SAVE AREA
MVC MSG(29),DSCBMSG1 MOVE MSG TO FORMAT LINE
L R6,RETF1 LOAD R6 WITH NBR OF FMT1'S RETURNED
CVF R6,WF1 CONVERT TO DEC FOR PRINTING
UNPK WFMTREC+29(7),WF1 UNPACK TO FORMAT LINE
OI WFMTREC+35,X'F0' SET APPROPRIATE BITS
MVC PDETLINE(133),WFMTREC MOVE RECORD TO OUTPUT LINE
PUT OUTFILE,PDETLINE WRITE A RECORD TO THE OUTPUT FILE
MVC PDETLINE(133),WFMTREC MOVE RECORD TO OUTPUT LINE
PUT OUTFILE,PDETLINE WRITE A RECORD TO THE OUTPUT FILE

Chapter 1. Using the Volume Table of Contents 113
**CVAF Macros**

```
FMTOEXIT DS 0H EXIT FROM FMTPRTN
   L   R14,FMTOSAVE LOAD C(FMTOSAVE) INTO R14
   BR  R14 EXIT

***********************************************************************
* BUFLRTN *
* - INITIALIZE BUFFER LIST HEADER (BFLH) *
* - INITIALIZE BUFFER LIST ELEMENTS (BFLE) *
***********************************************************************

BUFLRTN DS 0H BUFFER LIST INITIALIZATION ROUTINE
   ST  R14,BUFLSAVE STORE C(R14) INTO SAVE AREA
   XC  BFLHDEF(BFLSIZE),BFLHDEF CLEAR BUFRT LIST AREA
   LA  R1,BFLHDEF R1 - BUFFER LIST HEADER
   USING BFLMAP,R1 ESTABLISH ADDRESSABILITY
   MI  BFLHNOE,BUFNBR SET NUMBER OF BUFFER ELEMENTS
   OI  BFLHFL,BFLHDSCB IDENTIFY AS DSCB BUFR ELEMENT LIST
   LA  R2,BFLHDEF+BFLHN R2 - FIRST BUFFER LIST ELEMENT
   USING BFLE,R2 ESTABLISH ADDRESSABILITY
   LA  R3,DSCBDEF R3 - FIRST DSCB BUFFER
   LA  R4,BUFNBR R4 = NUMBER OF ELEMENTS AND BUFRS
   BFL0010 OI  BFLEFL,BFLECHR REQUEST CCHHR ON RETURN
   MVI  BFLHNOE,BUFNBR SET NUMBER OF BUFFER ELEMENTS
   OI  BFLHFL,BFLHDSCB IDENTIFY AS DSCB BUFR ELEMENT LIST
   LA  R2,BFLHDEF+BFLHN R2 - FIRST BUFFER LIST ELEMENT
   USING BFLE,R2 ESTABLISH ADDRESSABILITY
   LA  R3,DSCBDEF R3 - FIRST DSCB BUFFER
   LA  R4,BUFNBR R4 = NUMBER OF ELEMENTS AND BUFRS
   BCT  R4,BFL0010 LOOP THROUGH ALL ELEMENTS
   DRO  R1,R2 DROP TEMP USING
   BUFLEXIT DS 0H EXIT FROM BUFLRTN
   L  R14,BUFLSAVE LOAD C(BUFLSAVE) INTO R14
   BR  R14 EXIT

***********************************************************************
* FCL1RTN *
* - INITIALIZE FILTER CRITERIA LIST (FCL) HEADER AND ELEMENT *
* - INITIALIZE A FCL TO READ FOR TWO SPECIFIC SEQ DATASETS *
***********************************************************************

FCL1RTN DS 0H FCL INITIALIZATION ROUTINE
   ST  R14,FCL1SAVE STORE C(R14) INTO SAVE AREA
   XC  FCLDEF2(FCLSIZE2),FCLDEF2 CLEAR FCL AREA
   LA  R1,FCLDEF2 R1 - FCL HEADER
   USING FCLMAP,R1 ESTABLISH ADDRESSABILITY
   LA  R2,FCLHDEND R2 - FIRST FCL ELEMENT
   USING FCLDSN,R2 ESTABLISH ADDRESSABILITY
   MVI  FCLID,CFCID SET THE EYECATCHER 'FCL'
   MVI  FCLCOUNT,=H'80' SET LENGTH FOR DSN1 (15)
   LA  R3,DS01 DSN=CVAFFLT1.DATA01
   ST  R3,FCLDSNA SET ADDR OF DSN
   LA  R2,FCLSNEL(R2) LOAD R2 WITH ADDR OF 2ND FCL ELEMENT
   MI  FCLSNLG,X'0F' SET LENGTH FOR DSN2 (15)
   LA  R3,DS02 DSN=CVAFFLT1.DATA02
   DRO  R1,R2 DROP TEMP USING
   FC1EXIT DS 0H EXIT FROM FCLRTN
   L  R14,FCLISAVE LOAD C(FCLISAVE) INTO R14
   BR  R14 EXIT

***********************************************************************
* FCL2RTN *
* - INITIALIZE FILTER CRITERIA LIST (FCL) HEADER AND ELEMENT *
***********************************************************************

FCL2RTN DS 0H FCL INITIALIZATION ROUTINE
   ST  R14,FCLISAVE STORE C(R14) INTO SAVE AREA
   XC  FCLDEF2(FCLSIZE2),FCLDEF2 CLEAR FCL AREA
   LA  R1,FCLDEF2 R1 - FCL HEADER
   USING FCLMAP,R1 ESTABLISH ADDRESSABILITY
   LA  R2,FCLHDEND R2 - FIRST FCL ELEMENT
   USING FCLDSN,R2 ESTABLISH ADDRESSABILITY
   MVI  FCLID,CFCID SET THE EYECATCHER 'FCL'
   MVI  FCLCOUNT,=H'80' SET NUMBER OF FCL ELEMENTS
   MI  FCLISLSN,X'80' SET FLAG FOR FULLY QUAL DSN
   MVI  FCLSNLG,X'0F' SET LENGTH FOR DSN1 (15)
   LA  R3,DS01 DSN=CVAFFLT1.DATA01
   ST  R3,FCLDSNA SET ADDR OF DSN
   LA  R2,FCLSNEL(R2) LOAD R2 WITH ADDR OF 2ND FCL ELEMENT
   MI  FCLSNLG,X'0F' SET LENGTH FOR DSN2 (15)
   LA  R3,DS02 DSN=CVAFFLT1.DATA02
   DRO  R1,R2 DROP TEMP USING
   FC1EXIT DS 0H EXIT FROM FCLRTN
   L  R14,FCLISAVE LOAD C(FCLISAVE) INTO R14
   BR  R14 EXIT

```
**CVAF Macros**

```
* - INITIALIZE A FCL TO READ FOR ONE SPECIFIC PDSE DATASET *
***********************************************************************
FCL2RTN DS 0H FCL INITIALIZATION ROUTINE
ST R14,FCL2SAVE STORE C(R14) INTO SAVE AREA
XC FCLDEF(FCLSIZE),FCLDEF CLEAR FCL AREA
LA R1,FCLDEF R1 - FCL HEADER
USING FCLMAP,R1 ESTABLISH ADDRESSABILITY
LA R2,FCLHDEND R2 - FIRST (ONLY) FCL ELEMENT
USING FCLDSN,R2 ESTABLISH ADDRESSABILITY
MVC FCLID,CFCLID SET THE EYECATCHER 'FCL'
MVC FCLCOUNT,=H'1' SET NUMBER OF FCL ELEMENTS
MVI FCL1FLAG,X'80' SET FLAG FOR FULLY QUAL DSN
MVI FCLDSNLG,X'0F' SET LENGTH FOR DSN (15)
LA R3,DS03 DSN=CVAFFLT1.PDSE01
* PDSE DS - 122 EXTENTS
* 1 FORMAT1 AND 10 FORMAT3'S
ST R3,FCLDSNA SET ADDR OF DSN
DROP R1,R2 DROP TEMP USING
FC2EXIT DS 0H EXIT FROM FCLRTN
L R14,FCL2SAVE LOAD C(FCL2SAVE) INTO R14
BR R14 EXIT

***********************************************************************
* FCL3RTN *
* - INITIALIZE FILTER CRITERIA LIST (FCL) HEADER AND ELEMENT *
* - INITIALIZE A FCL TO READ FOR ONE SPECIFIC SEQ DATASET *
***********************************************************************
FCL3RTN DS 0H FCL INITIALIZATION ROUTINE
ST R14,FCL3SAVE STORE C(R14) INTO SAVE AREA
XC FCLDEF(FCLSIZE),FCLDEF CLEAR FCL AREA
LA R1,FCLDEF R1 - FCL HEADER
USING FCLMAP,R1 ESTABLISH ADDRESSABILITY
LA R2,FCLHDEND R2 - FIRST (ONLY) FCL ELEMENT
USING FCLDSN,R2 ESTABLISH ADDRESSABILITY
MVC FCLID,CFCLID SET THE EYECATCHER 'FCL'
MVC FCLCOUNT,=H'1' SET NUMBER OF FCL ELEMENTS
MVI FCL1FLAG,X'80' SET FLAG FOR FULLY QUAL DSN
MVI FCLDSNLG,X'0F' SET LENGTH FOR DSN (15)
LA R3,DS01 DSN=CVAFFLT1.DATA01
* SEQ DS - 5 EXTENTS
* 1 FORMAT1 AND 1 FORMAT3
ST R3,FCLDSNA SET ADDR OF DSN
DROP R1,R2 DROP TEMP USING
FC3EXIT DS 0H EXIT FROM FCLRTN
L R14,FCL3SAVE LOAD C(FCL3SAVE) INTO R14
BR R14 EXIT

***********************************************************************
* FCL4RTN *
* - INITIALIZE FILTER CRITERIA LIST (FCL) HEADER AND ELEMENT *
* - INITIALIZE A FCL TO READ THE DSCBS ON THE ENTIRE VOLUME *
***********************************************************************
FCL4RTN DS 0H FCL INITIALIZATION ROUTINE
ST R14,FCL4SAVE STORE C(R14) INTO SAVE AREA
XC FCLDEF(FCLSIZE),FCLDEF CLEAR FCL AREA
LA R1,FCLDEF R1 - FCL HEADER
USING FCLMAP,R1 ESTABLISH ADDRESSABILITY
LA R2,FCLHDEND R2 - FIRST (ONLY) FCL ELEMENT
USING FCLDSN,R2 ESTABLISH ADDRESSABILITY
MVC FCLID,CFCLID SET THE EYECATCHER 'FCL'
MVC FCLCOUNT,=H'1' SET NUMBER OF FCL ELEMENTS
MVI FCL1FLAG,X'80' SET FLAG FOR FULLY QUAL DSN
MVI FCLDSNLG,X'0F' SET LENGTH FOR DSN (15)
LA R3,DS01 DSN=CVAFFLT1.DATA01
* SEQ DS - 5 EXTENTS
* 1 FORMAT1 AND 1 FORMAT3
ST R3,FCLDSNA SET ADDR OF DSN
DROP R1,R2 DROP TEMP USING
FC4EXIT DS 0H EXIT FROM FCLRTN
L R14,FCL4SAVE LOAD C(FCL4SAVE) INTO R14
BR R14 EXIT
```

Chapter I. Using the Volume Table of Contents 115
CVAF Macros

ST R3,FCLDSNA SET ADDR OF DSN
DROP R1,R2 DROP TEMP USING
FC4EXIT DS 0H EXIT FROM FCLRTN
L R14,FCL4SAVE LOAD C(FCL4SAVE) INTO R14
BR R14 EXIT

* ***********************************************************************
* CVAFRD1 *
* - INVOKE THE CFAFFILT MACRO AND READ THE DSCBS (1 DSN) *
* ***********************************************************************

CVAFRD1 DS 0H CVAFFILT - READ ROUTINE (2 DSN)
ST R14,CVR1SAVE STORE C(R14) INTO SAVE AREA
SLR R2,R2 ZERO OUT R2
LA R2,CVPLDEF LOAD R2 WITH ADDR OF CVPL
L R3,UCBADD LOAD R3 WITH UCB ADDRESS
CVAFFILT ACCESS=READ,UCB=(R3),FCL=FCLDEF,BUFLIST=BFLHDEF, X
MF=(E,(R2))
ST R15,RETCODE STORE RC FOR LATER INTERROGATION
CVRIEXIT DS 0H EXIT FROM CVAFRD1
L R14,CVR1SAVE LOAD C(CVR1SAVE) INTO R14
BR R14 EXIT

* ***********************************************************************
* CVAFRD2 *
* - INVOKE THE CFAFFILT MACRO AND READ THE DSCBS (2 DSN'S) *
* ***********************************************************************

CVAFRD2 DS 0H CVAFFILT - READ ROUTINE (2 DSN'S)
ST R14,CVR2SAVE STORE C(R14) INTO SAVE AREA
SLR R2,R2 ZERO OUT R2
LA R2,CVPLDEF LOAD R2 WITH ADDR OF CVPL
L R3,UCBADD LOAD R3 WITH UCB ADDRESS
LA R4,FCLDEF2 LOAD R4 WITH ADDR OF FCLDEF2
LA R5,BFLHDEF LOAD R5 WITH ADDR OF FCLDEF2
CVAFFILT ACCESS=READ,UCB=(R3),FCL=(R4),BUFLIST=(R5), X
MF=(E,(R2))
ST R15,RETCODE STORE RC FOR LATER INTERROGATION
CVR2EXIT DS 0H EXIT FROM CVAFRD2
L R14,CVR2SAVE LOAD C(CVR2SAVE) INTO R14
BR R14 EXIT

* ***********************************************************************
* CVAFRDA *
* - INVOKE THE CFAFFILT MACRO AND READ ALL THE DSCBS *
* ***********************************************************************

CVAFRDA DS 0H CVAFFILT - READ ALL DSCBS ROUTINE
ST R14,CVRASAVE STORE C(R14) INTO SAVE AREA
CVAFFILT ACCESS=READ,UCB=UCBADD,FCL=FCLDEF,BUFLIST=BFLHDEF, X
FLTAREA=KEEP,IOAREA=KEEP, X
MF=(E,CVPLDEFA)
ST R15,RETCODE STORE RC FOR LATER INTERROGATION
CVRAEXIT DS 0H EXIT FROM CVAFRDA
L R14,CVRASAVE LOAD C(CVRASAVE) INTO R14
BR R14 EXIT

* ***********************************************************************
* CVAFRS *
* - INVOKE THE CFAFFILT MACRO USING RESUME *
* ***********************************************************************

CVAFRS DS 0H CVAFFILT - RESUME ROUTINE
ST R14,CVRSSAVE STORE C(R14) INTO SAVE AREA
CVRS0000 DS 0H NOW CHECK RC AND STAT CODES
CH R15,RCODE00 IS THE RC FROM RESUME 07
BE CVRS0050 YES BRANCH TO CVRS0050

116 z/OS V2R1.0 DFSMSdfp Advanced Services
**CVAF Macros**

CH R15,RCODE04 IS THE RC FROM RESUME 4?
BE CVRS0020 YES BRANCH TO CVRS0020

CVRS0010 DS 0H ELSE PRINT MSG (RC IS 8,12,OR 16)
MVC PDETLINE(133),RCERMSG MOVE MSG TO LINE
PUT OUTFILE,PDETLINE WRITE A RECORD TO THE OUTPUT FILE
B CVRSEXIT BRANCH TO EXIT ROUTINE

CVRS0020 DS 0H PROCESS FOR RC04 - CHECK FOR STAT064
CLI CVSTAT,STAT064 IS THE CVSTAT CODE 064(RESUME NEEDED)
BNE CVRS0060 NO - BRANCH TO CVRS0060
CLI RESFLG,X'01' IS RESUME FLAG ON
BE CVRS0030 YES - BRANCH TO CVRS0030
MVC PDETLINE(133),ST64MSG1 MOVE MSG TO LINE
PUT OUTFILE,PDETLINE WRITE A RECORD TO THE OUTPUT FILE
MVC PDETLINE(133),ST64MSG2 MOVE MSG TO LINE
PUT OUTFILE,PDETLINE WRITE A RECORD TO THE OUTPUT FILE
BAL R14,FMTCTRTN INVOKE FORMAT COUNT ROUTINE
BAL R14,FMTOPRTN INVOKE FORMAT OUTPUT MSG ROUTINE
MVI RESFLG,X'01' RESUME NEEDED - SET FLAG ON
B CVRS0040 BRANCH TO CVRS0040

CVRS0030 DS 0H RESUME PROCESSING
MVC PDETLINE(133),ST64MSG3 MOVE MSG TO LINE
PUT OUTFILE,PDETLINE WRITE A RECORD TO THE OUTPUT FILE
BAL R14,FMTCTRTN INVOKE FORMAT COUNT ROUTINE
BAL R14,FMTOPRTN INVOKE FORMAT OUTPUT MSG ROUTINE
MVC PDETLINE(133),ST64MSG4 MOVE MSG TO LINE
PUT OUTFILE,PDETLINE WRITE A RECORD TO THE OUTPUT FILE

* RELEASE WORK AREAS

* CVAFFILT ACCESS=RESUME,MF=(E,CVPLDEFA)
B CVRS0000 BRANCH TO CHECK RETURN CODE AGAIN

CVRS0050 DS 0H RC IS 0 NO LONGER NEED RESUME
MVC PDETLINE(133),ST64MSG3 MOVE MSG TO LINE
PUT OUTFILE,PDETLINE WRITE A RECORD TO THE OUTPUT FILE
BAL R14,FMTCTRTN INVOKE FORMAT COUNT ROUTINE
BAL R14,FMTOPRTN INVOKE FORMAT OUTPUT MSG ROUTINE
MVC PDETLINE(133),ST64MSG4 MOVE MSG TO LINE
PUT OUTFILE,PDETLINE WRITE A RECORD TO THE OUTPUT FILE

* WORKING STORAGE *

***********************************************************************
** CVAFRL **
** - INVOKE THE CVAFFILT MACRO AND RELEASE WORK AREAS **
***********************************************************************

** CVAFRL **
DS 0H CVAFFILT - RLSE ROUTINE
ST R14,CVRLSAVE STORE C(R14) INTO SAVE AREA
CVAFFILT ACCESS=RLSE,FCL=0,BUFLIST=0,FLTAREA=NOKEEP, X
MF=(E,CVPLDEFA)
B CVRSEXIT BRANCH TO EXIT ROUTINE
CVRS0060 DS 0H RC IS 4 BUT CVSTAT IS NOT 064
MVC PDETLINE(133),RC04MSG MOVE MSG TO LINE
PUT OUTFILE,PDETLINE WRITE A RECORD TO THE OUTPUT FILE
CVRSEXIT DS 0H EXIT FROM CVARLS
L R14,CVRLSAVE LOAD C(CVRLSAVE) INTO R14
BR R14 EXIT

***********************************************************************
** WORKING STORAGE **
***********************************************************************
CVAF Macros

* DS 0D
  DC CL36'CVAFFEXP-WORKING STORAGE BEGINS HERE'
* ***********************************************
* EQUATES *
* ***********************************************
* EABN100 EQU 100 USER ABEND CODE 100 - VTOC OPEN ERROR
EABN101 EQU 101 USER ABEND CODE 101 - OUTDD OPEN ERROR
EABN102 EQU 102 USER ABEND CODE 102 - OUTDD CLOSE ERROR
BUFNBR EQU 11 11 BUFFERS TO BE USED
RO EQU 0
R1 EQU 1
RDEB EQU 1 REG1 FOR DEB ADDRESS
R2 EQU 2
RUCB EQU 2 REG2 FOR UCB ADDRESS
R3 EQU 3
R4 EQU 4
R5 EQU 5
R6 EQU 6
R7 EQU 7
R8 EQU 8
R9 EQU 9
R11 EQU 11
R12 EQU 12
R13 EQU 13
R14 EQU 14
R15 EQU 15
* ***********************************************
* SAVE AREAS *
* ***********************************************
* SAVE DC 18'0' MAIN PROGRAM SAVE AREA
OPENSþE DC F'0' OPEN FILES ROUTINE SAVE AREA
CLOSESAVE DC F'0' CLOSE FILES ROUTINE SAVE AREA
ABENSþE DC F'0' ABEND ROUTINE SAVE AREA
IDVLSAVE DC F'0' IDENTIFY VOLUME ROUTINE SAVE AREA
BUFFSAVE DC F'0' INITIALIZE BUFFER ROUTINE SAVE AREA
FCL1SAVE DC F'0' INITIALIZE FCL1 ROUTINE SAVE AREA
FCL2SAVE DC F'0' INITIALIZE FCL2 ROUTINE SAVE AREA
FCL3SAVE DC F'0' INITIALIZE FCL3 ROUTINE SAVE AREA
FCL4SAVE DC F'0' INITIALIZE FCL4 ROUTINE SAVE AREA
TSTRSAVE DC F'0' TEST RETURN CODE ROUTINE SAVE AREA
CVRISAVE DC F'0' CVAFFILT READ 1 DSN ROUTINE SAVE AREA
CVR2SAVE DC F'0' CVAFFILT READ 2 DSN ROUTINE SAVE AREA
CVRASAVE DC F'0' CVAFFILT READ ALL ROUTINE SAVE AREA
CVRLSAVE DC F'0' CVAFFILT RLSE ROUTINE SAVE AREA
CVRSSAVE DC F'0' CVAFFILT RESUME ROUTINE SAVE AREA
FMTCSAVE DC F'0' FORMAT DSCB COUNT ROUTINE SAVE AREA
FMTOSAVE DC F'0' FORMAT OUTPUT ROUTINE SAVE AREA
* ***********************************************
* CONSTANTS *
* ***********************************************
* RETCODE DC F'999'
CODE0 DC H'0' RETURN CODE 0 - HALFWORD
CODE04 DC H'4' RETURN CODE 4 - HALFWORD
DSNBR DC X'FF' INDICATE NBR OF DSN TO PROCESS
SFLDC DC X'00' RESUME FLAG - OFF
BLNMDC DC CL133' '
SRTMSDC DC CL133' CVAFFEXP START OF OUTPUT MESSAGES '
ENDMSDC DC CL133' CVAFFEXP END OF OUTPUT MESSAGES '
ST64MSG1 DC CL133' RC04 VERIFIED - CVSTAT 064 RESUME IS NECESSARY '
ST64MSG2 DS OC133
CVAF Macros

DC CL49' CVAFFILT (INITIAL) RETURNED THE FOLLOWING DSCBS'
DC CL84' FOR THE VOLUME:
ST64MSG3 DS OCL133
DC CL49' CVAFFILT (RESUME) RETURNED THE FOLLOWING DSCBS'
DC CL84' FOR THE VOLUME:
ST64MSG4 DS OCL133
DC CL48' CVAFFILT RESUME OPERATION COMPLETE - ALL DSCBS'
DC CL85' RETURNED'
DSCBMSGA DS OCL133
DC CL48' CVAFFILT RETURNED THE FOLLOWING DSCBS FOR DSN: '
DS DC CL44' '
DC CL41' '
DSCBMSGB DS OCL133
DC CL48' AND FOR DSN: '
DS2 DC CL44' '
DC CL41' '
DSCBMSGC DS OCL133
DC CL48' CVAFFILT LOGICAL ERROR STATUS RETURNED - DSN: '
DS3 DC CL44' '
DC CL41' '
NO DSCBM DC CL133' NO DSCBS RETURNED FROM CVAFFILT'
RC00MSG DC CL133' RC00 VERIFIED - THE REQUEST WAS SUCCESSFUL'
RC04MSG DC CL133' RC04 VERIFIED - LOGICAL ERROR STATUS IN CVSTAT'
RCEMMSG DC CL133' RC08, RC12, OR RC16 RETURNED FROM CVAFFILT'
RLSEMSG DC CL133' NON ZERO RETURN CODE BACK FROM RLSE'
DSCBMSG1 DC CL29' NUMBER OF FORMAT 1 DSCBS - '
DSCBMSG2 DC CL29' NUMBER OF FORMAT 3 DSCBS - '
DS01 DC CL44' CVAFFLT1.DATA01'
DS02 DC CL44' CVAFFLT1.DATA02'
DS03 DC CL44' CVAFFLT1.POSE01'
CFCLID DC CL4' FCL'
***********************************************************************
* WORK AREAS *
***********************************************************************
*
WF1 DS D DOUBLE WORD - FORMAT 1 WORK AREA
WF3 DS D DOUBLE WORD - FORMAT 3 WORK AREA
DEBADD DC F'0' DEB ADDRESS SAVE AREA
UCBADD DC F'0' UCB ADDRESS SAVE AREA
RETF1 DC F'0' COUNT OF FMT 1 DSCBS RET BY CVAFFILT
RETF3 DC F'0' COUNT OF FMT 3 DSCBS RET BY CVAFFILT
WFMTREC DS OCL133 WORK FORMAT RECORD FOR OUTPUT
MSG DC CL29' ' GENERAL MESSAGE
FMTCNT DC CL08' ' FORMAT COUNT
TOEOL DC CL96' ' SPACES TO END OF LINE
*
***********************************************************************
* PRINT LINES *
***********************************************************************
*
DS 0D TAKE CARE OF SLACK BYTES
PDETLINE DS OCL133 DETAIL LINE
DC CL133' '
EPDETLEN EQU *-PDETLINE LENGTH OF DETAIL LINE
*
***********************************************************************
* DCB - OUTPUT FILE (OUTFILE) *
***********************************************************************
*
OUTFILE DCB DDNAME=OUTDD,
DSORG=PS,
RECFM=FBA,
LRECL=133,
MACRF=PM
*
***********************************************************************
* VTDCB DCB AREA *
***********************************************************************
CVAF Macros

***********************************************************************
* VTOCDCB DCB DDNAME=CVAFDD,MACRF=E,EXLST=XLST1,DSORG=PS
XLST1 DC X'87'
  DC AL3(JFCB1)
JFCB1 DS OCL176
TESTNAME DS CL44
  DS CL8
  DS BL1
  DS CL123
*
***********************************************************************
* MAPPPING MACROS *
***********************************************************************
* CVPLMAP ICVAFPL CVPLFSA=YES CVAF PARMLIST
FCLMAP ICVFCL FILTER CRITERIA LIST
BFLMAP ICVAFBFL BUFFER LIST
  PUSH PRINT
  PRINT NOGEN
DSCBMAP DSECT
  DSCB DSECT
IECSDSL1 (1) USE FMT I DSCB MAPPING TO GET BUFFER
DCBD DSORG=XE,DEVD=DA MAP OF DCB
IEZDEB MAP OF DEB
  POP PRINT
*
*
CVAFFEXP CSECT , CONT OF CSECT AFTER MAPPING MACROS
*
***********************************************************************
* CVAF PARAMETER LISTS *
***********************************************************************
* CVPLDEF CVAFFILT MF=L,BRANCH=NO,FLTAREA=KEEP
CVPLDEFA CVAFFILT BRANCH=(YES,SUP),MF=L
*
***********************************************************************
* SPACE ALLOCATION FOR CVPL, FCL, BFL, AND DSCB BUFFERS *
***********************************************************************
* FCLDEF DS (FCLHDLEN+FCLDSNEL)X FCL HEADER AND ONE FCL ELEMENT
FCLSIZE EQU *=FCLDEF
  * DEFINE A CVAF BUFFER LIST WITH
FCLDEF2 DS (FCLHDLEN+2*FCLDSNEL)X FCL HEADER AND TWO FCL ELEMENTS
FCLSIZE2 EQU *=FCLDEF2
  * DEFINE A CVAF BUFFER LIST WITH
  * N BUFFER LIST ELEMENTS
BFLHDEF DS (BFLHLN)X BUFFER LIST HEADER
BFLEDEF DS (BUFNBR*BFLELN)X N BUFFER LIST ELEMENTS
BFLSIZE EQU *=BFLHDEF
  * DEFINE N FULL DSCB BUFFERS
DSCBDEF DS (BUFNBR*DSCBSIZ)X
  * END CVAFFEXP END OF CVAFFEXP

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 65 for additional information.
The format of the CVAFSEQ macro is:

```
/SM590000/SM590000
CVAFSEQ
   label
   ACCESS= GT
   GTEQ
   BUFLIST= addr
   DSN= addr
/SM590000/SM590000
   UCB= (ucbaddr)
   DEB= addr
   DSNONLY= NO
   ARG= addr
/SM590000/SM590000
   NOKEEP
   IOARE= KEEP
   (KEEP, addr)
   (NOKEEP, addr)
   IXRCDS= KEEP
   (KEEP, addr)
   (NOKEEP, addr)
/SM590000/SM590000
   (1)
   NO
   BRANCH= YES
   (YES, SUP)
   (YES, PGM)
```

Notes:
1. If YES is coded, the default is SUP.

**ACCESS:** Specify Relationship between Supplied and Returned DSN

- **ACCESS=GT**
  Specifies that the DSN or argument value is to be used to return a DSCB whose DSN or argument is greater than that supplied.

- **ACCESS=GTEQ**
  Specifies that the DSN or argument value is to be used to return a DSCB whose DSN or argument is greater than or equal to that supplied.

  **Recommendation:** A CVAF call specifying ACCESS=GTEQ should be followed by an ACCESS=GT request, or the same DSCB or name is returned.

**BUFLIST:** Specify One or More Buffer Lists

- **BUFLIST=addr**
  Supplies the address of a buffer list used to read or write DSCBs and VIRs.

**DSN:** Specify Access by DSN Order or by Physical-Sequential Order

- **DSN=addr**
  Supplies the address of a 44-byte area containing either zeros or a data set name. Specifying the DSN keyword causes access of an indexed VTOC by DSN order. BUFLIST is required if DSNONLY=NO is coded or the default.

- **DSN omitted**
  If you omit the DSN keyword, access of an indexed or nonindexed VTOC is by physical-sequential order. BUFLIST is required.

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...
CVAF Macros

(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 66 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.

Recommendations:

- Code the address of the UCB parameter only as register (2-12). Coding an RX-type address gives you unpredictable results.
- Do not use the UCB address passed back in the CVPL from a previous CVAFSEQ request, particularly in AMODE=24, as it may be invalid (because it is a captured and then uncaptured address). The recommendation is to use IOAREA=KEEP.

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 57 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.

Note: Do not switch back and forth between AMODE=24 and AMODE=31
during a succession of CVAFSEQ requests while IOAREA=KEEP is active. This
can cause problems such as ABENDS in CVAF.

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.
CVAF Macros

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.

**Requirement:** 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=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.

**EADSCB=value: Specify the support level for extended attribute DSCBs.**

**EADSCB=OK**

This specification indicates that the calling program supports extended attribute DSCBs. An extended address volume may have these DSCBs.
CVAF Macros

allocated to it. The returned DSCBs (format-3, format-8) may contain extent
descriptors described by 28-bit cylinder addresses or DSCBs (format-9) that
contain additional attribute information.

For calls that initiate physical sequential access (DSN=0 or omitted), a
CVAFSEQ request issued to an EAV volume will be failed if this new,
EADSCB=OK, indicator is not set.

For calls that initiate index order (DSN=address) where the BUFLIST=address
keyword is specified, a CVAFSEQ request issued to an EAV volume will be
failed if the EADSCB=OK indicator is not set and the DSCB associated with
this address is a format-8 DSCB.

The failing error code for these cases will be reflected as follows:
- CVAF status code (CVSTAT) set to STAT082.
- Return code 4

EADSCB=OK will set the CV4EADOK indicator in the CVPL. All other calls to
CVAFSEQ are allowed and EADSCB=OK will be ignored. That is CVAFSEQ
calls with DSNONLY=YES and ARG=address specified.

EADSCB=NOTOK
Indicates a calling program does not support extended attribute DSCBs.
EADSCB=NOTOK will turn off the CV4EADOK indicator in the CVPL. 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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (X'00')</td>
<td>The request was successful.</td>
</tr>
</tbody>
</table>
| 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 139. |
| 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 139. |
### CVAF Macros

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>12 (X'0C')</td>
<td>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.</td>
</tr>
<tr>
<td>16 (X'10')</td>
<td>An I/O error was encountered.</td>
</tr>
</tbody>
</table>

---

**Example of using the CVAFSEQ macro with an indexed VTOC**

This example uses the CVAFSEQ macro to count the number of VSAM 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 VSAM 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
```assembly
STM 14,12,12(13)
BALR 12,0
USING *,12
ST 13,SAVEAREA+4
LA RWORK,SAVEAREA
ST RWORK,8(,13)
LR 13,RWORK
```

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>REGISTERS</strong></td>
<td></td>
</tr>
<tr>
<td>REG1 EQU 1 REGISTER 1</td>
<td></td>
</tr>
<tr>
<td>RWORK EQU 3 WORK REGISTER</td>
<td></td>
</tr>
<tr>
<td>RDEB EQU 4 DEB ADDRESS</td>
<td></td>
</tr>
<tr>
<td>RDSN1 EQU 6 ADDRESS OF DATA SET NAME 1</td>
<td></td>
</tr>
<tr>
<td>RDSN2 EQU 7 ADDRESS OF DATA SET NAME 2</td>
<td></td>
</tr>
<tr>
<td>REG15 EQU 15 RETURN CODE REGISTER 15</td>
<td></td>
</tr>
</tbody>
</table>

---

XC BUFLIST(BFLHLN+BFLLEN),BUFLIST ZERO BUFFER LIST
OI BFLHF1,BFLHSCB DSCBS TO BE READ WITH BUFFER LIST
MVI BFLHNOE,1 ONE BUFFER LIST ENTRY
LA RWORK,DS1MTID ADDRESS OF DSCB BUFFER
ST RWORK,BFLEBUF PLACE IN BUFFER LIST
MVI BFLELTH,DSCBLTH DATA PORTION OF DSCB READ - DSN * SUPPLIED IN CVPL
CVAF Macros

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*
       NAME IS GREATER THAN OR EQUAL TO *
       BUFLIST-BUFLIST, THAT OF DSN1 *
       EADSCB=OK, *
       MF=(E,CVPL)

LOOP EQU * LOOP UNTIL END OF DATA OR DATA SET *
       NAME GREATER THAN DSN2
LTR REG15,REG15 ANY ERROR
BZ TESTDSN BRANCH IF NOT-CHECK DSN LIMIT

*****************************************************
* 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

*****************************************************
* END OF DATA
* *****************************************************
B RELEASE RELEASE CVAF RESOURCES AND RETURN
TESTDSN EQU * IS DATA SET NAME GREATER THAN DSN2
CLI DS1FMID,DS11DC IS THIS A FORMAT 1 DSCB?
BE CKVSAM BRANCH IF FORMAT 1
CLI DS1FMID,DS81DC IS THIS A FORMAT 8 DSCB?
BNE CKLAST BRANCH IF NO. CAN NOT BE VSAM.

CKVSAM EQU * CHECK VSAM
CLC DS1DSNAM,0(RDSN2) HAS LIMIT BEEN REACHED?
BH RELEASE BRANCH IF HIGH TO RELEASE RESOURCES
TM DS1DSORG+1,DS1ORGAM IS DATA SET VSAM
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?
BNE RELOOP BRANCH IF NO-READ NEXT ONE
B RELEASE RELEASE CVAF RESOURCES AND RETURN

RELOOP EQU * READ NEXT DSCB
CVAFSEQ ACCESS=GT, GET DSCB WITH DATA SET NAME *
       EADSCB=OK, GREATER THAN THE ONE LAST READ *
       MF=(E,CVPL)
B LOOP CHECK RESULTS OF CVAFSEQ

OTHERERR EQU * UNEXPECTED ERROR

*****************************************************
* 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 IECVAFBFL 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
Example of using the CVAFSEQ macro to process a volume in physical sequential order

This example will use the CVAFSEQ macro to read through the DSCBs in physical sequential order. Although CVAFSEQ can be used to process both an indexed and non indexed volume in physical sequential order this example uses a non indexed volume. The CVAFSEQ call will return DSCBs where BFLEARG is set to a starting CCHHR. This value is initially set to zero and the CVAFSEQ call uses ACCESS=GT. A buffer list with five buffer list entries is contained within the program and is used to read up to five DSCBs at a time.

Output from this program will be to OUTDD. It will be a list of all the datasets on the volume and their corresponding CCHHRs. The output from this program is based on the volume and dataset information detailed within the source code example. If the output received is different it can be verified using the IEHLLIST utility.

This program must be APF authorized because it uses the UCB= and BRANCH=(YES,PGM) parameters. It is bad programming practice to give a program APF authorization unnecessarily. In this case these two options just give a slight performance improvement. To remove APF authorization from this example, perform these steps:
- Remove the SETCODE and ENTRY statements for the binder.
- Replace the UCB parameter with DEB=(reg). Precede the CVAFSEQ macro with an instruction to load the specified register from DEBADD.
- Remove the BRANCH parameter on CVAFSEQ.

Sample JCL for CVAFSEQ macro to process a volume in sequential order:
The following is the sample JCL used to Assemble, Link, and Execute the example source. Changes will have to be made to this JCL, as appropriate, for each customer environment.

```sql
//SEQXMP2 JOB ,MSGCLASS=X,TIME=(,10),
   NOTIFY=ASYSUID
//*
//STEP01 EXEC PROC=ASMACLG
//SYSIN DD *
   (INCLUDE EXAMPLE SOURCE HERE)
   /*
   /*
   //L.SYSLMOD DD DSN=YOUR.AUTH.LINKLIB(SEQXMP2),DISP=SHR
   //L.SYSIN DD *
      SETCODE AC(1)
      ENTRY SEQXMP2
   /*
   /*
   //G.SYSABEND DD SYSOUT=*    //SYSPRINT DD SYSOUT=*     
   /*
   //G.CVAFFDD DD DISP=SHR,UNIT=3390,Vol=SER=339003
   //G.OUTDD DD DSN=CVAFFSEQ1.OUTPUT,
```
Code example of the CVAFSEQ macro to process a volume in sequential order:

SEQXMP2 TITLE 'CVAF CVAFSEQ - SEQXMP2 EXAMPLE'
SEQXMP2 CSECT
SEQXMP2 AMODE 24
SEQXMP2 RMODE 24
*
***********************************************************************
**
** SEQXMP2 - THIS MODULE WILL READ THROUGH THE DSCBS ON A
** NONINDEXED VOLUME. IT WILL USE THE CVAFSEQ MACRO
** TO READ UP TO 5 DSCBS AT A TIME IN PHYSICAL
** SEQUENTIAL ORDER.
** OUTPUT FROM THIS MODULE WILL BE TO OUTDD. IT WILL
** LIST ALL OF THE DATASETS ON THE VOLUME AND THEIR
** CORRESPONDING CCHHR'S. THE FMT4, FMT5, AND FMT7 DSCBS
** AND THEIR CORRESPONDING CCHHRS WILL NOT BE LISTED.
**
** THE FOLLOWING DATASETS WERE ALLOCATED IN THE FOLLOWING
** ORDER ON A NONINDEXED VOLUME.
** 10 SEQUENTIAL DATASETS: CVAFSEQ1.SEQ01-CVAFSEQ1.SEQ10
** 5 PDS DATASETS: CVAFSEQ1.PDS01-CVAFSEQ1.PDS05
** 5 PDSE DATASETS: CVAFSEQ1.PDSE01-CVAFSEQ1.PDSE05
** 2 VSAM DATASETS: CVAFSEQ1.VSAM01-CVAFSEQ1.VSAM02
**
** NOTE: THE OUTPUT FROM THIS EXAMPLE IS BASED ON USING A NON SMS
** MANAGED NON INDEXED VOLUME FOR ALLOCATING THE TEST
** DATASETS (CVAFDD DD).
** IF A SMS MANAGED NON INDEXED VOLUME IS USED THE POSITION
** OF THE VVDS DATASET WILL BE THE SECOND ENTRY IN THE LIST.
**
** DASD VOLUMES USED IN THIS EXAMPLE:
** - 1 3390-3 WHERE THE OUTDD DATASET IS DEFINED
** - 1 3390-3 WHERE THE TEST DATASETS DETAILED ABOVE ARE DEFINED
** THIS VOLUME INITIALIZED USING: VTOC(3326,0,195) NOINDEX
**
**
** OUTPUT IN OUTDD DATASET SHOULD BE THE FOLLOWING:
**

**
** SEQXMP2 START OF OUTPUT MESSAGES
**
** DSN: CVAFSEQ1.SEQ01   CCHHR: OCFE000003
** DSN: CVAFSEQ1.SEQ02   CCHHR: OCFE000004
** DSN: CVAFSEQ1.SEQ03   CCHHR: OCFE000005
** DSN: CVAFSEQ1.SEQ04   CCHHR: OCFE000006
** DSN: CVAFSEQ1.SEQ05   CCHHR: OCFE000007
** DSN: CVAFSEQ1.SEQ06   CCHHR: OCFE000008
** DSN: CVAFSEQ1.SEQ07   CCHHR: OCFE000009
** DSN: CVAFSEQ1.SEQ08   CCHHR: OCFE00000A
** DSN: CVAFSEQ1.SEQ09   CCHHR: OCFE00000B
** DSN: CVAFSEQ1.SEQ10   CCHHR: OCFE00000C
** DSN: CVAFSEQ1.PDS01   CCHHR: OCFE00000D
** DSN: CVAFSEQ1.PDS02   CCHHR: OCFE00000E
** DSN: CVAFSEQ1.PDS03   CCHHR: OCFE00000F
** DSN: CVAFSEQ1.PDS04   CCHHR: OCFE000010
** DSN: CVAFSEQ1.PDS05   CCHHR: OCFE000011
** DSN: CVAFSEQ1.PDSE01  CCHHR: OCFE000012
**SEQXMP2 - LOGIC NOTES**

- **THIS MODULE WILL PERFORM THE FOLLOWING:**

  **INITIALIZE**
  - Initialize starting CCHHR value to 0 (GT zero used in CVAFSEQ)
  - Obtain the necessary information from the DASD volume
  - Open an output file
  - Write necessary messages to the output file

- **MAINLINE**
  - Do WHILE more datasets on volume to process
  - Load 5 entry table with DSCB address
  - Init buffer list hdr to read DSCBs and starting CCHHR
  - Init buffer list entry with DSCB address and length
  - Issue CVAFSEQ macro to read 5 entries
  - Load CCHHR returned from CVAFSEQ into table
  - Process table entries and produce output: DSN/CCHHR list

- **FINALIZATION**
  - Write necessary messages to the output file
  - Close the output file
  - Exit

- **SEQXMP2 - JOB INFORMATION**

  **NORMAL END OF JOB:**
  - RC=00 and OUTDD output as detailed above

  **ABNORMAL END OF JOB:**
  - ABEND 100 - Error opening VTOC on the DASD volume that is associated with the CVAFDD DD statement
  - ABEND 101 - Error opening the OUTDD dataset
  - ABEND 102 - Error closing the OUTDD dataset

- **HOUSEKEEPING**
  - Save caller’s registers and establish a new register save area

- **STANDARD LINKAGE CONVENTION**
  - STM R14,R12,12(R13)
  - BALR R11,0
  - DCL R11 AS IMPLIED BASE REG
**CVAF Macros**

```assembly
USING BASE,R11,R12 R12 IS ALSO BASE REG
BASE L R12,BASEADDR SET UP ADDRESSING FOR R12
B CVAFSQ00 BRANCH AROUND DECLARES
BASEADDR DC A(BASE+4096) ADDRESSING FOR R12
CVAFSQ00 DS OH CONTINUE...
ST R13,SAVE+4 SAVE PTR TO CALLER'S SAVE AREA
* LA R14,SAVE GET ADDRESS OF THE NEW SAVE AREA
* ST R14,8,(R13) CHAIN CALLER'S AREA TO OURS
LR R13,R14 ESTABLISH THE NEW SAVE AREA

***********************************************************************
** INITIALIZATION **
***********************************************************************

INITIAL DS 0H INITIALIZATION SECTION
MVC CCHHR$1,CCHHR$0 INIT CCHHR START TO ZERO
BAL R14,1DVolRTN INVOKE RTN TO IDENTIFY THE VOLUME(S)
BAL R14,OPENRTN INVOKE OPEN OUTPUT DATASET RTN
MVC PDETLINE(133),STRMSG MOVE START MSG TO LINE
PUT OUTFILE,PDETLINE WRITE A RECORD TO THE OUTPUT FILE
MVC PDETLINE(133),BLNKLNE MOVE BLANK LINE TO LINE
PUT OUTFILE,PDETLINE WRITE A RECORD TO THE OUTPUT FILE

***********************************************************************
** MAINLINE **
***********************************************************************

MAINLINE DS 0H MAINLINE SECTION
MVI SWEOD,NODEO SET SWITCH TO NO EOD INITIALLY
BAL R14,LDTABRTN INVOKE LDTABRTN TO LOAD TABLE
BAL R14,INITBRTN INVOKE INITBRTN TO INIT BUFFER LIST
BAL R14,CVAFSRTN INVOKE CVAFSRTN TO ISSUE CVAFSEQ CALL
BAL R14,LODCRTN INVOKE LODCRTN TO LOAD CCHHR IN TBL
BAL R14,PRTBRTN INVOKE PRTBRTN TO PROCESS TBL ENTRIES
CLI SWEOD,EOD DID WE REACH THE END OF DATA?
BNE MAINLINE NO, PROCESS MORE DATA
L R15,RETCODE LOAD R15 WITH RETURN CODE
CH R15,RCODE00 IF RC WAS 00?
BNE FINAL NO - DO NOT PRINT EOD MESSAGE
MVC PDETLINE(133),BLNKLNE YES - MOVE BLANK LINE TO LINE
PUT OUTFILE,PDETLINE WRITE THE REC TO OUTPUT FILE
MVC PDETLINE(133),EDMSG MOVE EOD MSG TO LINE
PUT OUTFILE,PDETLINE WRITE THE REC TO OUTPUT FILE

***********************************************************************
** FINALIZATION **
***********************************************************************

FINAL DS 0H FINALIZATION SECTION
MVC PDETLINE(133),BLNKLNE MOVE BLANK LINE TO LINE
PUT OUTFILE,PDETLINE WRITE A RECORD TO THE OUTPUT FILE
MVC PDETLINE(133),ENDMSG MOVE END MSG TO LINE
PUT OUTFILE,PDETLINE WRITE A RECORD TO THE OUTPUT FILE
BAL R14,CLOSERTN INVOKE CLOSE OUTPUT DATASET RTN
L R13,4(R13) RESTORE REGISTER
LM R14,R12,12(R13) RESTORE CALLERS REGISTERS
LA R15,0 SET RC TO 0
BR R14 RETURN TO CALLER

***********************************************************************
** OPENRTN **
***********************************************************************
```

Chapter 1. Using the Volume Table of Contents  131
**CVAF Macros**

---

**OPENRTN**

- ROUTINE TO OPEN OUTPUT FILE USED BY THIS MODULE

```
OPENRTN DS 0H
ST R14,OPENSAVE
OPEN (OUTFILE,(OUTPUT))
TM OUTFILE+(DCBIOFLGS-IHADCB),DCBOFOPN IS FILE OPEN
BO OPENEXIT FILE OPEN OK - EXIT OPEN RTN
LA R1,EABN101
BAL R14,ABENDRTN INVOKE ABEND ROUTINE
OPENEXIT DS 0H
L R14,OPENSAVE
BR R14
```

**CLOSERTN**

- ROUTINE TO CLOSE OUTPUT FILE USED BY THIS MODULE

```
CLOSERTN DS 0H
ST R14,CLOSSAVE
CLOSE (OUTFILE)
LTR R15,R15 CHECK IF CLOSED OK
BZ CLOSEXIT IF OK BRANCH TO CLOSEXIT
LA R1,EABN102 ELSE SETUP FOR USER ABEND 102
BAL R14,ABENDRTN INVOKE ABEND ROUTINE
CLOSEXIT DS 0H EXIT FROM CLOSE ROUTINE
L R14,CLOSSAVE
BR R14
```

**ABENDRTN**

- FORCE AN ABEND ROUTINE

```
ABENDRTN DS 0H
ST R14,ABENSAVE
ABEND (R1),DUMP ISSUE USER ABEND WITH DUMP
ABENEXIT DS 0H EXIT FROM ABEND ROUTINE
L R14,ABENSAVE
BR R14
```

**IDVOLRTN**

- OBTAIN THE NECESSARY INFORMATION FROM THE DASD VOLUME

```
IDVOLRTN DS 0H
ST R14,IDVLSAVE
RDJFCB (VTOCDCB,(INPUT)) READ JFCB / OPEN VTOC
MVI JFCB1,X'04' PUT IN ID FOR FORMAT 4
MVC JFCB1+1(43),JFCB1 SETUP FOR VTOC OPEN
OPEN (VTOCDCB,(INPUT)),TYPE=J OPEN VTOC (OPEN TYPE=J)
TM VTOCDCB+(DCBIOFLGS-IHADCB),DCBOFOPN TEST FOR GOOD OPEN
BO IDVOL010 BRANCH TO IDVOL010 - GOOD OPEN
LA R1,EABN100 ELSE SETUP FOR USER ABEND 100
BAL R14,ABENDRTN INVOKE ABEND ROUTINE
IDVOL010 DS 0H
ST R14,IDVLSAVE
L R14,IDVLSAVE
```

---

CVAF Macros

- ROUTINE TO OPEN OUTPUT FILE USED BY THIS Module

```
OPENRTN DS 0H
ST R14,OPENSAVE
OPEN (OUTFILE,(OUTPUT))
TM OUTFILE+(DCBIOFLGS-IHADCB),DCBOFOPN IS FILE OPEN
BO OPENEXIT FILE OPEN OK - EXIT OPEN RTN
LA R1,EABN101
BAL R14,ABENDRTN INVOKE ABEND ROUTINE
OPENEXIT DS 0H
L R14,OPENSAVE
BR R14
```

**CLOSERTN**

- ROUTINE TO CLOSE OUTPUT FILE USED BY THIS MODULE

```
CLOSERTN DS 0H
ST R14,CLOSSAVE
CLOSE (OUTFILE)
LTR R15,R15 CHECK IF CLOSED OK
BZ CLOSEXIT IF OK BRANCH TO CLOSEXIT
LA R1,EABN102 ELSE SETUP FOR USER ABEND 102
BAL R14,ABENDRTN INVOKE ABEND ROUTINE
CLOSEXIT DS 0H EXIT FROM CLOSE ROUTINE
L R14,CLOSSAVE
BR R14
```

**ABENDRTN**

- FORCE AN ABEND ROUTINE

```
ABENDRTN DS 0H
ST R14,ABENSAVE
ABEND (R1),DUMP ISSUE USER ABEND WITH DUMP
ABENEXIT DS 0H EXIT FROM ABEND ROUTINE
L R14,ABENSAVE
BR R14
```

**IDVOLRTN**

- OBTAIN THE NECESSARY INFORMATION FROM THE DASD VOLUME

```
IDVOLRTN DS 0H
ST R14,IDVLSAVE
RDJFCB (VTOCDCB,(INPUT)) READ JFCB / OPEN VTOC
MVI JFCB1,X'04' PUT IN ID FOR FORMAT 4
MVC JFCB1+1(43),JFCB1 SETUP FOR VTOC OPEN
OPEN (VTOCDCB,(INPUT)),TYPE=J OPEN VTOC (OPEN TYPE=J)
TM VTOCDCB+(DCBIOFLGS-IHADCB),DCBOFOPN TEST FOR GOOD OPEN
BO IDVOL010 BRANCH TO IDVOL010 - GOOD OPEN
LA R1,EABN100 ELSE SETUP FOR USER ABEND 100
BAL R14,ABENDRTN INVOKE ABEND ROUTINE
IDVOL010 DS 0H
ST R14,IDVLSAVE
L R14,IDVLSAVE
```

---

CVAF Macros

- ROUTINE TO OPEN OUTPUT FILE USED BY THIS Module

```
OPENRTN DS 0H
ST R14,OPENSAVE
OPEN (OUTFILE,(OUTPUT))
TM OUTFILE+(DCBIOFLGS-IHADCB),DCBOFOPN IS FILE OPEN
BO OPENEXIT FILE OPEN OK - EXIT OPEN RTN
LA R1,EABN101
BAL R14,ABENDRTN INVOKE ABEND ROUTINE
OPENEXIT DS 0H
L R14,OPENSAVE
BR R14
```

**CLOSERTN**

- ROUTINE TO CLOSE OUTPUT FILE USED BY THIS MODULE

```
CLOSERTN DS 0H
ST R14,CLOSSAVE
CLOSE (OUTFILE)
LTR R15,R15 CHECK IF CLOSED OK
BZ CLOSEXIT IF OK BRANCH TO CLOSEXIT
LA R1,EABN102 ELSE SETUP FOR USER ABEND 102
BAL R14,ABENDRTN INVOKE ABEND ROUTINE
CLOSEXIT DS 0H EXIT FROM CLOSE ROUTINE
L R14,CLOSSAVE
BR R14
```

**ABENDRTN**

- FORCE AN ABEND ROUTINE

```
ABENDRTN DS 0H
ST R14,ABENSAVE
ABEND (R1),DUMP ISSUE USER ABEND WITH DUMP
ABENEXIT DS 0H EXIT FROM ABEND ROUTINE
L R14,ABENSAVE
BR R14
```

**IDVOLRTN**

- OBTAIN THE NECESSARY INFORMATION FROM THE DASD VOLUME

```
IDVOLRTN DS 0H
ST R14,IDVLSAVE
RDJFCB (VTOCDCB,(INPUT)) READ JFCB / OPEN VTOC
MVI JFCB1,X'04' PUT IN ID FOR FORMAT 4
MVC JFCB1+1(43),JFCB1 SETUP FOR VTOC OPEN
OPEN (VTOCDCB,(INPUT)),TYPE=J OPEN VTOC (OPEN TYPE=J)
TM VTOCDCB+(DCBIOFLGS-IHADCB),DCBOFOPN TEST FOR GOOD OPEN
BO IDVOL010 BRANCH TO IDVOL010 - GOOD OPEN
LA R1,EABN100 ELSE SETUP FOR USER ABEND 100
BAL R14,ABENDRTN INVOKE ABEND ROUTINE
IDVOL010 DS 0H
ST R14,IDVLSAVE
L R14,IDVLSAVE
```

---
LDTABRTN DS 0H LOAD TABLE ROUTINE
ST R14,LDTBSAVE STORE C(R14) INTO SAVE AREA
LA R6,DSCB01 LOAD R6 WITH ADDRESS OF DSCB01
ST R6,TDSCB01 STORE ADDRESS OF DSCB01 INTO TABLE
LA R6,DSCB02 LOAD R6 WITH ADDRESS OF DSCB02
ST R6,TDSCB02 STORE ADDRESS OF DSCB02 INTO TABLE
LA R6,DSCB03 LOAD R6 WITH ADDRESS OF DSCB03
ST R6,TDSCB03 STORE ADDRESS OF DSCB03 INTO TABLE
LA R6,DSCB04 LOAD R6 WITH ADDRESS OF DSCB04
ST R6,TDSCB04 STORE ADDRESS OF DSCB04 INTO TABLE
LA R6,DSCB05 LOAD R6 WITH ADDRESS OF DSCB05
ST R6,TDSCB05 STORE ADDRESS OF DSCB05 INTO TABLE
LDTBEXIT DS 0H EXIT FROM LDTABRTN
L R14,LDTBSAVE LOAD C(LDTBSAVE) INTO R14
BR R14 EXIT

INITBRTN DS 0H INITIALIZE BUFFER LIST ROUTINE
ST R14,INITSAVE STORE C(R14) INTO SAVE AREA
LA R7,BUFLISTE LOAD R7 WITH ADDRESS OF BUFLIST ENTRY
USING BFLE,R7 ESTABLISH ADDRESSABILITY TO BFLE
LA R8,BUFLISTH LOAD R8 WITH ADDRESS OF BUFLIST HDR
USING BFLHDR,R8 ESTABLISH ADDRESSABILITY TO BFLHDR
LA R2,TABLE LOAD R2 WITH ADDRESS OF TABLE
USING TBLMAP,R2 ESTABLISH ADDRESSABILITY USING TBLMAP
XC BFLHDR(BFLHLN+TBLNBR*BFLELN),BFLHDR CLEAR BUFLIST
OI BFLHLF,BFLHDCB DSCBs TO BE READ WITH BUFFER LIST
MVC BFLEARG,CCHARS MOVE STARTING CCHAR TO ARG
MVI BFLHNOE,TBLNBR MOVE NBR OF TBL ENTRIES TO BUFFER NBR
SLR R10,R10 INIT R10 WITH ZERO
IC R10,BFLHNOE NBR OF BUFFER ENTRIES IN R10
ST R10,COUNT NBR OF BUFFER ENTRIES IN COUNT
INIT0010 DS 0H INITIAL BUFFER LIST WITH DSCB ADDR/LENG
L R6,DSCBA LOAD R6 WITH DSCB ADDRESS FROM TABLE
ST R6,BFLEBUF-BFLE(,R7) PLACE IN BUFFER LIST
MVI BFLELTH-BFLE(R7),DSCBLTH FULL DSCB READ
OI BFLEFL,BFLECHR CCHR TO BE RETURNED
LA R2,TBLLNG(,R2) POINT TO NEXT TABLE ENTRY
LA R7,BFLELN(,R7) POINT TO NEXT BUFFER LIST ENTRY
BCT R10,INIT0010 BRANCH TO INIT0010 IF C(R10) GT ZERO
DROP R2 DROP R2
INITEXIT DS 0H EXIT FROM INITBRTN
L R14,INITSAVE LOAD C(INITSAVE) INTO R14
BR R14 EXIT

CVAFSEQ RTN DS 0H CVAFSEQ REQUEST ROUTINE
ST R14,CVAFSIZE STORE C(R14) INTO SAVE AREA
L RUCB,UCBADD LOAD R4 WITH UCB ADDRESS
CVAFSEQ UCB=(RUCB), CVAFSEQ MACRO INVOCATION X

Chapter I. Using the Volume Table of Contents 133
CVAF Macros

```
EADSCB=OK, X
BRANCH=(YES,PGM), X
MF=(E,CVPL)

ST R15,RETCODE STORE RC INTO RETCODE
LTR R15,R15 TEST RC RETURNED FROM CVAFSEQ
CH R15,RCODE00 IS IT A RC00?
BZ CVAFEXIT YES - BRANCH TO EXIT ROUTINE
CH R15,RCODE04 IS IT A RC04?
BE CVAFSR04 YES - BRANCH TO PROCESS RC04
CH R15,RCODE08 IS IT A RC08?
BE CVAFSR08 YES - BRANCH TO PROCESS RC08
CH R15,RCODE12 IS IT A RC12?
BE CVAFSR12 YES - BRANCH TO PROCESS RC12
CH R15,RCODE16 IS IT A RC16?
BE CVAFSR16 YES - BRANCH TO PROCESS RC16
MVC PDETLINE(133),RCERMSG ELSE FORMAT UNEXPECTED RC
PUT OUTFILE,PDETLINE WRITE A RECORD TO THE OUTPUT FILE
B CVAFEXIT EXIT ROUTINE

CVAFSR04 DS 0H RETURN CODE 04
MVC PDETLINE(133),RC04MSG FORMAT RC04 MESSAGE
PUT OUTFILE,PDETLINE WRITE A RECORD TO THE OUTPUT FILE
CLI CVSTAT,STAT032 IS CVSTAT CODE STAT032?
BE CVAFSS32 YES - BRANCH TO PROCESS STAT032
MVC PDETLINE(133),STAT2MSG NO - FORMAT STAT2MSG
PUT OUTFILE,PDETLINE WRITE A RECORD TO THE OUTPUT FILE
B CVAFEXIT BRANCH TO EXIT ROUTINE

CVAFSS32 DS 0H CVSTAT CODE = STAT032
MVC PDETLINE(133),STAT032MSG FORMAT STAT032MSG
PUT OUTFILE,PDETLINE WRITE A RECORD TO THE OUTPUT FILE
B CVAFEXIT BRANCH TO EXIT ROUTINE

CVAFSR08 DS 0H RETURN CODE 08
MVC PDETLINE(133),RC08MSG FORMAT RC08 MESSAGE
PUT OUTFILE,PDETLINE WRITE A RECORD TO THE OUTPUT FILE
B CVAFEXIT BRANCH TO EXIT ROUTINE

CVAFSR12 DS 0H RETURN CODE 12
MVC PDETLINE(133),RC12MSG FORMAT RC12 MESSAGE
PUT OUTFILE,PDETLINE WRITE A RECORD TO THE OUTPUT FILE
B CVAFEXIT BRANCH TO EXIT ROUTINE

CVAFSR16 DS 0H RETURN CODE 16
MVC PDETLINE(133),RC16MSG FORMAT RC16 MESSAGE
PUT OUTFILE,PDETLINE WRITE A RECORD TO THE OUTPUT FILE
B CVAFEXIT EXIT FROM CVAFSRTN

L R14,CVAFSAVE LOAD C(CVAFSAVE) INTO R14
BR R14 EXIT

***********************************************************************
* LODCRTN *
* - LOAD CCHHR FROM BUFFER LIST ENTRY INTO PROCESSING TABLE. THIS *
* VALUE RETURNED FROM CVAFSEQ CALL. *
***********************************************************************

LODCRTN DS 0H LOAD CCHHR INTO PROCESSING TABLE
ST R14,LODCSAVE STORE C(R14) INTO SAVE AREA
DROP R7,R8 DROP R7 AND R8
LA R7,BUFLISTE LOAD R7 WITH ADDRESS OF BUFLIST ENTRY
USING BFLE,R7 ESTABLISH ADDRESSTABILITY TO BFLE
LA R2,TABLE LOAD R2 WITH ADDRESS OF TABLE
USING TBLMAP,R2 ESTABLISH ADDRESSTABILITY USING TBLMAP
L R10,COUNT LOAD R10 WITH TABLE ENTRY COUNT

LODC0010 DS 0H PROCESS CCHHR VALUE RETURNED
LA R8,BFLEARG LOAD R8 WITH ADDRESS OF BFLEARG
ST R8,CCHHRA STORE CCHHR VALUE INTO TABLE
LA R2,TBLLNG(,R2) POINT TO NEXT TABLE ENTRY
LA R7,BFLELN(,R7) POINT TO NEXT BUFFER LIST ENTRY
BCT R10,LODC0010 BRANCH TO LODC0010 IF C(R10) GT ZERO
DROP R2,R7 DROP R2 AND R7
```
LODCEXIT DS 0H    EXIT FROM LODCRTN
   L    R14,LODCSAVE    LOAD (LODCSAVE) INTO R14
   BR    R14    EXIT

***********************************************************************
* PRTBRTN *
* - PROCESS TABLE WHICH CONTAINS ADDRESS OF DSCB AND ADDRESS OF *
* CCHHR FOR EACH ENTRY RETURNED FROM CVAFSEQ CALL. TABLE IS *
* CURRENTLY 5 ENTRIES. *
***********************************************************************

PRTBRTN DS 0H    PROCESS TABLE ENTRIES (DSCBA/CCHHRA)
   ST    R14,PRTBSAVE    STORE C(R14) INTO SAVE AREA
   L    R10,COUNT    LOAD COUNT IN R10
   LA    R2,TABLE    LOAD ADDRESS OF TABLE INTO R2
   USING TBLMAP,R2    ESTABLISH ADDRESSABILITY TO TABLE

PRTB0000 DS 0H    PROCESS ENTRIES
   L    R3,DSCBA    ADDRESSABILITY TO DSCBA
   L    R4,CCHHRA    ADDRESSABILITY TO CCHHRA
   CLC    0(1,R3),FMT4    IS IT A FMT4?
          BE    PRTB0020    YES, BRANCH TO POINT TO NEXT ENTRY
   CLC    0(1,R3),FMT5    IS IT A FMT5?
          BE    PRTB0020    YES, BRANCH TO POINT TO NEXT ENTRY
   CLC    0(1,R3),FMT7    IS IT A FMT7?
          BE    PRTB0020    YES, BRANCH TO POINT TO NEXT ENTRY
   CLC    0(1,R3),FMT9    IS IT A FMT9?
          BE    PRTB0020    YES, BRANCH TO POINT TO NEXT ENTRY

* DETERMINE IF END OF DATA WAS REACHED
   CLC    0(1,R3),NODSN    IS THERE '00' IN FIRST BYTE
          BNE    PRTB0010    NO, THEN CONTINUE TO PROCESS DSN
   MVI    SWEOD,EOD    YES, SET SWITCH TO END OF DATA
   B    PRTBEXIT    EXIT OUT OF ROUTINE

PRTB0010 DS 0H    PROCESS DSN - FORMAT
   MVC    DSNMSG(44),0(R3)    MOVE DSN TO PRINT LINE
   * PROCESS / FORMAT CCHHR
      MVC    CCHHRS(5),0(R4)    MOVE CCHHR TO CCHHRS START VARIABLE
      UNPK    CCHHRM(L'CCHHRM+1),CCHHRS(6)    UNPACK CCHHR
      TR    CCHHRM,TCHAR1    CONVERT TO PRINTABLE HEX
      PUT    OUTFILE,MSG1    PRINT THE DSN LINE

PRTB0020 DS 0H    INCREMENT COUNTER FILEEDIT
   LA    R2,TBLLLNG(R2)    POINT TO NEXT TABLE ENTRY
   BCT    R10,PRTB0000    BRANCH TO PRTB0000 IF C(R10) GT ZERO

PRTBEXIT DS 0H    EXIT FROM PRTBRTN
   L    R14,PRTBSAVE    LOAD C(PRTBSAVE) INTO R14
   BR    R14    EXIT

***********************************************************************
* WORKING STORAGE *
***********************************************************************

DS 0D
DC    CL36'SEQXMP2-WORKING STORAGE BEGINS HERE'

***********************************************************************
* EQUATES *
***********************************************************************

EABN100 EQU 100    USER ABEND CODE 100-VTOC OPEN ERROR
EABN101 EQU 101    USER ABEND CODE 101-OUTDD OPEN ERROR
EABN102 EQU 102    USER ABEND CODE 102-OUTDD CLOSE ERROR
R0    EQU 0
R1    EQU 1
RDEB EQU 1    REG1 FOR DEB ADDRESS
R2    EQU 2
R3    EQU 3
R4    EQU 4
RUCB EQU 4    REG4 FOR UCB ADDRESS
CVAF Macros

R5 EQU 5
R6 EQU 6
R7 EQU 7
R8 EQU 8
R9 EQU 9
R10 EQU 10
R11 EQU 11
R12 EQU 12
R13 EQU 13
R14 EQU 14
R15 EQU 15

***********************************************************************
* SAVE AREAS *
***********************************************************************
*
SAVE DC 18F'0' MAIN PROGRAM SAVE AREA
OPENSAVE DC F'0' OPEN FILES ROUTINE SAVE AREA
CLOSESAVE DC F'0' CLOSE FILES ROUTINE SAVE AREA
ABENSAVE DC F'0' ABEND ROUTINE SAVE AREA
IDVSSAVE DC F'0' IDENTIFY VOLUME ROUTINE SAVE AREA
LDTBSAVE DC F'0' LOAD TABLE ROUTINE SAVE AREA
INITSAVE DC F'0' INIT BUFFER ROUTINE SAVE AREA
PRTBSAVE DC F'0' PROCESS TABLE ROUTINE SAVE AREA
CVAFSAVE DC F'0' CVAFSEQ REQUEST ROUTINE SAVE AREA
LODCSAVE DC F'0' LOAD CCHHR ROUTINE SAVE AREA
*
***********************************************************************
* CONSTANTS *
***********************************************************************
*
RETCODE DC F'0' RETURN CODE SAVE FIELD
RCODE00 DC H'0' RETURN CODE 0 - HALFWORD
RCODE04 DC H'4' RETURN CODE 4 - HALFWORD
RCODE08 DC H'8' RETURN CODE 8 - HALFWORD
RCODE12 DC H'12' RETURN CODE 12 - HALFWORD
RCODE16 DC H'16' RETURN CODE 16 - HALFWORD
CCHHR0 DC XLS'0000000000' INIT WITH ZERO
FMT4 DC XLL'04' FMT4?
FMT5 DC XLL'05' FMT5?
FMT7 DC XLL'07' FMT7? (ONLY CERTAIN DEVICE TYPES)
FMT9 DC XLL'09' FMT9? (ONLY CERTAIN DEVICE TYPES)
NODSN DC XLL'00' END OF DATA?
BLNKLNE DC CL133'
STRTMSC DC CL133' SEQXMP2 START OF OUTPUT MESSAGES
ENQMSG DC CL133' SEQXMP2 END OF OUTPUT MESSAGES
RC00MSG DC CL133' RC00 RETURNED FROM SEQXMP2
RC04MSG DC CL133' RC04 RETURNED FROM SEQXMP2
RC08MSG DC CL133' RC08 RETURNED FROM SEQXMP2
RC12MSG DC CL133' RC12 RETURNED FROM SEQXMP2
RC16MSG DC CL133' RC16 RETURNED FROM SEQXMP2
RCERMSG DC CL133' UNEXPECTED RC (??) RETURNED FROM SEQXMP2
STATIMSG DC CL133' CVSTAT CODE STAT032 ENCOUNTERED
STAT2IMSG DC CL133' UNEXPECTED CVSTAT CODE RETURNED FROM SEQXMP2
EOOMSG DC CL133' END OF DATA REACHED - ALL DATA SETS PROCESSED
MSG1 DS 0CL133
DC CL6' DSN: '
DSNMSG DS CL44'
DC CL7'CCHHR: '
CCHHRM DS CL10'
DC CL66'

***********************************************************************
* WORK AREAS *
***********************************************************************
*
BUFLST DS 0F BUFFER LIST WORK AREA
BUFLISTH DC 2F'0' BUFFER LIST HEADER
CVAF Macros

BUFLISTE DC 1F'0' 5 BUFFER LIST ENTRIES
*
UCBADD DC F'0' UCB ADDRESS SAVE AREA
DEBADD DC F'0' DEB ADDRESS SAVE AREA
COUNT DC F'0' TABLE COUNTER
DS D
DSCB01 DS XL140 DSCB01 BUFFER AREA
DSCB02 DS XL140 DSCB02 BUFFER AREA
DSCB03 DS XL140 DSCB03 BUFFER AREA
DSCB04 DS XL140 DSCB04 BUFFER AREA
DSCB05 DS XL140 DSCB05 BUFFER AREA
*
CCHRS DC XL5'0000000000' STARTING CCHR
*
***********************************************************************
* PRINT LINES *
***********************************************************************
*
DS 0D
PDETLINE DS OCL133 DETAIL LINE
DC CL133'
EPDETLEN EQU *-PDETLINE LENGTH OF DETAIL LINE
*
***********************************************************************
* DCB - OUTPUT FILE (OUTFILE) *
***********************************************************************
*
OUTFILE DCB DDNAME=OUTDD, X
DSORG=PS, X
RECFM=FBA, X
LRECL=133, X
MACRF=PM
*
***********************************************************************
* VTOC DCB AREA *
***********************************************************************
*
VTOCDCB DCB DDNAME=CVAFDD,MACRF=E,EXLST=XLST1,DSORG=PS,DCBE=VTOCDCBE
XLST1 DC X'87'
JFCB1 DS OCL176
TESTNAME DS CL44
DS CL8
DS BL1
DS CL123
VTOCDCBE DCBE EADSCB=OK
*
***********************************************************************
* TABLES *
***********************************************************************
*
TABLE DC 0H START OF TABLE DSCB ADDR / CCHR ADDR
*
TDSCB01 DS F'0' DSCB01 ADDRESS (140 BYTE DSCB)
TCHR01 DS F'0' CCHR ADDRESS FOR DSCB01 - RETURNED
*
TBLLLNG EQU *-TABLE LENGTH OF TABLE ENTRY
*
TDSCB02 DS F'0' DSCB02 ADDRESS (140 BYTE DSCB)
TCHR02 DS F'0' CCHR ADDRESS FOR DSCB02 - RETURNED
*
TDSCB03 DS F'0' DSCB03 ADDRESS (140 BYTE DSCB)
TCHR03 DS F'0' CCHR ADDRESS FOR DSCB03 - RETURNED
*
TDSCB04 DS F'0' DSCB04 ADDRESS (140 BYTE DSCB)
TCHR04 DS F'0' CCHR ADDRESS FOR DSCB04 - RETURNED
**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 94 for an example of using the CVAFTST macro with the CVAFDSM macro.

The format of the CVAFTST macro is:

```
label-CVAFTST-UCB=(reg)
```

**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 16 MB 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.

**Recommendation:** 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 UCBSMAC macro and specifying the COPY, UCBAREA, CMXTAREA, and DCEAREA keywords. The UCB copy and common extension copy must be below the 16 MB line and on a word boundary. Data accessed with DCEAREA can be above the 16 MB line. Refer to [z/OS HCD Planning](#) for details.

**Return Codes from CVAFTST**

On return from CVAF, register 15 contains one of the following return codes:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (X'00')</td>
<td>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.</td>
</tr>
<tr>
<td>4 (X'04')</td>
<td>The system supports an indexed VTOC, but the volume has a nonindexed VTOC.</td>
</tr>
<tr>
<td>8 (X'08')</td>
<td>The system supports an indexed VTOC and the volume has an indexed VTOC.</td>
</tr>
<tr>
<td>12 (X'0C')</td>
<td>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.</td>
</tr>
<tr>
<td>16 (X'10')</td>
<td>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.</td>
</tr>
</tbody>
</table>

**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 [z/OS DFSMSdfp Diagnosis](#).

---

**VTOC Error Responses**

The following actions are taken if an error occurs in the VTOC:

- 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 record 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 139.

**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.
CVAF Macros

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 z/OS DFSMSdfp Diagnosis.

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 information covers 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.

To convert a non-indexed VTOC to an indexed VTOC, use the BUILDIX command with the IXVTOC keyword. The reverse operation can be performed by using the BUILDIX command and specifying the OSVTOC keyword.

To refresh a volume VTOC and INDEX in its current format, use the ICKDSF command REFORMAT with the RVTOC keyword. To optionally extend the VTOC and INDEX, use the ICKDSF command REFORMAT with the EXTVOCT and EXTINDEX keywords.


Protecting the VTOC and VTOC Index

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

RACF®

You can protect the VTOC and VTOC index using the Resource Access Control Facility (RACF), a component of the Security Server for z/OS, 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 z/OS Security Server RACF Security Administrator’s Guide.

APF

The authorized program facility (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
Managing the VTOC

- Rename a data set to a name that begins with SYS1.VTOCIX.

For additional information on using APF, see z/OS MVS Programming: Authorized Assembler Services Guide.

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 with the DASDVOL class. For additional information, see Chapter 6, “Using Password Protected Data Sets,” on page 247.

Copying/Restoring/Initializing the VTOC

The following topics discuss VTOC considerations when updating volumes:

- “Volumes Containing a Nonindexed VTOC”
- “Volumes Containing an Indexed VTOC”

Volumes Containing a Nonindexed VTOC

When updating volumes containing a non-indexed VTOC keep the following considerations in mind:

- 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, the system can automatically update the UCB with the new VTOC location and new volume serial number following a Restore or Copy Volume operation if you have the REFUCB function enabled. To enable this function, do one of the following:
  - Specify ENABLE(REFUCB) in the DEVSUPxx parmlib member
  - Issue the MODIFY command as follows:
    ```
    F DEVMAN,ENABLE(REFUCB)
    ```

    If you do not enable REFUCB, you cannot restore on a volume with an indexed VTOC. In that case, 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 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 nonindexed VTOC or from nonindexed VTOC to indexed VTOC, it generally is advisable to vary the device offline to the sharing system or systems before beginning the operation. However it is not necessary to vary the volume offline on other systems in the same sysplex z/OS Version 1 Release 5 or higher using ICKDSF release 17.

Volumes Containing an Indexed VTOC

Use Device Support Facilities (ICKDSF) to convert a VTOC to a non-indexed format to update the volume. If you do not, keep the following considerations in mind:

- 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.

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. 31-bit programs must pass a clean UCB addr on RENAME, when applicable.

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. See Table 22 on page 148

Authorized callers of SCRATCH can prevent erasure of the data by setting bit 22 to 1, which overrides the RACF profile ERASE attribute.
Managing the VTOC

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 z/OS DFSMS Access Method Services Commands 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 or an z/OS UNIX file. You will receive unpredictable results if you use SCRATCH for z/OS UNIX 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 247.

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, you can scratch the data set only if you have ALTER access authority to either the data set/volume serial in the DATASET class or to the volume serial in the DASDVOL class.

Requirement: 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.

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

### SCRATCH and CAMLST Macro Specification

The format of the SCRATCH and CAMLST macros is:

```
   /SM590000/SM590000
   SCRATCH listname_addrx
   /SM590000/SM590000
   /SM590000/SM590000
   listname CAMLST SCRATCH, dsname_relexp, vol_list_relexp
   /SM590000/SM590000
   , , 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
SCRATCH DELABC
SHOW NO UCB IS SUPPLIED
DELETE DATA SET A.B.C
FROM TWO VOLUMES,
Ignoring expiration
DATE IN THE DSCB
```
Managing the VTOC

```
DELABC CAMLST SCRATCH,DSABC,,VOLIST,,OVRD
DSABC DC CL4'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
```

**Recommendation:** 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.

**SCRATCH Parameter List**

The CAMLST macro generates the SCRATCH parameter list, but your code can set several options that CAMLST does not support. See Table 22.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length or Bit Pattern</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0(0)</td>
<td>1</td>
<td>Flags. Always X'41'.</td>
</tr>
<tr>
<td>1(1)</td>
<td>1</td>
<td>Flags.</td>
</tr>
<tr>
<td>.1.. ....</td>
<td></td>
<td>Do not delete the Resource Access Control Facility (RACF) profile. Has an effect only if JSCBPASS is on. JSCBPASS is on if the program properties table gives authority to bypass security. See the NOPASS option on the primary POI task (PPT) statement in the SCHEDxx member of SYS1.PARMLIB as described in MVS Initialization and Tuning Reference.</td>
</tr>
<tr>
<td>x.xx xxxx</td>
<td>Reserved.</td>
<td></td>
</tr>
<tr>
<td>2(2)</td>
<td>1</td>
<td>Flags.</td>
</tr>
<tr>
<td>1... ....</td>
<td></td>
<td>SYSZTIOT already is enqueued. SCRATCH bypasses enqueueing on SYSZTIOT if this bit is on and the caller is APF-authorized or is running in a system key or supervisor state. If SYSZTIOT is not already enqueued, system damage might result.</td>
</tr>
<tr>
<td>.10. ...0</td>
<td>Always set for SCRATCH.</td>
<td></td>
</tr>
<tr>
<td>...1 ...0</td>
<td>OVRD coded on CAMLST.</td>
<td></td>
</tr>
<tr>
<td>.... 1...</td>
<td>Bypass RACF profile checking. SCRATCH bypasses RACF profile checking if this bit is on and the caller is APF-authorized or is running in a system key or supervisor state.</td>
<td></td>
</tr>
</tbody>
</table>
Table 22. SCRATCH Parameter List (continued)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length or Bit Pattern</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.... 1..</td>
<td>Erase all allocated space for the data set.</td>
<td></td>
</tr>
<tr>
<td>.... 1.</td>
<td>Do not erase allocated space. SCRATCH bypasses space erasure if this bit is on and the caller is APF-authorized or is running in a system key or supervisor state.</td>
<td></td>
</tr>
<tr>
<td>3(3)</td>
<td>1</td>
<td>Reserved.</td>
</tr>
<tr>
<td>4(4)</td>
<td>4</td>
<td>Address of 44-byte data set name.</td>
</tr>
<tr>
<td>8(8)</td>
<td>4</td>
<td>Reserved.</td>
</tr>
<tr>
<td>12(C)</td>
<td>4</td>
<td>Address of volume list.</td>
</tr>
</tbody>
</table>

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 Table 23.

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 z/OS DFSMSdfp Diagnosis 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 Table 25 on page 150 and the scratch status codes are shown in Table 24 on page 150. 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.)

Table 23 describes the conditions indicated by the SCRATCH return code.

Table 23. SCRATCH Return Codes

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000 (X'00')</td>
<td>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 z/OS DFSMSdfp Diagnosis to determine the failure-related conditions.</td>
</tr>
<tr>
<td>004 (X'04')</td>
<td>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.</td>
</tr>
<tr>
<td>008 (X'08')</td>
<td>An unusual condition was encountered on one or more volumes.</td>
</tr>
<tr>
<td>012 (X'0C')</td>
<td>One of the following conditions occurred:</td>
</tr>
<tr>
<td></td>
<td>• The SCRATCH parameter list is not valid.</td>
</tr>
<tr>
<td></td>
<td>• The volume list is not valid.</td>
</tr>
<tr>
<td></td>
<td>• 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.</td>
</tr>
<tr>
<td></td>
<td>The SCRATCH status code will not have been set.</td>
</tr>
</tbody>
</table>
Managing the VTOC

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:

Table 24. SCRATCH Status Codes

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (X'00')</td>
<td>The data set has been deleted from this volume.</td>
</tr>
<tr>
<td>1 (X'01')</td>
<td>The VTOC of this volume does not contain the format-1 or format-8 DSCB to be deleted.</td>
</tr>
</tbody>
</table>
| 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 or format-8 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. |
| 8 (X'08')  | The format-1 or format-8 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:

Table 25. Secondary Status Codes

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (X'00')</td>
<td>No secondary status for this volume.</td>
</tr>
<tr>
<td>128 (X'80')</td>
<td>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.</td>
</tr>
</tbody>
</table>

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 or format-8 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. 31-bit programs must pass a clean UCB addr on RENAME, when applicable.

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
Multivolume Considerations
To rename a data set that is stored on more than one volume, all volumes must be mounted.

Unrenamable Data Sets and UNIX Files
You cannot use the RENAME macro with either a SYSIN or SYSOUT data set or UNIX file (such as an z/OS UNIX file). You will receive unpredictable results if you use RENAME for UNIX files.

You cannot rename VIO data sets.
Data Set Security
You can rename a RACF-defined data set only if you have ALTER access authority to the data set in the DATASET class.

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 247.

Renaming a Data Set That Might be in Use
You can rename a data set that is allocated to the current address space but it cannot be open.

In general, you cannot rename a data set whose name is the same as any data set that is allocated to another address space in the same system or in the scope of the SYSDSN enqueue. The system bypasses this restriction if all of the following are true:

- Your program sets on a certain bit in the CAMLST macro expansion. You can code this instruction: OI listname+2,X’10’.
- You have at least read authority to the RACF facility class named STGADMIN.DPDSRN.olddsname, where olddsname is up to 23 characters of the existing data set name. You can use a generic class name such as STGADMIN.DPDSRN.SYS2.*. IBM recommends that no one have authority to STGADMIN.DPDSRN.* because it is too broad.
- The data set is not SMS-managed.

Alternatively, you can use the data set rename option of PDF. If you attempt to rename a non-SMS-managed, non-VSAM data set, the data set name is in use and you have the appropriate RACF facility class authority, then PDF asks whether you wish to proceed because you know that the data set is not actually open. Let the rename proceed only if you know the data set being renamed is not open on any system.

Attention: This option should be used with extreme caution. Very few people should have RACF authority to STGADMIN.DPDSRN.olddsname. Do not use this option unless you know the data set is not open on any system. After the data set is renamed, someone could delete it in a different address space. If someone has it open by the old name, new data sets will appear at those places on the disk. This would be a security violation that the system does not detect.

The data set rename function writes a type 18 SMF record to provide information to storage administrators, system programmers, and auditors. The record contains an indicator of whether it was successful due to the use of this duplicate name override function. If you request the option in the CAMLST macro expansion but the data set name is not in use, then the SMF indicator will not be on.
RENAME and CAMLST Macro Specification

The format of the RENAME and CAMLST macros is:

```
RENAME—listname_addrx
```

```
rename—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
```

**Recommendation:** 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.

**RENAME Parameter List**

The CAMLST macro generates the RENAME parameter list but your code can set several options that CAMLST does not support. See Table 26.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length or Bit Pattern</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0(0)</td>
<td>1</td>
<td>Flags. Always X'C1'</td>
</tr>
<tr>
<td>1(1)</td>
<td>1</td>
<td>Flags.</td>
</tr>
<tr>
<td>....</td>
<td></td>
<td>Do not define RACF profile.</td>
</tr>
<tr>
<td>....</td>
<td></td>
<td>Do not update the changed bit (DS1DSCHA) in the format 1 DSCB.</td>
</tr>
<tr>
<td>x.x xxxx</td>
<td>Reserved.</td>
<td></td>
</tr>
<tr>
<td>2(2)</td>
<td>1</td>
<td>Flags.</td>
</tr>
<tr>
<td>....</td>
<td></td>
<td>Always set for RENAME.</td>
</tr>
<tr>
<td>....</td>
<td></td>
<td>See &quot;Renaming a Data Set That Might be in Use&quot; on page 152.</td>
</tr>
<tr>
<td>x.x xxxx</td>
<td>Reserved.</td>
<td></td>
</tr>
<tr>
<td>3(3)</td>
<td>1</td>
<td>Reserved.</td>
</tr>
<tr>
<td>4(4)</td>
<td>4</td>
<td>Address of 44-byte existing data set name.</td>
</tr>
<tr>
<td>8(8)</td>
<td>4</td>
<td>Address of 44-byte new data set name.</td>
</tr>
<tr>
<td>12(C)</td>
<td>4</td>
<td>Address of volume list.</td>
</tr>
</tbody>
</table>

**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 Table 27 on page 155.

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 z/OS DFSMSdfp Diagnosis for a description of this information.

Each volume's volume list entry contains the rename status code as shown in Table 28 on page 155. 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.

Table 27 on page 155 describes the conditions indicated by the DADSM return code.
Table 27. DADSM RENAME Return Codes

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000 (X'00')</td>
<td>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 z/OS DFSMSdfp Diagnosis to determine the failure-related conditions.</td>
</tr>
<tr>
<td>004 (X'04')</td>
<td>No volume containing any part of the data set was mounted. The data set can be a VIO data set but cannot be renamed.</td>
</tr>
<tr>
<td>008 (X'08')</td>
<td>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 z/OS DFSMSdfp Diagnosis to determine the failure-related conditions.</td>
</tr>
</tbody>
</table>
| 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 following conditions described in Table 28.

Table 28. RENAME Status Codes

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (X'00')</td>
<td>The format-1 DSCB for the data set has been renamed in the VTOC on this volume.</td>
</tr>
<tr>
<td>1 (X'01')</td>
<td>The VTOC of this volume does not contain the format-1 or format-8 DSCB of the data set to be renamed.</td>
</tr>
</tbody>
</table>
| 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. |
| 3 (X'03')  | A format-1 or format-8 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 or format-8 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. |
Managing the VTOC

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 (X'06')</td>
<td>The operator was unable to mount this volume.</td>
</tr>
<tr>
<td>7 (X'07')</td>
<td>The data set was not renamed, because it was currently open for processing.</td>
</tr>
<tr>
<td>8 (X'08')</td>
<td>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.</td>
</tr>
</tbody>
</table>
Chapter 3. Using Catalog Management Macros

This information covers catalog management macro instructions for compatibility purposes only. Catalog management macro instructions can be used to perform the following functions:

- Retrieve information from an integrated catalog facility catalog.
- Catalog, uncatalog, or re-catalog in an integrated catalog facility catalog the following types of data sets:
  - DASD data sets that are not VSAM or SMS-managed
  - Tape data sets.

Application Program Considerations

A catalog management request can be satisfied in an integrated catalog facility catalog. Consider the following restrictions and limitations in relation to your application programs:

- 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.
- Register 15 contains the return code. These return codes are explained in the sections below.

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. 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, the user catalog is searched. When the entry is found, no other catalog is searched.
3. The master catalog is searched.

See z/OS DFSMS Access Method Services Commands 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 following information:

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

Restriction: 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 need to retrieve data from more than 20 volumes, use IGGCSI (Catalog Search Interface). See (link to pertinent section in Man Cat).
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 data set would be considered partially-qualified names:

- LEVEL1.LEVEL2.LEVEL3
- LEVEL1.LEVEL2
- LEVEL1

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

See "z/OS DFSMSdfp Diagnosis" for descriptions of the contents of the volume control block and the other catalog data areas.

See "Return Codes from LOCATE" on page 162 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. For each volume, the list contains the volume serial number, device type, and file sequence number. Volumes are divided by whether they fall within the minimum unit count, or outside of it. Volumes within and without are in descending binary order by device type, except for reusable KSDS. For a reusable KSDS, the volume serial numbers are returned in the order: first index, first data, remaining index, remaining data.

A volume list 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.

A CAMLST LOCATE on a VSAM cluster returns the volumes for all components. The format of this volume list is as described previously. The output is translated into the format described in the preceding two paragraphs 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
```

---

158  z/OS V2R1.0 DFSMSdfp Advanced Services
listname_addr
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.

dname_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.

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 A 6-BYTE 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 162).

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 to identify a data set in a generation data group:
- 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.
**Rule:** 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 CAMLIST NAME macros is:

```
LOCATE list_addrx
```

```
CAMLST NAME listname,dsname_relexp,area_relexp
```

**list_addrx**

Points to the parameter list (labeled listname) set up by the CAMLIST 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.

**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.
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 162). See “Retrieving Information by Data Set Name (LOCATE and CAMLST NAME)” on page 158 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.

The format for the LOCATE and CAMLST NAME macros is:

```
LOCATE list_addrx

label
```

```
listname CAMLST NAME, dsname_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.

`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.
```

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" for a description of the LOCATE return codes.

Reading a Block by Relative Block Address (LOCATE and CAMLST BLOCK)
This format is no longer supported and will result in an error.

Return Codes from LOCATE
Control is returned to your program at the next executable instruction following the LOCATE macro instruction. Register 15 contains one of the following return codes. If register 15 is non-zero, then register 0 contains an ICF catalog return code, described under message IDC3009I in z/OS MVS System Messages, Vol 6 (GOS-IEA).

Table 29. LOCATE Return Codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (X'00')</td>
<td>Operation successful.</td>
</tr>
<tr>
<td>4 (X'04')</td>
<td>Either the required catalog does not exist or it cannot be opened.</td>
</tr>
<tr>
<td>8 (X'08')</td>
<td>The user is not authorized to perform this operation. Register 0 contains hexadecimal 38.</td>
</tr>
<tr>
<td>12 (X'0C')</td>
<td>An invalid low-level GDG name was found.</td>
</tr>
<tr>
<td>16 (X'10')</td>
<td>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.</td>
</tr>
<tr>
<td>20 (X'14')</td>
<td>An invalid name has been provided</td>
</tr>
</tbody>
</table>
Table 29. LOCATE Return Codes (continued)

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
</tr>
</thead>
</table>
| 24 (X'18') | One of the following happened:  
- A permanent I/O or unrecoverable error was encountered.  
- An error was found in a parameter list. R1 is set to X'08000000'.  
- There was a nonzero return code from ESTAE or GETMAIN. R1 is set to X'08000000'. |
| 38 (X'26') | DFSMShsm LOCATE preprocessor has experienced an error. |

**Note:** See z/OS MVS System Messages, Vol 7 (IEB-IEE) and z/OS MVS System Messages, Vol 8 (IEF-IGD) message IDC3009I, for documentation of integrated catalog facility catalog and VSAM catalog return codes.

## Using Non-VSAM Data Set Catalog Entries

You can catalog, uncatalog, and recatalog non-VSAM data sets 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, see the DEFINE and the DELETE commands in z/OS DFSMS Access Method Services Commands.

### Cataloging a Non-VSAM Data Set (CATALOG and CAMLST CAT)

The format of the CATALOG and CAMLST macros is:

```
CATALOG list_addrx

CAMLST name_relexp, vol_list_relexp

CAT or CATBX

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.

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.
```
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
    DSNNAME   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. DSNNAME,
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 167.

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_addr
-label)
(CAMLST
-listname
-UNCAT
-UCATDX
-name_relexp)
```

- **list_addr**
  - 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 environment, if the data set name is less than 44 characters, it must be padded with blanks until the 44-character length is reached.

**Example**

In the following example, the catalog entry for data set A.B.C is removed from a catalog.

```
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. DSNMAP, 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 167.

**Restriction:** 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 must 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
CAMLST RECAT
name_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 DFSMSdssm 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.

`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.
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. Register 15 contains one of the following return codes. If register 15 is non-zero, then register 0 contains an ICF catalog return code, described under message IDC3009I in the "z/OS MVS System Messages, Vol 6 (GOS-IEA)"

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (X'00')</td>
<td>Operation successful.</td>
</tr>
<tr>
<td>4 (X'04')</td>
<td>Either the required catalog does not exist or it is not open.</td>
</tr>
<tr>
<td>8 (X'08')</td>
<td>One of the following happened:</td>
</tr>
<tr>
<td></td>
<td>• The existing catalog structure is inconsistent with the operation</td>
</tr>
<tr>
<td></td>
<td>requested.</td>
</tr>
<tr>
<td></td>
<td>• The user is not authorized to perform the operation.</td>
</tr>
<tr>
<td>20 (X'14')</td>
<td>There is insufficient space in the catalog data set. If R1 =</td>
</tr>
<tr>
<td></td>
<td>X'08000000', then an invalid name has been provided</td>
</tr>
<tr>
<td>28 (X'1C')</td>
<td>One of the following happened:</td>
</tr>
<tr>
<td></td>
<td>• A permanent I/O or unrecoverable error was encountered.</td>
</tr>
<tr>
<td></td>
<td>• An error was found in a parameter list. R1 is set to X'08000000'.</td>
</tr>
<tr>
<td></td>
<td>• There was a nonzero return code from ESTAE or GETMAIN. R1 is set to</td>
</tr>
<tr>
<td></td>
<td>X'08000000'.</td>
</tr>
</tbody>
</table>
CATALOG Macros
Chapter 4. Executing Your Own Channel Programs

This information describes 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. IBM recommends using an access method such as VSAM in place of EXCP or EXCPVR.

The EXCP and EXCPVR macro instructions allow 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 z/OS UNIX files and file systems. They are not supported for user-written applications using EXCP. This information covers 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 information, 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. You also need to understand the information in the following publications:

- **HLASM Programmer’s Guide** contains information about coding programs in the assembler language.
- **z/Architecture Principles of Operation**, SA22-7832, describes channel command words (CCWs) and channel programs.
- **z/OS DFSMS Using Data Sets** contains the standard procedures for I/O processing under the operating system.
- **z/OS DFSMS 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 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.

Comparing EXCP and EXCPVR

EXCP and EXCPVR are two macros you can use to initiate channel program I/O operations, or as it is often put, execute a channel program. Both provide the same function - a device dependent way to perform I/O operations. However, there are a number of differences:

- In order to issue an EXCPVR request, your program must run in a protection key between zero and seven, run in supervisor state, or be APF authorized. An EXCP request, on the other hand, can be issued by programs running in any key, including user key (key 8) or problem state.
If you issue an EXCPVR request, your program is responsible for translating its own virtual channel program into a real channel program. This includes page fixing your channel program and I/O buffers either before issuing the EXCPVR request, or by using the page fix appendage and updating your channel program with real addresses, building indirect address lists when needed, and updating the address fields within your channel program with real addresses. This allows your program to improve the efficiency of I/O operations in a paging environment, but does add some complexity to your program.

If you issue an EXCP request, you supply a virtual channel program. In other words, the channel program contains the virtual addresses of storage areas that may be in pageable storage, and the system is responsible for translating your virtual channel program into a real channel program.

Note that EXCP requests issued in an APF-authorized program in a V=R address space (EXCP V=R requests, in other words) are not translated. A V=R address space is one defined with ADDRSPC=REAL in the JCL EXEC statement. Because the address space is V=R, any CCWs created by the user already have correct real data addresses. (Translation would only re-create the user’s channel program, so the CCWs are used directly.)

For information on using APF, see the z/OS MVS Programming: Authorized Assembler Services Guide.

Table 31. Summary of the differences between EXCP, EXCPVR, and EXCP V=R

<table>
<thead>
<tr>
<th>Function</th>
<th>EXCP Characteristics</th>
<th>EXCPVR Characteristics</th>
<th>EXCP V=R Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program state and key</td>
<td>Any protection key, supervisor or problem state.</td>
<td>Protection keys 0-7, supervisor state, or APF authorized.</td>
<td>Protection keys 0-7, supervisor state, or APF authorized.</td>
</tr>
<tr>
<td>User responsible for translating their channel program</td>
<td>No</td>
<td>Yes</td>
<td>No translation required.</td>
</tr>
<tr>
<td>CCW formats supported</td>
<td>0 and 1</td>
<td>0 and 1</td>
<td>0 and 1</td>
</tr>
<tr>
<td>Virtual IDAW supported</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>MIDAWs supported</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>High Performance FICON® for System z (zHPF) channel programs supported</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>User can modify channel program during execution</td>
<td>No</td>
<td>Yes, non-zHPF only</td>
<td>Yes</td>
</tr>
<tr>
<td>Self modifying channel programs supported</td>
<td>No</td>
<td>Yes, non-zHPF only</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Using EXCP and EXCPVR

This information briefly explains the procedures required when issuing the EXCP and EXCPVR macro instruction. To issue the EXCP or EXCPVR macro instruction directly, perform the following tasks.

1. Allocate the data set or device to be used for the EXCP request. See “Allocating the Data Set or Device” on page 171.
2. Construct and open a data control block (DCB) with the DCB and OPEN macro instructions. Optionally create a data control block extension (DCBE) before issuing the OPEN macro. See “Initializing a Data Control Block (OPEN)” on page 240 and “Opening the Data Set” on page 172.
3. Create a channel program containing the commands necessary to perform the I/O operations on the appropriate device. See “Creating the Channel Program” on page 173.

4. Create the control blocks needed to initiate the EXCP request. This includes the input/output block (IOB) and event control block (ECB), and optionally the input/output block extension (IOBE), and input/output error data block (IEDB). See “Creating the EXCP-Related Control Blocks” on page 184. For more information on specific control blocks, see “Control Block Fields” on page 196.

5. Issue an EXCP or EXCPVR macro instruction to pass the address of the IOB, and optionally the IOBE, to the routines that initiate and supervise I/O operations.

After issuing EXCP or EXCPVR, issue a WAIT or EVENTS macro instruction specifying the address of the ECB, to wait for the channel program to complete. See “Executing the Channel Program” on page 185.

6. Once the I/O request has completed, examine the completion status of your EXCP or EXCPVR request and process any error conditions that may have occurred. See “Processing the I/O Completion Status” on page 189.

7. If volume switching is necessary (because of a unit exception or end of DASD extent), issue the EOV macro. See “Handling End of Volume and End-Of-Data-Set Conditions” on page 194.

8. When data set processing is complete, close the data set to restore the DCB. See “Closing the Data Set” on page 195.

9. If your program called dynamic allocation, it can optionally call dynamic unallocation.

### Allocating the Data Set or Device

You allocate the data set or device in one of the following ways:

- Using job step allocation, see [z/OS MVS JCL Reference](#).
- Using dynamic allocation, see one of the following:
  - [z/OS MVS Programming: Authorized Assembler Services Guide](#)
  - [z/OS TSO/E Command Reference](#)

When your program calls dynamic allocation using SYC 99 or the DYNALLOC macro, you can use the XTIOT option, which is bit S99TIOEX. It causes creation of an XTIOT entry instead of an entry in the TIOT. The XTIOT resides above the 16 MB line. This option requires your program to be APF authorized, in supervisor state, or in a system key (0–7).

With dynamic allocation, you can use the NOCAPTURE option, which is bit S99ACUCB. If the actual UCB address is above the 16 MB line and you do not code LOC=ANY option on the DCBE macro, then, OPEN and EOV will capture it and EOV and CLOSE will uncapture it. *(Capturing means to create a 24-bit address for a UCB that has an actual address above the 16 MB line. This NOCAPTURE option also causes creation of an XTIOT instead of a TIOT entry, but it does not require your program to have authorization). If you code the LOC=ANY option on the DCBE, then OPEN will not capture the UCB and its address will be in a four-byte field in the DEB.*

With dynamic allocation, you can use the S99DSABA bit, which allows the DSAB control block to reside above the 16 MB line. If you turn S99DSABA on, then you must also turn on S99TIOEX and your program must be authorized.
The purpose of these three dynamic allocation options (XTIOT, NOCAPTURE, and LOC=ANY) is to reduce use of storage below 16MB. BSAM, BPAM, and QSAM also support these three options.

To learn whether the allocation has any of these three options, issue the DEVTYPE macro with the INFO=AMCAP option. On a system before z/OS V1R11, bits S99DSABA and S99TIOEX will be off. On a release V1R12 or later system, these options should only be used when NON_VSAM_XTIOT=YES is specified in the DEVSUPxx parmlib member. This is represented by the DFAXTBAM bit in the DFA as mapped by IHADFA.

Opening the Data Set

You must supply a DCB for each data set or device to open. The OPEN macro instruction finishes initializing 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 and DCBE
- Verifying or creating standard labels
- Positioning tape
- Loading your appendage routines.
- Capturing the UCB if the DCBE option "LOC" was set or defaulted to "BELOW", that is LOC=BELOW or not coded, or the "NON_VSAM_XTIOT" option of the DEVSUPxx member of PARMLIB was set or defaulted to "NO", that is NON_VSAM_XTIOT=NO or not coded. In other words, OPEN does not capture the UCB if LOC and NON_VSAM_XTIOT were specified as follows: LOC=ANY and NON_VSAM_XTIOT=YES.

The parameters and different forms of the OPEN macro instruction are described in "OPEN - Initialize Data Control Block for Processing the JFCB" on page 302 and z/OS DFSMS Macro Instructions for Data Sets.

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. Your program can do this by reading in the JFCB with a RDJFCB macro instruction and opening each volume of the data set with a separate DCB. See "Using BSAM or EXCP for Random I/O to a Multivolume Data Set" on page 292 for an example of how to code a routine to do this, and "Reading and Modifying a Job File Control Block (RDJFCB Macro)" on page 284 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 perform the following tasks:

- Verify that an application has master password or RACF alter authority for the data set.
- 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.

When opening a VSAM component on a volume that supports extended attribute DSCBs, the specified DCB must point to a DCBE with the EADSCB=OK keyword. When EADSCB=OK is specified, your program must support extended attribute DSCBs. These are format-8 and format-9 DSCBs, where the extent descriptors may contain 28-bit cylinder numbers.

---

**Creating the Channel Program**

The channel program contains the instructions used by the channel subsystem and the device to execute an I/O operation. This section describes what you need to consider when you create a channel program for an EXCP request.

There are two types of channel programs, which are described in the following sections:

- "CCW Channel Program" on page 175
- "zHPF Channel Program" on page 175

**CCW Channel Program**

The CCW 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 (IOB) Fields" on page 211 for additional information.) If you specify both data and command chaining in the same channel command word, data chaining takes precedence.

The location of the CCWs, IDALs, MIDALs and I/O buffers in virtual and central storage depend on whether you are using EXCP or EXCPVR, whether you are using format 0 or 1 CCWs, and whether the device supports 64-bit IDALs or MIDALs. In particular:

- When you use format-0 CCWs, the CCWs and the storage areas pointed to by the CCWs (IDALs, MIDALs, or I/O buffers) must be in 24-bit storage. Otherwise, you can use 31-bit storage.
- When the CCW points to an indirect address list (IDAL), each indirect address list word (IDAW) in the list points to a 31-bit or 64-bit I/O buffer, depending on whether 31-bit or 64-bit IDAWs are used and whether the device supports 64-bit IDAWs. 64-bit IDAWs are only supported for disk and tape devices.
• When the CCW points to a modified indirect address list (MIDAL), each modified indirect address word (MIDAW) in the list points to a 64-bit I/O buffer. MIDAWs are only supported for EXCPVR requests for disk devices and only when running on an IBM System z9® or higher processor.

• For EXCP requests, the system translates your virtual channel program into a real channel program. During the translation process, the CCWs are copied to fixed storage, IDALs are created, and the I/O buffers are page fixed. Therefore, the storage restrictions mentioned above apply only to the virtual storage locations of the CCWs and IDALs; the CCWs and IDALs may reside anywhere in central storage. However, if the I/O buffer resides in 64-bit central storage, the device must support 64-bit IDAWs or else the system fails the EXCP request.

• Because the system does translate your channel program for an EXCPVR request, the storage restrictions mentioned above apply to both virtual and central storage. For example, if format-0 CCWs are used, the CCWs and the storage areas pointed to by the CCWs (IDALs, MIDALs, or I/O buffers) must be in 24-bit virtual and central storage. However, the IDALs and MIDALs can still point above the 16MB line.

The following table summarizes the channel program storage requirements for different types of CCW channel programs:

<table>
<thead>
<tr>
<th>Request and format</th>
<th>Channel Program Component</th>
<th>Virtual storage location</th>
<th>Central storage location</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXCP format 0</td>
<td>CCW</td>
<td>24-bit</td>
<td>Any</td>
</tr>
<tr>
<td></td>
<td>IDAL</td>
<td>24-bit</td>
<td>Any</td>
</tr>
<tr>
<td></td>
<td>I/O Buffer</td>
<td>• 24-bit if pointed to by CCW&lt;br&gt;• 31 or 64-bit if pointed to by an IDAW</td>
<td>Any</td>
</tr>
<tr>
<td>EXCP format 1</td>
<td>CCW</td>
<td>31-bit</td>
<td>Any</td>
</tr>
<tr>
<td></td>
<td>IDAL</td>
<td>31-bit</td>
<td>Any</td>
</tr>
<tr>
<td></td>
<td>I/O Buffer</td>
<td>• 31-bit if pointed to by CCW&lt;br&gt;• 31 or 64-bit if pointed to by an IDAW or MIDAW</td>
<td>Any</td>
</tr>
<tr>
<td>EXCPVR format 0</td>
<td>CCW</td>
<td>24-bit</td>
<td>24-bit</td>
</tr>
<tr>
<td></td>
<td>IDAL</td>
<td>24-bit</td>
<td>24-bit</td>
</tr>
<tr>
<td></td>
<td>MIDAL</td>
<td>24-bit</td>
<td>24-bit</td>
</tr>
<tr>
<td></td>
<td>I/O Buffer</td>
<td>• 24-bit if pointed to by CCW&lt;br&gt;• 31 or 64-bit if pointed to by an IDAW or MIDAW</td>
<td>• 24-bit if pointed to by CCW&lt;br&gt;• 31 or 64-bit if pointed to by an IDAW or MIDAW</td>
</tr>
<tr>
<td>EXCPVR format 1</td>
<td>CCW</td>
<td>31-bit</td>
<td>31-bit</td>
</tr>
<tr>
<td></td>
<td>IDAL</td>
<td>31-bit</td>
<td>31-bit</td>
</tr>
<tr>
<td></td>
<td>MIDAL</td>
<td>31-bit</td>
<td>31-bit</td>
</tr>
<tr>
<td></td>
<td>I/O Buffer</td>
<td>• 31-bit if pointed to by CCW&lt;br&gt;• 31 or 64-bit if pointed to by an IDAW or MIDAW</td>
<td>• 31-bit if pointed to by CCW&lt;br&gt;• 31 or 64-bit if pointed to by an IDAW or MIDAW</td>
</tr>
</tbody>
</table>
**zHPF Channel Program**

zHPF channel programs are supported for EXCP and EXCPVR requests for DASD devices. The following diagram shows the different parts of a zHPF channel program.

The parts of a zHPF channel program include:

- A transport control word (TCW) that contains pointers to all of the other areas of the channel program and the number of bytes to be read or written. The channel uses the TCW to transport the commands and data to the device and locate the status block used to store ending status information; it is not sent to the device. You can use the IOSDTCW macro to map the TCW.

  For an EXCPVR request, the TCW must reside in 24 or 31-bit central and virtual storage.
  For EXCP, the TCW must be in 24 or 31-bit virtual storage or anywhere in central storage.
  For both EXCP or EXCPVR, the TCW must start on a 64-byte boundary. If the TCW does not start on a 64-byte boundary, EXCP fails the request with abend code 800, reason code X’OA’.

- A transport status block (TSB) that contains I/O completion information, sense data and measurement statistics. The TSB is assigned by z/OS, not the EXCP user. If you provide a TCW with a non-zero TSB address, EXCP fails the request with abend code 800. The system copies all relevant status information from the TSB into the IOBE.

- A transport command control block (TCCB) containing the commands and control data parameters to be passed to the device. The TCCB should start on a double word boundary, must reside within a 4 KB page, and may reside in 64-bit virtual and central storage. You can use the IOSDTCCB macro to map the TCCB, including device command words (DCWs). The TCCB consists of three parts:
  - Transport control area header (TCAH) containing information about the transport control area (TCA) and the commands contained within.
  - Transport control area (TCA) containing the commands and control parameters. Each command is represented by a DCW that consists of a
command code, flags to indicate chaining and other options, a control data count, and a data byte count, if the command is used to transfer data. If the command transfers control data (command parameters) to the device, the control data follows the DCW in the TCA.

Unlike CCWs, DCWs do not point to their corresponding I/O buffers. The I/O buffers for all DCWs are pointed to by the TCW, and the I/O buffers associated with a particular DCW are based on the amount of data transferred by the previous DCWs.

The maximum size of the TCA is 240 bytes.

- Transport control area trailer (TCAT) containing the number of bytes transferred.

The System z I/O architecture allows the TCCB to be pointed to either directly by the TCW or indirectly via a transport indirect address list (TIDAL). However, TCCB TIDALs are not supported for EXCP or EXCPVR requests. If a TCCB TIDAL is specified, EXCP fails the request with abend code 800.

- One or more I/O buffers. The TCW may point to a single read and/or write buffer.

For EXCPVR requests, the I/O buffer can be up to 4 KB in size. If more than 4 KB of data needs to be transferred or the data is non-contiguous or spans a page, a data TIDAL must be created. The I/O buffers and the TIDAWs may reside in 64-bit virtual and central storage. A TIDAL can also be chained to another TIDAL by setting the TIDAL transfer-in-channel (TTIC) bit in the last TIDAW in the list. In this case, the TIDAW address field does not point to an I/O buffer, but instead points to another TIDAL.

For EXCP requests, the I/O buffers can be greater than 4KB - see "TIDAW requirements for EXCP requests" on page 181.

The following table summarizes the channel program storage requirements for zHPF channel programs:

<table>
<thead>
<tr>
<th>Request and format</th>
<th>Channel Program Component</th>
<th>Virtual storage location</th>
<th>Central storage location</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXCP</td>
<td>TCW</td>
<td>31-bit</td>
<td>Any</td>
</tr>
<tr>
<td></td>
<td>TCCB</td>
<td>Any</td>
<td>Any</td>
</tr>
<tr>
<td></td>
<td>TIDAL</td>
<td>Any</td>
<td>Any</td>
</tr>
<tr>
<td></td>
<td>I/O Buffer</td>
<td>Any</td>
<td>Any</td>
</tr>
<tr>
<td>EXCPVR</td>
<td>TCW</td>
<td>31-bit</td>
<td>31-bit</td>
</tr>
<tr>
<td></td>
<td>TCCB</td>
<td>Any</td>
<td>Any</td>
</tr>
<tr>
<td></td>
<td>TIDAL</td>
<td>Any</td>
<td>Any</td>
</tr>
<tr>
<td></td>
<td>I/O Buffer</td>
<td>Any</td>
<td>Any</td>
</tr>
</tbody>
</table>

**Comparing CCW and zHPF channel programs**

<table>
<thead>
<tr>
<th>Function</th>
<th>CCW channel programs</th>
<th>zHPF channel programs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Devices supported</td>
<td>All</td>
<td>DASD only</td>
</tr>
</tbody>
</table>
Table 34. Comparing CCW and zHPF channel programs (continued)

<table>
<thead>
<tr>
<th>Function</th>
<th>CCW channel programs</th>
<th>zHPF channel programs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processors supported</td>
<td>All</td>
<td>EXCP and EXCPVR are only supported on a zEnterprise® 196 (z196) zEnterprise 114 (z114) or higher processor</td>
</tr>
<tr>
<td>Command set</td>
<td>All commands for all currently supported devices</td>
<td>Subset of DASD commands</td>
</tr>
<tr>
<td>Maximum channel program size</td>
<td>No limit</td>
<td>240 bytes because of the TCA size limit</td>
</tr>
<tr>
<td>Maximum data transferred per CCW or TCW</td>
<td>64 KB-1</td>
<td>64 KB, 2 MB, or 4 GB, depending on the processor</td>
</tr>
<tr>
<td>Read and write commands in the same channel program</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Skip on read</td>
<td>CCW or MIDAW skip bit</td>
<td>TIDAW skip bit</td>
</tr>
<tr>
<td>Indirect addressing</td>
<td>IDAL, MIDAL</td>
<td>TIDAL</td>
</tr>
<tr>
<td>Data Chaining</td>
<td>Yes</td>
<td>No, but TIDAL is equivalent to data chaining.</td>
</tr>
<tr>
<td>Status modifier/search commands</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Program controlled interrupt (PCI)</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Append to running channel program</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Self-modifying channel programs</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>VIO data sets</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

EXCP 64-bit Storage Considerations

For EXCP virtual requests, the system page fixes the I/O buffers associated with your channel program. These I/O buffers may reside in 31-bit or 64-bit virtual storage.

If the I/O buffers reside in 64-bit virtual storage (referred to as a memory object), you must follow the requirements below:

- You must specify the CONTROL=AUTH parameter on the IARV64 macro request that creates the memory object. If you specify or default to CONTROL=UNAUTH on the IARV64 macro, EXCP fails when it attempts to page fix the storage.
- The I/O buffers must not reside within a memory object backed by 1 MB fixed real page frames. For example, you cannot specify PAGEFRAMESIZE=1MEG or PAGEFRAMESIZE=MAX on the IARV64 GETSTOR request that creates the memory object. If you do, EXCP fails when it attempts to page fix the storage.

IDAW Requirements for EXCP Requests

For EXCP requests, the virtual channel program provided by the caller can have one or more CCWs with the indirect address (IDA) flag set and the address portion of these CCWs pointing to a single 31-bit or 64-bit IDAW. This EXCP function is referred to as virtual IDAWs.
The 31-bit 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
- Read, read backward, and sense commands, with the skip flag set.
- All nondata-transfer type commands: for example, recalibrate, rewind, forward space, fold, block data check, no operation, control commands

A virtual IDAW points to a single, contiguous 31-bit or 64-bit virtual storage area. The virtual storage area can be backed by central storage above or below 2 GB. Either type of virtual IDAW can point to storage that is backed above 2 GB. When EXCP translates the channel program, the virtual IDAW is translated into one or more real IDAWs in 31-bit central storage.

**Note:** 64-bit IDAWs are supported only for direct access storage devices (DASD) and on all IBM-supplied cartridge tape devices. You should examine bit UCBEIDAW in the UCB to determine whether 64-bit IDAWs are supported by the device.

### IDAW Requirements for EXCPVR Requests

For EXCPVR requests, the caller is responsible for creating IDAWs when the data areas associated with the CCWs in your channel program cross certain boundaries. This can be done before issuing the **EXCPVR macro** or by the SIO appendage after the EXCPVR macro is issued. See the [“SIO Appendage”](#) on page 227 for more information.

For format-0 channel programs, the CCWs and IDAWs must be below 16 MB in central storage. For format-1 channel programs, the CCWs and IDAWs must be below 2 GB in central storage. Regardless of the channel program format, you can use 31-bit or 64-bit IDAWs to point to storage areas above or below 2 GB in central storage.

**Note:** 64-bit IDAWs are supported only for direct access storage devices (DASD) and on all IBM-supplied cartridge tape devices. You should examine bit UCBEIDAW in the UCB to determine whether 64-bit IDAWs are supported by the device. If you use 64-bit IDAWs, an IOBE must be specified for the EXCPVR request and flag IOBEEIDA in the IOBE must be set.

If data areas do cross boundaries, provide an additional IDAW in the IDAL for each crossed boundary:

- If you use 31-bit IDAWs, you must use 2 KB boundaries.
- If you use 64-bit IDAWs, you must use 4 KB boundaries.

The channel subsystem uses the IDAL to identify the address where it will continue reading or writing when a boundary is crossed during a read or write operation. The IDAL must contain central storage addresses when it is processed by the channel subsystem.

Before you convert the virtual addresses in the channel program to real addresses, you must first page fix the data areas in central storage. This can be done before issuing the EXCPVR macro or by the page fix appendage after the EXCPVR macro is issued. After the data areas have been page fixed, you can use the LRA instruction to convert the virtual addresses in the channel program to central storage addresses, as long as the central storage addresses are below 2 GB. The LRA instruction returns a 31-bit central storage address regardless of whether you are in 24-bit or 31-bit addressing mode, but fails in those addressing modes if the
central storage address is above 2 GB. If the central storage address is above 2 GB, you must either use the LRAG or STRAG instruction to convert the virtual address to a real address or else use the LRA instruction after first switching to 64-bit addressing mode.

If all of the central storage addresses associated with the data buffers are below 2 GB, then you can use 2 KB (31-bit) IDAWs to address the data. Otherwise, you must use 4 KB (64-bit) IDAWs to address the data. For the most efficient use of system resources, code LOC=(ANY,64) or LOC=(BELOW,64) when you obtain storage with the GETMAIN or STORAGE macro. See z/OS MVS Programming: Assembler Services Reference ABE-HSP.

The following illustration shows the relationship between the CCW and the IDAWs.

Note the following information about the IDAL:

- If you are using 31-bit IDAWs, then after the first entry, put one entry in the IDAL for each 2 KB page boundary that your data area crosses.
- If you are using 64-bit IDAWs, put one entry in the IDAL for each 4 KB page boundary that your data area crosses and set flag IOBEEIDA in the IOBE.
- If the format 0 CCW has an IDAL address rather than a data address, you must set the indirect address flag (bit 37) on to signal this to the channel. The equivalent format 1 CCW bit is bit 13.
- You can determine the maximum number of entries needed in the IDAL from the count in the CCW as follows:

  \[
  \text{Number of IDAWSs} = \frac{(\text{CCW-byte-count} + P + P - 2)}{P}
  \]
- Substitute 2048 for P in this formula for 31-bit IDAWs.
- Substitute 4096 for P in this formula for 64-bit IDAWs.

For example, the following shows the maximum number of IDAWs for a particular CCW byte-count and a page boundary of 4 KB:

<table>
<thead>
<tr>
<th>CCW byte-count</th>
<th>Maximum number of IDAWs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4095</td>
<td>2</td>
</tr>
<tr>
<td>4096</td>
<td>2</td>
</tr>
<tr>
<td>4097</td>
<td>2</td>
</tr>
<tr>
<td>4098</td>
<td>3</td>
</tr>
</tbody>
</table>

The number of 31-bit IDAWs that are required depends on the number of 2 KB boundaries that are crossed by the data. For example, if your data is 800 bytes long and does not cross a 2 KB boundary, no IDAWs are required. If your data crosses a 4 KB boundary, then two IDAWs are required. If your data is 5000 KB long, at least two IDAWs are required. If your data crosses two 4 KB boundaries, four IDAWs 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 2 KB or 4 KB 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 4-byte IDAL contains 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 contains the following addresses:

- 707FF
- 70800

**MIDAW Requirements**

For EXCPVR requests, the channel program provided by the caller can have one or more CCWs with the modified indirect address (MIDA) flag set. When this flag is set, the CCW points to a list of one or more modified indirect address words (MIDAWs). MIDAWs are similar to IDAWs except that the boundary and length requirements are relaxed. For example, each IDAW in a list, except the first, must point to a storage area on either a 2 KB or 4 KB boundary depending on the type of IDAW, and the length includes storage up to the next 2 KB or 4 KB boundary or until the CCW count is exhausted. On the other hand, MIDAWs can point anywhere on a page and contain any length as long as a page boundary is not crossed. MIDAWs can reduce the number of CCWs in your channel program by eliminating the need for data chaining, thereby improving the performance of your channel program.
MIDAWs are supported only for direct access storage devices (DASD) and only when running on an IBM System z9 or higher processor. Examine bit UCBMIDAW in the UCB to determine whether MIDAWs are supported by the device.

If you use MIDAWs, you must specify an IOBE for the EXCPVR request and set flag IOBEMIDA in the IOBE.

**TIDAW requirements for EXCP requests**

The input and output data pointers in the TCW may point to a single, contiguous area of storage, or may point to a virtual TIDAL that consists of one or more virtual TIDAWs, each of which may points to a contiguous area of storage. Unlike the zHPF channel programs created by EXCPVR, these storage areas may be larger than 4K and may span page boundaries.

During channel program translation EXCP creates a real TIDAL when any of the following is true:

- The TCW input or output data pointer points to a virtual storage area that spans pages. For example, the TCW input address points to a 16K byte virtual storage area or points to a 2K byte area that crosses a page boundary.
- The TCW points to a virtual TIDAL. Unlike CCW channel programs, the TIDAL may consist of multiple TIDAWs, each of which can point to a virtual storage area that spans pages. Supporting multiple TIDAWs is necessary for zHPF because there is only a single input and output area pointer (rather than one per CCW), so the storage areas are more likely to be scattered throughout virtual storage.

**An EXCP Request with a Single 16K Storage Area**

In Figure 11, the TCW points to a single 16K area of storage. When EXCP translates the channel program, it creates four TIDAWs, one for each page.

![TCW to TIDAW translation](image)

*Figure 11. How EXCP translates an EXCP request with a single 16K storage area*

Note that if the TCW points to an area that is less than or equal to 4K but spanned pages, EXCP will create two TIDAWs.

**An EXCP Request with a virtual TIDAL**

In Figure 12 on page 182, the TCW points to the virtual address of a TIDAL. The TIDAWs within the TIDAL point either to storage areas that larger than 4K or to storage areas that cross page boundaries. The TIDAL itself also crosses a page boundary. Note that for EXCPVR requests, this would not be allowed because it violates I/O architecture rules. However, for EXCP requests, the virtual TIDAL may have any alignment and may cross page boundaries - EXCP will translate the
virtual TIDAL to a real TIDAL and ensure proper alignment and ensure that page boundaries are not crossed.

**Determining Whether zHPF is Supported for a Device**

Before you create and use a zHPF channel program, you must make sure that zHPF and the zHPF incorrect length facility are supported for both the device and the processor involved. An EXCP or EXCPVR request is only supported on z196 or z114 processors that support the zHPF incorrect length facility. To determine whether the processor supports zHPF, the zHPF incorrect length facility and EXCP/EXCPVR, issue the IOSZHPF macro with the DEVINFO=YES parameter.

DEVINFO returns 8 bytes of information about zHPF functions supported for the device, including:

- Processor and z/OS capabilities
  - Functions supported by the channel subsystem and online channels for a device
  - Maximum data transfer size
  - EXCPVR/EXCP virtual supported
  - Incorrect length support

- Device capabilities - You must specify DEVINFO=YES to get device related information. Refer to the device specific architecture to interpret the device related DEVINFO=YES information.

- For those capabilities that have both a processor and device component, you must check both bits IOSDZHPF and IECDZHPF to determine if the capability is supported.
As Figure 13 shows, you must check the following in order to check to see whether the processor and device supports zHPF:

- If the processor supports zHPF
- If the processor supports incorrect length
- If the device supports incorrect length

### Modifying a Channel Program During Execution

For EXCP requests that run in a V=V 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, which run before the channel program is translated, 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 affect 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.

Modifying a channel program during execution is supported for EXCPVR and EXCP requests that run in a V=R address space, since the system does not build a translated copy of your channel program. Modifying a channel program during execution is only supported for CCW channel programs, not zHPF channel programs.

### VIO Considerations

VIO data sets are supported for EXCP and EXCPVR requests. When an EXCP or EXCPVR request is issued for a VIO data set, the system simulates all of the common channel commands. Both format-0 and format-1 CCW channel programs are supported for VIO data sets, but note that 64-bit IDAWs, MIDAWs, and zHPF channel programs are not supported for a VIO data set.
For EXCPVR requests, the addresses in the CCWs and IDAWs must be virtual addresses - they must not be converted to real addresses.

Creating the EXCP-Related Control Blocks

Using EXCP requires familiarity with the function and structure of the IOB, ECB, DCB, DEB, and optionally IOBE, DCBE, UCB and IEDB. DCB, IOB, IOBE, IEDB, and ECB fields are illustrated in the “Control Block Fields” on page 196 section. The DEB fields used for EXCP and EXCPVR are illustrated in Appendix A, “Control Blocks,” on page 443 (all the DEB fields are illustrated in z/OS DFSMSdfp Diagnosis).

The IOB, ECB, DCB, and DEB must be in 24-bit virtual storage. The IOBE, IEDB, and DCBE may reside in 31-bit virtual storage, even if your program runs in 24-bit mode.

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 (IOB)

The input/output block (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. See “Input/Output Block (IOB) Fields” on page 211.

Input/Output Block Common Extension (IOBE)

The input/output block common extension (IOBE) specifies the type of channel program and its format, options that control the execution of the channel program and 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 required for format-1 CCW channel programs and zHPF channel programs, but is optional for format-0 CCW channel programs. See “Input/Output Block Common Extension (IOBE) Fields” on page 216.

Event Control Block (ECB)

The event control block (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. See “Event Control Block (ECB) Fields” on page 220.

Input/Output Error Data Block (IEDB)

The system uses the input/output error data block IEDB to provide extended error information. This control block is optional. See “Input/Output Error Data Block (IEDB) Fields” on page 209.

Data Control Block (DCB)

The data control block (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. See “Data Control Block (DCB) Fields” on page 196.

**Data Control Block Extension (DCBE)**

The data control block extension (DCBE) provides further processing options. The DCBE options currently supported by EXCP are the following:

- BLKSIZE
- BLOCKTOKENSIZE
- CAPACITYMODE
- EADSCB
- EODAD
- LOC
- SYNC

See “Data Control Block Extension (DCBE) Fields” on page 207.

**Data Extent Block (DEB)**

The data extent block (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. See “Data Extent Block (DEB) Fields” on page 222.

**Executing the Channel Program**

This information explains how to pass the channel program to the system for execution and how the system uses your channel program and control blocks after you issue EXCP or EXCPVR request.

**Using the EXCP macro instruction**

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:

```
/SM590000/SM590000
EXCP iob_addr label /SM590000/SM630000
```

*iob_addr—RX-type address, (2-12), or (1)*

The address of the IOB of the channel program to be executed.

If your program is also supplying an IOBE, then set register 0 to the address of the IOBE, and set flag IOBCEF on in the IOB to indicate that the IOB extension is present. An IOBE is required for format-1 CCW channel programs.
Using the EXCPVR macro instruction

The EXCPVR macro instruction provides you with the same functions as the EXCP macro instruction and 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 eight, executing in supervisor state, or be APF authorized.

The program issuing the EXCPVR must remain in authorized state until completion of the channel programs. For a description of how to authorize a program, see z/OS MVS Programming: Authorized Assembler Services Guide.

The format of the EXCPVR macro is:

```
EXCPVR.iob_addr
```

*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 to the address of the IOBE, and set flag IOBCEF on in the IOB to indicate that the IOB extension is present. An IOBE is required for format-1 CCW and zHPF channel programs.

To use EXCPVR, follow the procedures needed to execute an EXCP request and also follow the procedures listed below. If you have already page fixed your channel program and data areas prior to issuing the EXCPVR macro, or you are 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.
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.
3. If you have not already built indirect address lists or if your channel program still contains virtual addresses prior to issuing the EXCPVR macro, you must do the following two items in your start I/O (SIO) appendage. The SIO appendage is described in "Start-I/O Appendage" on page 225.
   • Determine whether the data areas in virtual storage specified in the address fields of your channel program cross page boundaries. If they do, build an indirect data address list and update your channel program with the address of the indirect address list. For CCW channel programs, build an IDAL or MIDAL.
     If you build an IDAL, put the address of the IDAL in the affected CCW and turn on the IDA bit in the CCW.
     If you build a MIDAL, put the address of the MIDAL in the affected CCW and turn on the MIDA bit in the CCW. For zHPF channel programs, build a
TIDAL, put the address of the TIDAL in the input or output address field in the TCW, and turn on the input or output TIDAL bit in the TCW.

- Translate the addresses in your channel programs from virtual to central storage addresses.

**Initiating the Channel Program**

Issuing EXCP or EXCPVR macro requests execution of the channel program specified in the IOB. The system validates the request by checking fields of the control blocks associated with this request. If the system detects invalid information in a control block, it initiates abnormal termination procedures. The system gets the address of the:

- DCB from the IOB
- DEB from the DCB
- UCB from the DEB.

If this is an EXCPVR request and you have provided a page-fix appendage, the system passes control to it to allow you to either page fix your channel program and data areas, or to provide EXCP with a page fix list so that EXCP will do the page fixing. For a description of the page-fix appendage and its linkage to the system, see “Page Fix and EXCPVR Start I/O Appendage” on page 226.

If you have provided a start I/O (SIO) appendage, the system passes control to it. The system does not examine the channel program until the return from the SIO appendage. The return address from the SIO appendage determines whether the system executes or skips the I/O operation. For a description of the SIO appendage and its linkage to the system, see “Start-I/O Appendage” on page 225.

**Translating the Channel Program**

For EXCP requests that do not run in a V=R address space, the system performs the following tasks:

- 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.

**DASD Channel Program Prefix CCW Commands**

For direct access devices, specify the seek address in the IOB. The system constructs a CCW chain for a CKD device, The CCW chain issues the seek and set file mask specified in the DEB, and passes control to the real version of your channel program. You cannot issue the initial seeks or set the file mask CCWs. For an ECKD™ device, the system constructs a Define Extent or Prefix command instead of the seek and set file mask command. The system uses the contents of DEBXDEF, as described in “Appendix A, “Control Blocks,” on page 443, when building the Define Extent or Prefix command. 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. Your program can issue the DEVTYPE macro to learn whether the device supports ECKD. Use the minimum type of call or the INFOLIST type with INFO=DASD.

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.
The DEVTYPE macro with the INFO=DASD parameter will also tell your program whether the device supports the locate record extended command and it has controller cache. See "DEVTYPE—Info Form" on page 270.

DASD Rotational Positioning Sensing
On newer storage subsystems, such as IBM D58000, the sector value has no effect. On other subsystems, your channel program can be more efficient for device and channel usage if it supplies sector numbers. Your program can read sector numbers with the read sector command or calculate them with the sector conversion routine. That routine is described in "Obtaining the Sector Number of a Block on an RPS Device" on page 236.

Command Retry Considerations
Command retry is a function of many IBM 3990 and newer storage controllers. 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 EXCPVR and EXCP V=R requests, 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:

<table>
<thead>
<tr>
<th>CCW#</th>
<th>Channel Program</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Read, data chain</td>
</tr>
<tr>
<td>2</td>
<td>Read, data chain</td>
</tr>
<tr>
<td>3</td>
<td>Read, data chain, PCI</td>
</tr>
<tr>
<td>4</td>
<td>Read, command chain</td>
</tr>
</tbody>
</table>

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 z/OS DFSMS 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.

**Processing the I/O Completion Status**

For direct access and tape devices, the system considers the channel program completed when it has received both a channel-end and device-end. The channel-end and device end can be presented by the device separately or simultaneously. But for other devices, 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, the system places a completion code in the IOB and IEDB and then in the ECB (after appendages have been called). 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 device 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 CCWs 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 *z/Architecture 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 IOB, IEDB, and then in the ECB, after appendages have been called.

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

**JOBFLAGS in “Input/Output Block (IOB) Fields” on page 211**
and **Purging and Restoring I/O Requests (PURGE and RESTORE macros)” on page 304**

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 new 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. For example, DASD channel programs are almost always retrieved from
the beginning. You will want to build channel programs that the ERP can restart
after any CCW or DCW has failed.

For CCW channel programs, 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 the ERP (see **Modifying a Channel Program During
Execution” on page 183**):

- 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 OXL16 Area for Locate Record command
DC X'06010002' Operation, auxiliary, block count (2)
Seek DC X'xxxxxxxx' CCHH for seek
Search DC X'xxxxxxxxxxx' 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. (On an IBM 3990 storage subsystem, it will execute more efficiently if an appropriate value is stored at “Sector” also. On newer storage subsystems, such as the IBM 2107, the sector value has no effect.)

If all or part of the third CCW executes (Read Count), then reexecution of the channel program by the ERP will give different results.

**Requesting Extended Error Information**

You can request that EXCP and EXCPVR processing return extended error information. This extended error information consists of sense information, a completion code, and other information 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, you must perform the following steps:

- Follow the EXCP/EXCPVR initialization procedures described in “Input/Output Block (IOB) Fields” on page 211, 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 (IOBE) Fields” on page 216 and “Input/Output Error Data Block (IEDB) Fields” on page 209.
- 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 the EXCP macro instruction” on page 185 and “Using the EXCPVR macro instruction” on page 186 respectively).

There are two versions of the IEDB. The version number within the IEDB defines how long the IEDB is - the version 1 IEDB is 48 bytes long, and the version 2 IEDB is 96 bytes long.

- **The version 1 IEDB contains the following information:**
  - The completion code from I/O processing that gives the status. The completion codes and their meanings are shown in Figure 26 on page 220.
  
  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.

- The version 2 IEDB is recommended for zHPF channel programs, and contains all of the information in the version 1 IEDB plus the failing storage address when a channel control check or channel data check occurs as a result as a storage or storage key error error, or if a program check or protection check occurs for a zHPF channel program due to a bad storage address.

**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.

**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. VIO is not supported for zHPF channel programs.

**Invalid ending status**

Normally when a channel program ends, the ending CCW or TCW address is stored in field IOBCMD31 or IOBCMDA depending on the type of channel program. However, for certain types of errors, the ending status of the channel
program is unpredictable and the ending address will be set to zero. Therefore, your program should never assume that a valid ending address is always provided in the IOB.

For CCW channel programs, the ending CCW address will be set to zero when any of the following conditions occur:

- An interface control check, channel control check, or channel data check has occurred and the ending address in the subchannel status word (SCSW) is not valid
- A program check, protection check, or chaining check has occurred
- The ending CCW address was in a part of the channel program that was modified by device dependent system code. If device dependent system code needs to modify your channel program, it does so in system related storage that is not available to your program.

For zHPF channel programs:

- The ending TCW address will be set to zero when an interface control check, channel control check, or channel data check has occurred and the ending address in the SCSW is not valid.
- The ending DCW offset in the IOBE is more useful for determining where the channel program ended than the TCW, since the DCW is analogous to the CCW in zHPF channel programs. The ending DCW offset will not be stored if the device did not send ending status containing a valid DCW offset or the channel was unable to store the ending status. In this case, the flag indicating that the DCW offset is valid will be off.

**Device No Longer Supports zHPF or Required zHPF functions**

As mentioned in "Determining Whether zHPF is Supported for a Device" on page [182], you must make sure that a device supports zHPF as well as all the zHPF functions your channel program requires before issuing an EXCP or EXCPVR request for a zHPF channel program. In addition, even if the processor and device support zHPF and all the zHPF functions required, these might be disabled when you actually submit your channel programs:

- zHPF itself might be disabled on a processor and device that supports it when you submit your channel program for any of the following reasons:
  - The operator issued a command to disable zHPF for the system
  - The zHPF feature is no longer enabled for the control unit associated with the device
  - Internal errors have occurred that caused zHPF to be disabled for the device
  - The device used in the EXCP or EXCPVR request has been swapped to a device that does not support zHPF.

When the device no longer supports zHPF, the IOB completion code (IOECBCC) is set to X'41' and the reason code (IOBERCOD) is set to X'0E'. You can retry the EXCP or EXCPVR request by creating a CCW channel program and either reissuing the EXCP or EXCPVR request, or returning at offset +8 in your abnormal end appendage to execute a new channel program.

- Some required zHPF functions might be disabled on a supporting processor and device if you swap to a device that does not support all required zHPF functions. In that case, you must evaluate the zHPF functions that are supported on that device to determine whether you still can use a zHPF channel program or whether you must substitute a CCW channel program instead.
Handling End of Volume and End-Of-Data-Set Conditions

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.

To determine how the system disposes of a tape volume when a program issues an EOV macro, see the description of the DISP parameter of the OPEN macro in z/OS DFSMS Macro Instructions for Data Sets.

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:

Table 36. DCBOFLGS Usage

<table>
<thead>
<tr>
<th>Position</th>
<th>Bit name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>DCBOFLWR</td>
<td>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. For DASD, a 1 also indicates that ECBFDAD and DCBTRBAL contain valid information. See “Device-Dependent Parameters” on page 203 for more information.</td>
</tr>
<tr>
<td>1</td>
<td>DCBOFLRB</td>
<td>indicates that a backward read was the last I/O operation</td>
</tr>
<tr>
<td>4</td>
<td>DCBOFPPC</td>
<td>indicates that concatenated data sets are to be treated as unlike. For further information, refer to z/OS DFSMS Using Data Sets.</td>
</tr>
<tr>
<td>5</td>
<td>DCBOFTM</td>
<td>indicates that a tapemark has been read (tape only).</td>
</tr>
</tbody>
</table>

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 on both the old and new tapes if they are labeled. See z/OS DFSMS 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 in "Device-Dependent Parameters" on page 203 for more information.

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 without the nocapture option of dynamic allocation, then EOV might have uncaptured the UCB for the previous volume. If so, that captured UCB address 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
```

\textit{dcb_addr}\textsuperscript{–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.

\textbf{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:

\textbf{24} If you do not specify the \texttt{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.

\textbf{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 16 MB, because providing a DCB above 16 MB causes an ABEND50D. The high-order byte of the address specified in the EOV macro must be zero.

---

**Closing the Data Set**

The \texttt{CLOSE} macro instruction restores one or more DCBs so that processing of their associated data sets can be terminated. Issue \texttt{CLOSE} for all DCBs that were used by your channel programs. Some of the procedures performed when \texttt{CLOSE} is executed are:

- Releasing data extent block
- Removing information transferred to DCB fields when \texttt{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 without the nocapture option and LOC=ANY in the DCBE macro was not specified.

When CLOSE is issued for a data set on a magnetic tape volume, the system processes labels 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 203.

The parameters and different forms of the CLOSE macro instruction are described in z/OS DFSMS Macro Instructions for Data Sets.

Control Block Fields

Control block field information covered here includes fields of the DCB, DCBE, input/output block, IOBE, IEDB, event control block, and data extent block.

To use the DCBD macro to map the DCB, see “Mapping the DCB” on page 207.

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 z/OS DFSMS 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. If the DCBMACRF field indicates that a DCB portion before this is missing or short, the system ignores values in those fields.
- 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, 16 bytes for this portion are 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 14 on page 198 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 “Mapping the DCB” on page 207.

The EXCP DCB does not have a field to contain the current or maximum block size but the DCBE does have such a field.
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.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>BUFNO</td>
</tr>
<tr>
<td>24</td>
<td>BUFL</td>
</tr>
<tr>
<td>28</td>
<td>IOBAD</td>
</tr>
<tr>
<td>32</td>
<td>BFTEK, BFALN</td>
</tr>
<tr>
<td>36</td>
<td>RECFM</td>
</tr>
<tr>
<td>40</td>
<td>(TIOT)</td>
</tr>
<tr>
<td>44</td>
<td>(IFLGS)</td>
</tr>
<tr>
<td>48</td>
<td>(OFLGS)</td>
</tr>
<tr>
<td>52</td>
<td>OPTCD</td>
</tr>
<tr>
<td>56</td>
<td>Reserved</td>
</tr>
<tr>
<td>60</td>
<td>EOEA</td>
</tr>
<tr>
<td>64</td>
<td>SIOA</td>
</tr>
<tr>
<td>68</td>
<td>XENDA</td>
</tr>
</tbody>
</table>

Common Interface

Foundation Block Extension

Foundation Block

EXCP Interface

Figure 14. Data Control Block Format for EXCP (After OPEN)

The fields in parentheses represent system information that is not associated with parameters of the DCB macro instruction.
DCB Fields that do not have Macro Parameters

**DCBOFLGS:**  See “Handling End of Volume and End-Of-Data-Set Conditions” on page 194.

**DCBIFLGS:**  See “Processing the I/O Completion Status” on page 189 and “Interruption Handling and Error Recovery Procedures” on page 190.

These are the bits that EXCP uses in DCBIFLGS after the DCB is OPEN:

**Table 37. Bits that EXCP uses in DCBIFLGS after the DCB is OPEN**

<table>
<thead>
<tr>
<th>Bits that EXCP uses in DCBIFLGS after the DCB is OPEN</th>
</tr>
</thead>
<tbody>
<tr>
<td>11. .... DCBIFPEC</td>
</tr>
<tr>
<td>..11 .... DCBIFPCT</td>
</tr>
<tr>
<td>.... 11.. DCBIFIOE</td>
</tr>
</tbody>
</table>

**DCBFIOT:**  If the data set allocation was dynamic and had the XTIOT (S99TIOEX) or nocapture (S99ACUCB) option, then this field contains zeroes. You can use DEBXTNP, DEBXDSAB and DSABFIOT to get the TIO or XTIOT entry address. If the allocation was not dynamic or did not have the XTIOT or nocapture option, this field is unsigned (16-bit) offset in the TIO to the entry.

An entry in the TIO or XTIOT is mapped by the IEFTIOT1 macro.

**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. You can code a dummy value and change it before issuing OPEN.

**MACRF=E**

The EXCP or EXCPVR 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 EOV and CLOSE functions can record the block count in the IBM standard or ISO/ANSI standard trailer labels.

On input, the EOV and CLOSE functions can compare the DCB block count with the block count in the standard trailer labels to detect missing or duplicate blocks. This is supported on reel and cartridge tapes.
On input or output, the EOV and CLOSE functions can compare the DCB block count with the block count calculated from tape cartridge subsystems to detect missing or duplicate blocks. The system does this for any label type and for unlabeled tapes. Refer to *z/OS DFSMS Using Magnetic Tapes*.

For magnetic tape devices with reels (not cartridges), coding REPOS=Y allows the dynamic device reconfiguration (DDR) function to move the volume to another tape drive if the drive has a failure. With a cartridge, DDR can do this without your program maintaining block count.

For magnetic tape devices with reels (not cartridges), restart requires the block count in order to restore the tape to its position at the time of the checkpoint. With a cartridge, restart can do this without requiring that you maintain a block count.

The system performs the following tasks:
- Y-The count is accurate.
- N-The block count is inaccurate.

If the operand is omitted, N is assumed.

### EXCP Interface Parameters

If you do not code any of the parameters described here, the DCB macro does not create this portion of the block and OPEN will ignore any value that your program sets. You can code values to affect the macro expansion and change any of these fields before the end of the DCB OPEN exit routine. For more information on EXCP appendages, see "Making Appendages Available to the System" on page 224.

- **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 "Using the EXCPVR macro instruction" on page 186 and "Page Fix and EXCPVR Start I/OAppendage" on page 226.

- **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 (four 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**

The IMSK parameter lets you specify whether or not you want to ignore system error routines:

- Code X'FFFFFFF' to specify that you want the system to use the error routines.
- If you code any value other than X'FFFFFFF', the system will not use the system's error routines. This causes one of the DCBIFOIOE bits to be set on. See "DCBIFLGS" on page 199.

**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 *z/OS DFSMS 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 "Handling End of Volume and End-Of-Data-Set Conditions" on page 194 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), and its 31-bit EODAD field is non-zero, the system uses that address instead.

**DSORG=PS or PO or DA**

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:

<table>
<thead>
<tr>
<th>Code</th>
<th>DCB Format:</th>
</tr>
</thead>
<tbody>
<tr>
<td>PS</td>
<td>QSAM or BSAM</td>
</tr>
<tr>
<td>PO</td>
<td>BPAM</td>
</tr>
<tr>
<td>DA</td>
<td>BDAM</td>
</tr>
</tbody>
</table>

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 203 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 *z/OS DFSMS 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. An alternate way to set or learn the block size is to use the BLKSIZE field in the DCBE.

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. If the low order bit is on, the address is invalid and if you issue the FREEPOOL macro, it has no effect. The FREEPOOL macro sets the low order bit on. The DCB macro sets the low order bit on if the common interface portion is valid.

**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. 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 a 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 DASD 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 \textit{z/OS DFSMS Installation Exits}). 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. You specify DCBE= 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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCBH1</td>
<td>X'80'</td>
</tr>
<tr>
<td>DCBH0</td>
<td>X'04'</td>
</tr>
</tbody>
</table>

Coding DCBE= in the DCB macro sets these 2 bits on. For further DCBE information see \textit{“Data Control Block Extension (DCBE) Fields” on page 207}. See \textit{z/OS DFSMS Macro Instructions for Data Sets} for more information on the DCBE= parameter of the DCB macro.

**DEVD=code**
- The device in which the data set might reside. The codes are listed in order of descending space requirements for the DCB:

<table>
<thead>
<tr>
<th>Code</th>
<th>Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>DA</td>
<td>Direct access</td>
</tr>
<tr>
<td>TA</td>
<td>Magnetic tape</td>
</tr>
<tr>
<td>PR</td>
<td>Printer</td>
</tr>
<tr>
<td>PC</td>
<td>Card punch</td>
</tr>
<tr>
<td>RD</td>
<td>Card reader</td>
</tr>
</tbody>
</table>

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 DCBDEV as appropriate as it reaches each new data set. These values are described in \textit{z/OS DFSMS Macro Instructions for Data Sets}. 

Chapter 4. Executing Your Own Channel Programs 203
The fields in parentheses represent information not associated with parameters of the DCB macro instruction. EOV sets all of these fields. OPEN sets DCBDVTBA and DCBDEVT, but not DCBFDAD. Your program can modify DCBFDAD, DCBKEYLE, or DCBTRBAL as described in the text.

When writing on DASD, maintain certain fields of the device-dependent portion of the DCB. The system uses the information in the following instances:

- 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 using 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.

Maintain the following fields of the device dependent portion of the DCB when writing on DASD:

- 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 not directly calculate this number but use the TRKCALC macro (see “Performing Track Calculations (TRKCALC macro)” on page 307). 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 MBBCCHHR, see Table 44 on page 233

If space is available for output data sets, the system uses the contents of the full disk address (DCBFDAD) field, plus one block, to write a file mark when the DCB is closed or EOV is issued. If the track balance field (DCBTRBAL) is less than eight, 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 “Handling End of Volume and End-Of-Data-Set Conditions” on page 194. Use the OPEN macro instruction to initialize DCBDVTBA and DCBDEVT. You can use DCBDVTBA or DCBDVTBL with the DEVTAB parameter of the TRKCALC macro (see “Performing Track Calculations (TRKCALC macro)” on page 307 for the TRKCALC description).

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.

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.

The I/O process increments this field by the contents of the IOBINCAM field of the IOB upon completion of each I/O request.

To indicate to the system that your program is maintaining DCBBLKCT, code foundation block parameter REPOS=Y. See “Foundation Block Parameters” on page 199.
The following DCB operands pertain to specific devices and can be specified only when the DEVD parameter is specified.

**KEYLEN=**<br>Length<br>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=**<br>Value<br>For magnetic tape reels, the tape recording density in bits per inch is shown in the following table:

<table>
<thead>
<tr>
<th>Value</th>
<th>Density</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>800 (NRZI)</td>
</tr>
<tr>
<td>3</td>
<td>1600 (PE)</td>
</tr>
<tr>
<td>4</td>
<td>6250 (GCR)</td>
</tr>
</tbody>
</table>

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 [z/OS DFSMS Macro Instructions for Data Sets](https://www.ibm.com) for further information on DEN.

**TRTCH=**<br>Value<br>For magnetic tape subsystems with Improved Data Recording Capability, the tape recording techniques consist of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Tape recording technique</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMP</td>
<td>Record data in compacted format.</td>
</tr>
<tr>
<td>NOCOMP</td>
<td>Record data in standard uncompacted format.</td>
</tr>
</tbody>
</table>

For 7-track magnetic tape, the tape recording technique:

<table>
<thead>
<tr>
<th>Value</th>
<th>Tape recording technique</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMP</td>
<td>Record data in compacted format.</td>
</tr>
<tr>
<td>C</td>
<td>Data conversion feature is available.</td>
</tr>
<tr>
<td>E</td>
<td>Even parity is used. (If omitted, odd parity is assumed.)</td>
</tr>
<tr>
<td>T</td>
<td>BCDIC to EBCDIC translation is required.</td>
</tr>
</tbody>
</table>

**MODE=**<br>Value<br>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=**<br>Value<br>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.

\textbf{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.

\section*{Mapping the DCB}

In addition to the operands described in \textit{z/OS DFSMS Macro Instructions for Data Sets} for the DSORG parameter of the DCBD macro, you can specify the following operand:

\textbf{DSORG=XA or XE}

Specify the section of the DCB to be mapped.

\begin{itemize}
  \item \textbf{XA}: Specifies a DCB with the foundation block and EXCP interface.
  \item \textbf{XE}: Specifies a DCB with the common interface, foundation block extension, and foundation block.
\end{itemize}

Code DSORG=(XA,XE) to map all four sections of the DCB.

\section*{Data Control Block Extension (DCBE) Fields}

The data control block extension (DCBE) provides further processing options. EXCP supports these options:

- The BLKSIZE parameter when you issue the OPEN and EOV macros.
- The BLOCKTOKENSIZE parameter to signify that your program is able to handle a DASD data set that has the large format attribute. The data set is not necessarily large. If you code BLOCKTOKENSIZE=LARGE, then it means that your program can handle a data set that exceeds 65535 tracks or might grow above that size. Look for these differences:
  - If the data set is a large format data set, the DSCB the DS1LARGE bit will be on and the two bytes in DS1LSTAR that contain a relative track number are logically extended with the DS1TTTHI byte. See the format 1 DSCB description in "Format-1 and Format-8 DSCBs" on page 5.
  - The two bytes in each DEBMTRK field are logically extended by the DEBNMTRKHI byte. See Appendix A, "Control Blocks," on page 443.
  - If your program calls either of the track conversion routines that CVTPRLTV or CVTPCNVT point to, then your program should use the +12 entry points instead of the +0 entry points.

If you do not code BLOCKTOKENSIZE=LARGE on the DCBE macro, then:

- If the OPEN macro option is not INPUT and the data set is large format, then OPEN will issue a 213-14 ABEND.
- If the OPEN macro option is INPUT and the data set has more than 65535 tracks on the volume, then OPEN will issue a 213-16 ABEND.

- The CAPACITYMODE parameter to write more data on an IBM 3590 Magnetic Tape Subsystem that emulates an IBM 3490 when you issue the OPEN macro.
- The EADSCB=OK parameter to specify that your application program supports one of the following, as appropriate:
  - \textbf{VT0C} that describes a volume supporting extended attribute DSCBs.

\paragraph{Reading and Modifying a Job File Control Block (RDJFCB Macro)} on page 284 describes how to open a VT0C.) An extended address volume may have extended attribute DSCBs. They are format-8 and format-9 DSCBs. If you do
not code this option, the OPEN function will issue ABEND 113-48 and message IEC142I. Code this option when your application program supports format-8 and format-9 DSCBs.

- A data set that has extended attribute DSCBs, Track addresses in DSCBs pointed to by a format-8 DSCB may contain cylinder addresses above 65,520. If you do not code this option, then OPEN issues a 113-44 ABEND and message IEC142I. Code this option when your application program supports MACRF=E (EXCP) and the data set has format 8 and 9 DSCBs.

- The EODAD parameter when you issue an EOV macro. If the DCBEEODAD field has a non-zero value when the system needs to use it, this value takes precedence over an EODAD specification in the DCB. See "Foundation Block Extension and Common Interface Parameters" on page 201.

- The LOC parameter when you issue an OPEN macro. To successfully open a data set that has been allocated with the XTIOT, UCB NOCAPTURE, or DSAB above the line dynamic allocation options, you must specify both:
  - DCBE option LOC=ANY. The LOC=ANY option is represented by the DCBELOCANY bit in the DCBE.
  - DEVSUPxx parmlib parameter NON_VSAM_XTIOT=YES. The DEVSUPxx parameter NON_VSAM_XTIOT is represented by DFAXTBAM bit in the DFA as mapped by IHADFA.

- The SYNC parameter to control buffered tape marks on an IBM 3590 when you issue the OPEN, EOV, or CLOSE macros.

Note that the DCBE can be above the 16 MB line even though your user program is running in 24-bit mode. See z/OS DFSMS Macro Instructions for Data Sets for more information concerning the DCBE.

Set and Retrieve Data Set Block Size

The DCBE has a field to contain the current or maximum block size for the data set.

In order to use this field, your program must code a value for the BLKSIZE keyword on the DCBE macro or turn on the DCBEULBI bit. The keyword value can be numeric or a non-relocatable symbolic expression. The value can be 0.

If you code a zero value or turn on DCBEULBI, then OPEN stores into DCBEKLKSI the block size value from the data set label if opening for input or output with DISP=MOD, which can be disk or tape. If your program issues the OPEN macro with the OUTPUT or OUTIN option, then OPEN tries to calculate an optimal value for BLKSIZE as it does for BSAM and QSAM. The system-determined block size function is described in z/OS DFSMS Using Data Sets. The basic principle is that your program supplies the LRECL value and RECFM value (not U) and OPEN calculates an optimal BLKSIZE value for the device and stores it into DCBEKLKSI.

Attention: OPEN might calculate a BLKSIZE value in the DCBE that exceeds the maximum that is supported by the regular access methods or other programs that read your program's data sets. For example, on magnetic tape the value probably will exceed 32760. If this is a problem, you can do the following:

Supply a DCB OPEN exit routine as described in z/OS DFSMS Using Data Sets. If your exit routine finds that the DCBEKLKSI field is still 0, it means that the data set label and the DD statement do not have a value for BLKSIZE. If your program is opening for output, it can leave the DCBE alone and let OPEN calculate an optimal BLKSIZE value or it can calculate one. This is your program's last chance to set it before the system stores the BLKSIZE value in various system control
blocks and the data set label. Your DCB OPEN exit routine can issue the DEVTYPE macro with the INFO=AMCAP parameter to learn the optimal and maximum BLKSIZE values for the device. If it is too large, your program can calculate a smaller valid value. Store a value into DCBEBLKS1 before OPEN can set it. The INFO=AMCAP parameter of DEVTYPE is described on “DEVTYPE—Info Form” on page 270.

OPEN builds the DEB after calling the DCB OPEN exit routine. If your program calculates a maximum block size, it can take into consideration a value for BLKSIZELim coded on the DD statement. To find this value, issue the RDJFCB macro with an X'13' code. See “Reading and Modifying a Job File Control Block (RDJFCB Macro)” on page 284.

Why should your program do this?
• To decrease your program’s dependence on the device type
• The system stores this value in the data set label for use by z/OS and on other systems
• The system stores this value in various SMF records to improve monitoring of system resources
• DFSMSrmm™ retains this information for its tape reports.

Unlike the regular access methods, EXCP processing does not do complete checking of the BLKSIZE value in an EXCP DCBE.

**Input/Output Error Data Block (IEDB) Fields**

The system uses the IEDB to provide extended error information, and for zHPF the failing storage address. The IEDB has two versions:
• The version 1 IEDB is 48 bytes.
• The version 2 IEDB is 96 bytes. This is the recommended version for zHPF channel programs.

You provide an IEDB by setting its address in an IOBE. 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. Refer to “Requesting Extended Error Information” on page 191 for more information on using the IEDB.
### Table 40. IEDB Structure Mapping

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length or Bit Pattern</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (X'0')</td>
<td>4</td>
<td>IEDBID</td>
<td>Eye catcher. Must be &quot;IEDB&quot;</td>
</tr>
<tr>
<td>4 (X'4')</td>
<td>1</td>
<td>IEDBVERS</td>
<td>Version.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• The version 1 IEDB is 48 bytes.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• The version 2 IEDB is 96 bytes. This is the recommended version for zHPF channel programs.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>For version 1, the macro defines IEDBVRS1 as the version constant. For version 2, the macro defines IEDBVRS2 as the version constant.</td>
</tr>
<tr>
<td>5 (X'5')</td>
<td>1</td>
<td>IEDBFLG1</td>
<td>Flags field.</td>
</tr>
<tr>
<td>1...</td>
<td>IEDBDSN</td>
<td></td>
<td>The sense data is invalid and begins with X'10FE'.</td>
</tr>
<tr>
<td>.1...</td>
<td>IEDBFSA</td>
<td></td>
<td>The failing storage address (IEDBFSA) is valid.</td>
</tr>
<tr>
<td>..xx</td>
<td>xxx</td>
<td></td>
<td>Reserved.</td>
</tr>
</tbody>
</table>
Input/Output Block (IOB) 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. 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 21 on page 212, 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.

You do not have to set the following IOB fields to any particular value before issuing EXCP because the system itself sets them:

- IOBSENS0
- IOBSENS1
- IOBECBC
- IOBCSW
- IOBSIOCC
- IOBCMD31
**IOBFLAG1 (1 byte)**

Set bit positions 0, 1, 6, and 7. One-bits in positions 0 and 1 (IOBDATCH and IOBCMDCH) 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 and tape 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 (IOBUNREL) 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 or XDAP macro with a BSAM, QSAM, or BPAM DCB, you should turn on bit 7 (IOBSPSVC) to prevent access-method appendages from processing the I/O request.

---

### Figure 21. Input/Output Block (IOB) Format

<table>
<thead>
<tr>
<th>Byte</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>IOBFLAG1</td>
<td>Bit positions 0, 1, 6, and 7.</td>
</tr>
<tr>
<td>1</td>
<td>IOBFLAG2</td>
<td>Bit positions 2 and 3.</td>
</tr>
<tr>
<td>2</td>
<td>IOBSENS0</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>IOBSENS1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>IOBECBCC</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>IOBECBPB</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>IOBFLAG3</td>
<td>Bit position 6 (IOBUNREL).</td>
</tr>
<tr>
<td>9</td>
<td>IOBSENS2</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>IOBSIOCC</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>IOBSTRTB</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>IOBFLAG4</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>IOBDCBPT</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>IOBRESTR</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>IOBRESTR+1</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>IOBINCAM</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>IOBINCC</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>IOBERRCT</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>IOSEEK</td>
<td>(first byte, M)</td>
</tr>
<tr>
<td>23</td>
<td>IOSEEK</td>
<td>(second through eight bytes, BBCCHHR)</td>
</tr>
<tr>
<td>27</td>
<td>IOSEEK</td>
<td>(third through eight bytes, BBCCHHR)</td>
</tr>
<tr>
<td>28</td>
<td>IOSEEK</td>
<td>(fourth through eight bytes, BBCCHHR)</td>
</tr>
<tr>
<td>29</td>
<td>IOSEEK</td>
<td>(fifth through eight bytes, BBCCHHR)</td>
</tr>
<tr>
<td>30</td>
<td>IOSEEK</td>
<td>(sixth through eight bytes, BBCCHHR)</td>
</tr>
<tr>
<td>31</td>
<td>IOSEEK</td>
<td>(seventh through eight bytes, BBCCHHR)</td>
</tr>
<tr>
<td>32</td>
<td>IOSEEK</td>
<td>(eighth through eight bytes, BBCCHHR)</td>
</tr>
</tbody>
</table>

---

Direct Access, Teleprocessing, and Graphic Devices

Direct Access, Storage Devices (DASD)
**IOBFLAG2 (1 byte)**

If you set bit 6 in the IOBFLAG1 field to zero, bits 2 and 3 (IOBRRT3 and IOBRRT2) in this field must then be set to one of the following:

- 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 system 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 system 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.

The first six of these 16 bits have these device-independent meanings:

1... .... Command reject
.1.... .... Intervention required
...1.... Bus out check
....1.... Equipment check
.... 1... Data check
..... 1... Overrun

The last ten of these 16 bits have device-dependent meanings. See appropriate hardware documentation.

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 190.

**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 (ECB) Fields” on page 220.

**IOBECBPT (3 bytes)**

The address of the 4-byte event control block (ECB) you have provided.

**IOBFLAG3 (1 byte) and IOBCSW (7 bytes)**

The system stores status information in these eight bytes. See “IOBFLAG3 and IOBCSW Format for Different Channel Program Types” on page 214.

**IOBSIOCC (1 byte)**

If the channel program uses format 0 CCWs, bits 2 and 3 contain the start subchannel (SSCH) condition code for the instruction the system issues to start the channel program.

If this is a format 1 CCW channel program or is a zHPF channel program, then field IOBSIOCC is redefined as field IOBSTART, which contains the four byte starting address of the channel program to be executed. (The IOBE field IOBESIOC is used instead of IOBSIOCC.)
**IOBSTRTB (3 bytes)**
If the channel program uses format 0 CCWs, the three byte 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 or EXCPVR. Refer to "Requesting Extended Error Information" on page 191 and "Requesting Different Levels of ERP Processing" on page 192.

You must set IOBCEF on if you want to use format 1 CCWs, 64 bit IDAWS, MIDAWS, or a zHPF channel program.

**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 (PURGE and RESTORE macros)" on page 304.

**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 17 on page 205.

**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.

**IOBFLAG3 and IOBCSW Format for Different Channel Program Types**
The fields IOBFLAG3 and IOBCSW have different meanings depending on the type of channel program and its format.

For format 0 CCW channel programs, these fields have the following format:
For format 1 CCW channel programs, these fields have the following format:

<table>
<thead>
<tr>
<th>8(8)</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>IOBFLAG3</td>
<td>IOBCMDA</td>
<td></td>
</tr>
<tr>
<td>12(C)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IOBUSTAT</td>
<td>IOBCSTAT</td>
<td>IOBRESCT</td>
</tr>
</tbody>
</table>

Figure 22. IOBFLAG3 and IOBCSW fields for format 0 channel program

For format 1 CCW channel programs, these fields have the following format:

<table>
<thead>
<tr>
<th>8(8)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>IOBCMD31</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>12(C)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>IOBUSTAT</td>
<td>IOBCSTAT</td>
</tr>
</tbody>
</table>

Figure 23. IOBFLAG3 and IOBCSW fields for format 1 channel program

For zHPF channel programs, these fields have the following format:

<table>
<thead>
<tr>
<th>8(8)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>IOBCMD31</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>12(C)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>IOBUSTAT</td>
<td>IOBCSTAT</td>
</tr>
</tbody>
</table>

Figure 24. IOBFLAG3 and IOBCSW fields for zHPF channel program

**IOBFLAG3**
This field is used by the system for format-0 CCW channel programs only.

**IOBCMDA**
The 24-bit ending CCW virtual address or zero for format-0 CCW channel programs. The ending CCW address points after the last executed CCW in your channel program. The ending address may be zero if either the system determined that the address was invalid or the last executed CCW was a CCW that was added by the system.

**IOBCMD31**
The 31-bit ending virtual address or zero for format-1 CCW channel programs and zHPF channel programs. For format-1 channel programs, the ending address points after the last executed CCW in your channel program. For
zHPF channel programs, the ending address points to the TCW. The ending address may be zero if either the system determined that the address was invalid, or for CCW channel programs, the last executed CCW was a CCW that was added by the system.

**IOBUSTAT**

The device status (previously called unit status).

**IOBCSTAT**

The subchannel status (previously called channel status). If the ending address is zero or the subchannel status byte in the IOB (IOBCSTAT) shows any of the following errors: program check, protection check, channel data check, channel control check, interface control check, or chaining check, and your appendage determines that the ERP has not yet run, then let the ERP try to recover and do not modify IOBSTART. If the ERP has completed and one or more of these six bits is on or the address is zero, then the status of the channel program is not known.

**IOBRESCT**

The residual count for format-0 and format-1 CCW channel programs, which is the number of bytes not transferred in the last CCW. For zHPF channel programs, the residual count is four bytes and contained in the IOBE.

**IOBFCXST**

Used by the system.

**IOBSESTA**

The subchannel extended status for zHPF channel programs. The subchannel extended status further qualifies subchannel related errors defined in IOBCSTAT. Macro IHASESQ defines the different subchannel extended status values.

### Input/Output Block Common Extension (IOBE) 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 the IOSDIOBE macro).

Set reserved bytes to X'00'. The IOBE can reside above or below 16MB virtual.

The constant IOBELNTH is set to the length of the IOBE and should be used when obtaining and clearing storage. In addition, the constant IOBEEND represents the end of the IOBE.
Table 41. IOBE Structure Mapping

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length or Bit Pattern</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (X'0')</td>
<td>4</td>
<td>IOBEID</td>
<td>Eye catcher (&quot;IOBE&quot;)</td>
</tr>
<tr>
<td>4 (X'4')</td>
<td>1</td>
<td>IOBEVERS</td>
<td>Version number (X'01')</td>
</tr>
<tr>
<td>5 (X'5')</td>
<td>1</td>
<td>IOBEFLG1</td>
<td>Reserved.</td>
</tr>
</tbody>
</table>

Figure 25. Format of an IOBE, mapped by the IOSDIOBE macro

Chapter 4. Executing Your Own Channel Programs 217
<table>
<thead>
<tr>
<th>Offset</th>
<th>Length or Bit Pattern</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 (X'6')</td>
<td>1</td>
<td>IOBEFLG2</td>
<td>Flag field 2. Set by issuer of EXCP or EXCPVR.</td>
</tr>
<tr>
<td>1...</td>
<td>...</td>
<td>IOBEMIDA</td>
<td>The channel program might have one or more CCWs that has the MIDA (modified indirect data addressing) bit on and point to a MIDAL (modified indirect addressing list). Supported only for EXCPVR requests. This bit must set to zero if you are using a zHPF channel program.</td>
</tr>
<tr>
<td>.1...</td>
<td>...</td>
<td>IOBEP</td>
<td>The user allows unlimited prefetching of CCWs. (Unlimited prefetching of the IDAWs, MIDAWs, and data associated with the current or prefetched CCW is always allowed.) If zero, no prefetching is allowed, except in the case of data chaining on output, where prefetching of one CCW describing a data area is allowed. Prefetching of CCWs applies only for FICON channels. It has no effect on Enterprise Systems Connection (ESCON®) or parallel channels. This bit applies to EXCPVR; EXCP always uses unlimited prefetching. This bit must set to zero if you are using a zHPF channel program.</td>
</tr>
<tr>
<td>...1...</td>
<td>...</td>
<td>IOBECPNM</td>
<td>The user guarantees that the channel program will not be modified during execution other than to add CCWs at the end. This bit is supported only for EXCPVR requests. EXCP requests that do not run in an authorized V=R address space cannot be modified during execution. This bit must set to zero if you are using a zHPF channel program.</td>
</tr>
<tr>
<td>...1...</td>
<td>...</td>
<td>IOBEEIDA</td>
<td>Any IDAWs are eight bytes each and point to 4096-byte boundaries with the possible exception of the first IDAW in each list. Currently supported only on direct access storage device (DASD) and on IBM-supplied cartridge tape devices. Bit UCBEIDAW in the UCB tells you whether the device supports 64-bit IDAWs. Supported for EXCP and EXCPVR and for format-0 and format-1 channel programs. Not currently supported in VIO. This bit must set to zero if you are using a zHPF channel program.</td>
</tr>
<tr>
<td>....</td>
<td>1...</td>
<td>IOBEPICS</td>
<td>PCI synchronization. Channel must synchronize after the next CCW following a PCI (at CCW+8). Supported for EXCPVR only if IOBEP is on. Not supported in VIO. EXCP requests never require PCI synchronization. This bit must set to zero if you are using a zHPF channel program.</td>
</tr>
<tr>
<td>....</td>
<td>.1...</td>
<td>IOBNORWS</td>
<td>No read/write synchronization. The channel should not synchronize on read/write transitions. User guarantees that the reads and writes are from different areas. Always supported for EXCP. Supported for EXCPVR only if IOBEP is on. This bit must set to zero if you are using a zHPF channel program.</td>
</tr>
<tr>
<td>....</td>
<td>...1...</td>
<td>IOB2CSWS</td>
<td>Two channel status words. If an error occurs where the control unit is executing ahead of the channel, the system returns two ending CCW addresses. The second ending CCW address is in the IEDB. Always allowed for EXCP. Supported for EXCPVR only if IOBEP is on. If this bit is off when the control unit has an error when executing ahead of the channel, the system simulates an invalid ending CCW address. This bit must set to zero if you are using a zHPF channel program.</td>
</tr>
<tr>
<td>....</td>
<td>...1...</td>
<td>IOBEFMT1</td>
<td>Channel program is format-1 CCWs and IOBSTART contains a 31-bit address. This bit also causes the system to use IOBESIOCC instead of IOBSIOCC and for the ending CCW address to be in IOBCMD31 instead of the three bytes in IOBCMDA. Supported for both EXCP and EXCPVR requests. This bit must set to zero if you are using a zHPF channel program.</td>
</tr>
<tr>
<td>7 (X'7')</td>
<td>1</td>
<td>IOBEERPM</td>
<td>Mask indicating the functions the ERP is allowed to perform.</td>
</tr>
<tr>
<td>1...</td>
<td>IOBEMSG</td>
<td>User allows basic error recovery procedures (ERP) recovery and permanent error messages that do not require interaction with the system or an operator. See &quot;Requesting Different Levels of ERP Processing&quot; on page 87.</td>
<td></td>
</tr>
<tr>
<td>....</td>
<td>xxx xxxxxx</td>
<td>Reserved.</td>
<td></td>
</tr>
</tbody>
</table>
### Table 41. IOBE Structure Mapping (continued)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length or Bit Pattern</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 (X'8')</td>
<td>4</td>
<td>IOBEUSER</td>
<td>Address field for user use.</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>IOBEUPTR</td>
<td>Character field for user use.</td>
</tr>
<tr>
<td>12 (X'C')</td>
<td>4</td>
<td>IOBEIEDB</td>
<td>Zero or address of IEDB if you want extended error information. For more information see, “[Requesting Extended Error Information” on page 191.</td>
</tr>
<tr>
<td>16 (X'10')</td>
<td>1</td>
<td>IOBEFLG3</td>
<td>Flag byte 3.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IOBENSER</td>
<td>Set by user to allow the device to bypass the channel program extent collision checking. Extent range enforcement remains active. Meaningful only if the DASD supports it; has no effect otherwise. Can improve performance on IBM 2105 Enterprise Storage Server™ and newer DASD control units when using multisystem access or multiple parallel access volumes (PAV).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IOBENVAL</td>
<td>Set by user to allow the device to bypass the validation checking of the parameters on define extent and locate record commands. Extent enforcement remains active; meaningful only if the DASD device supports it, has no effect otherwise. Can improve performance on IBM 2105 Enterprise Storage Server™ and newer DASD control units when using multisystem access or multiple parallel access volumes (PAV).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IOBEDSMC</td>
<td>Set by user to disable streaming mode control. This bit must set to zero if you are using a zHPF channel program.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IOBEIOT</td>
<td>If the user sets this to 0, IOBETIME applies to the request only while the request is active. If this is 1, IOBETIME applies while the request is queued or active.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IOBEDCWOOffsetValid</td>
<td>For zHPF channel programs, the DCW offset in IOBEDCWOOffset is valid.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IOBEResCountValid</td>
<td>For zHPF channel programs, the residual count in IOBEResCount is valid.</td>
</tr>
<tr>
<td>17 (X'11')</td>
<td>1</td>
<td>IOBESIOC</td>
<td>Same format as IOBSIOCC. Valid only if IOBEFMT1 is on.</td>
</tr>
<tr>
<td>18 (X'12')</td>
<td>1</td>
<td>IOBETIME</td>
<td>Time limit. If this value is non-zero, it is the number of seconds the user allows the request to take. Has an effect if the DEBACCS field shows the DCB is open for input. There will be no message or error recording due to reaching the limit. Bit IOBEIOT specifies what time is measured.</td>
</tr>
<tr>
<td>19(X'13')</td>
<td>1</td>
<td>IOBEFLG4</td>
<td>Flag byte 4.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IOBEZHPF</td>
<td>This is a zHPF channel program.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>This bit must be set to zero for CCW channel programs.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Reserved.</td>
</tr>
<tr>
<td>20(X'14')</td>
<td>4</td>
<td>IOBEResCount</td>
<td>For zHPF channel programs, this field contains the residual count. This field is valid only if IOBEResCountValid is on.</td>
</tr>
<tr>
<td>24(X'18')</td>
<td>2</td>
<td>IOBEDCWOOffset</td>
<td>For zHPF channel programs, this field contains the offset within the TCA of the DCW that was partially or completely executed. If the channel program could not be completed, this offset identifies the DCW for which processing could not be completed. This field is valid only if IOBEDCWOOffsetValid is on.</td>
</tr>
<tr>
<td>26(X'1A')</td>
<td>1</td>
<td>IOBEDDPC_RC</td>
<td>For zHPF channel programs, this field contains the device detected program check reason code. This field is valid only when the subchannel status indicates a program check and the subchannel extended status indicates a device dependent program check. The reason codes are documented in z/Architecture® Principles of Operation.</td>
</tr>
<tr>
<td>27(X'1B')</td>
<td>1</td>
<td>IOBEDDPC_RCQ</td>
<td>For zHPF channel programs, this field contains the first byte of the device detected program check reason code qualifier. This field is valid only when the subchannel status indicates a program check and the subchannel extended status indicates a device dependent program check. The reason code qualifiers are documented in z/Architecture Principles of Operation.</td>
</tr>
</tbody>
</table>
Table 41. IOBE Structure Mapping (continued)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length or Bit Pattern</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>28(X'1C')</td>
<td>1</td>
<td>IOBERCOD</td>
<td>The I/O completion reason code. This is the second byte of the completion code for the channel program. The system also places this code in the second byte of the event control block when the channel program is posted complete. The reason codes applicable to EXCP or EXCPVR and their meanings are listed in “Event Control Block (ECB) Fields.”</td>
</tr>
<tr>
<td>29(X'1D')</td>
<td>19</td>
<td>Reserved</td>
<td></td>
</tr>
</tbody>
</table>

Event Control Block (ECB) 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 26). 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.

The ECB is mapped by the IHAECB macro.

You do not have to set any particular values in the ECB before issuing EXCP or EXCPVR because the system clears it.

<table>
<thead>
<tr>
<th>Byte 0</th>
<th>Byte 1</th>
<th>Bytes 2 and 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Completion code byte</td>
<td>Reason code</td>
<td>Undefined</td>
</tr>
<tr>
<td>Bit 0</td>
<td>Bit 1</td>
<td></td>
</tr>
</tbody>
</table>

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 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 225 and “Channel-End Appendage” on page 231.

The system sets these codes in IOBECBCC and IEDBCOD before posting the ECB.

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>41</td>
<td>Permanent I/O error</td>
</tr>
<tr>
<td>42</td>
<td>Extent error (DASD only). Either IOBSEEK lies outside the extents described by the DEB or the channel program attempted execution outside the current extent.</td>
</tr>
</tbody>
</table>
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.

A program check or machine check occurred in IOS while processing the I/O request.

The channel program was purged.

An error occurred during tape repositioning that was requested by the tape ERP.

Error recovery routines were entered following a direct access error but are unable to read the home address or record 0.

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.

This completion code appears only in the IEDB, if one was provided. It is translated to a 41 completion code prior to updating the ECB and IOBECBCC.

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.

Normal I/O completion, but does not appear in the ECB.

Reason code byte
Reason code for the completion code.

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0E</td>
<td>A zHPF channel program was specified but the device does not support zHPF. This reason code appears with completion code 41.</td>
</tr>
<tr>
<td>10</td>
<td>A zHPF channel program was specified, but the I/O request was terminated because a capability needed by the I/O request is not supported by the processor, device, or software. This reason code is presented if an EXCPVR request is issued when the processor and device do not support the zHPF incorrect length facility. This reason code appears with completion code 41.</td>
</tr>
</tbody>
</table>
Data Extent Block (DEB) 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 A, “Control Blocks,” on page 443. You can find a complete view of DEB fields on page 443.

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 Table 42.

<table>
<thead>
<tr>
<th>Appendage</th>
<th>Description</th>
<th>When Called</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABE</td>
<td>Abnormal-end</td>
<td>Abnormal conditions</td>
</tr>
<tr>
<td>CHE</td>
<td>Channel-end</td>
<td>Channel-end, unit exception, wrong-length record</td>
</tr>
<tr>
<td>EOE</td>
<td>End-of-extent</td>
<td>DASD track address in I/O block outside allocated extent limits</td>
</tr>
<tr>
<td>PCI</td>
<td>Program-controlled interruption</td>
<td>When one or more PCI bits are on in a channel program</td>
</tr>
<tr>
<td>PGFX</td>
<td>Page-fix</td>
<td>Prior to SIO for EXCPVR requests</td>
</tr>
<tr>
<td>SIO</td>
<td>Start-I/O</td>
<td>Just prior to translating channel program</td>
</tr>
</tbody>
</table>

Appendages get control in supervisor state, protection key 0, receiving the pointers from the system described in the following table. The appendages receive control in 24-bit addressing mode and must return in the same mode.

This information is not part of the intended interface.

Register

<table>
<thead>
<tr>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
</tbody>
</table>
always are entered in 24–bit mode. The UCB address is captured by OPEN or EOV until EOV or CLOSE uncaptures it. If the DCBE option "LOC" was not set or defaulted to "BELOW", that is LOC=BELOW or not coded, or the "NON_VSAM_XTIOT" option of the DEVSUPxx member of PARMLIB was set or defaulted to "NO", that is NON_VSAM_XTIOT=NO or not coded. In other words, OPEN does not capture the UCB if LOC and NON_VSAM_XTIOT were specified as follows: LOC=ANY and NON_VSAM_XTIOT=YES.

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 Table 43.

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 the following 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. Table 43 summarizes register conventions. Note that the need to save and restore registers applies to all eight bytes in each register.

- 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.

The information here describes appendage types, with explanations of when they are entered, 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.

**Table 43. Entry Points, Returns, and Available Work Registers for Appendages**

<table>
<thead>
<tr>
<th>Appendage</th>
<th>Entry Point</th>
<th>Returns</th>
<th>Available Work Registers</th>
</tr>
</thead>
</table>
| EOE       | Reg 15      | • Reg 14 + 0 - Call ABE  
          |             | • Reg 14 + 4 - Skip  
          |             | • Reg 14 + 8 - Try again | Reg. 10, 11, 12, and 13 |
| SIO       | Reg 15      | • Reg 14 + 0 - Normal  
          |             | • Reg 14 + 4 - Skip | Reg. 10, 11, and 13 |
| PCI       | Reg 15      | • Reg 14 + 0 - Normal | Reg. 10, 11, 12, and 13 |
| PGFX      | Reg 15+4    | • Reg 14 + 0 - Normal | Reg. 10, 11, and 13 |
| CHE       | Reg 15      | • Reg 14 + 0 - Normal  
          |             | • Reg 14 + 4 - Skip  
          |             | • Reg 14 + 8 - Re-EXCP  
          |             | • Reg 14 + 12 - By pass | Reg. 10, 11, 12, and 13 |
Table 43. Entry Points, Returns, and Available Work Registers for Appendages (continued)

<table>
<thead>
<tr>
<th>Appendage</th>
<th>Entry Point</th>
<th>Returns</th>
<th>Available Work Registers</th>
</tr>
</thead>
</table>
| ABE       | Reg 15      | • Reg 14 + 0 - Normal  
            |             | • Reg 14 + 4 - Skip    
            |             | • Reg 14 + 8 - Re-EXCP  
            |             | • Reg 14 + 12 - 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

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 [z/OS MVS Initialization and Tuning Guide](#).  

The Authorized Appendage List (IEAPP00)

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 IEAPP00 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 IEAPP00 to SYS1.PARMLIB and put the names of the appendages in it with the IEBUPDTE utility or with another program that updates partitioned data sets. See the description of the IEAPP00 parmlib member in [z/OS MVS Initialization and Tuning](#) Reference.  

The following example shows JCL statements and IEBUPDTE input that add IEAPP00 to SYS1.PARMLIB and put the names of one EOE appendage, two SIO appendages, two CHE appendages, and one ABE appendage in IEAPP00:

```
// JOB ...  
// EXEC PGM=IEBUPDTE,PARM='NEW'  
//SYSPRINT DD SYSOUT=A  
//SYSUT2 DD DSN=SYS1.PARMLIB,DISP=SHR  
//SYSIN DD *  
./ ADD NAME=IEAPP00  
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:

```plaintext
// 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. IOBSTART contains the virtual address of the start of the channel program.

The start I/O appendage may build a zHPF channel program or a CCW channel program. The caller does not have to set the IOBEZHPF bit prior to issuing the EXCPVR request. If the start I/O appendage builds a zHPF channel program, it should set the IOBEZHPF bit if it is not already set, and reset the IOBEFMT1 bit. Otherwise, it should reset the IOBEZHPF bit.

If the device is not enabled for zHPF or does not have the necessary capabilities, the start I/O appendage should either build a CCW channel program, or post the request in error and return to EXCP indicating that the I/O operation should be skipped (return +4).

Optional return vectors give the I/O requester the following choices:
- Reg. 14 + 0 — Normal return. The system should translate the channel program, if required, and initiate the 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, as follows.
  – Set the completion code in register 10. This register is used to post the ECB.
  – Set register 11 to the ECB address in the IOB.
  – Issue the POST macro as shown in the example below:
    
    ```
    POST (11),(10),LINKAGE=BRANCH
    ```
    
    For more information on the POST macro, see [z/OS MVS Programming: Authorized Assembler Services Guide](https://www.ibm.com/support/knowledgecenter/ST3XAY_2.3.1/ptxptg/gpgpgm680.html) and [z/OS MVS Programming: Authorized Assembler Services Reference LLA-SDL](https://www.ibm.com/support/knowledgecenter/ST3XAY_2.3.1/ptxptg/gpgpgm680.html).

### 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.

Note that you cannot fix pages that were allocated with CONTROL=UNAUTH on the IARV64 macro. Unauthorized programs cannot override this setting, but they can allocate 64-bit storage with the IARST64 macro.

### PGFX Appendage

This appendage creates a list of the addresses of the areas that must be fixed to prevent paging exceptions during the execution of the current input/output (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:

```
03 31 32 33 63
3 3 3 3 3
Starting virtual address of area to be fixed
3 3 3 3 3
03 3 3 Ending virtual address of area to be fixed + 1
3 3 3 3
3 3
```

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 EXCPVR 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. The page-fix list must be in page-fixed storage.

**SIO Appendage**

If you are using EXCPVR to execute a channel program and your channel program does not already contain real addresses, 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 address lists (IDALs, MIDALs, or TIDALs) and turn on the appropriate indirect address list flag for the channel program.

You can use EXCPVR for VIO data sets with the following considerations for the SIO appendage:

- The use of IDAWs and an IDAL is not required.
- MIDALs and zHPF channel programs are not supported for VIO.
- Addresses in the CSWs and IDAWs 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.

When building an IDAL for CCW channel programs consider the following:

- Bit IOBEEIDA, described in "Input/Output Block Common Extension (IOBE) Fields" on page 216, specifies whether you are using 31-bit or 64-bit IDAWs. With 31-bit IDAWs, use 2 KB boundaries when determining whether a storage boundary is crossed for a CCW. With 64-bit IDAWs, use 4 KB boundaries.
- The LRA instruction returns a 31-bit central storage address regardless of whether you are in 24-bit or 31-bit addressing mode, but fails in those addressing modes if the central storage address is above 2 GB. If the central storage address is above 2 GB, you must either use the LRAG or STRAG instruction to convert the virtual address to a real address or use the LRA instruction (after first switching to 64-bit addressing mode). If your program switches to 64-bit addressing mode, you must ensure that you save and restore the high 32 bits of any register that you modify. If you fail to do this, unpredictable results can occur for other 64-bit programs in your address space.

See "IDAW Requirements for EXCP Requests" on page 177 for more information on creating IDAWs for EXCPVR requests.

**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.

Note that PCI and PCI appendages are not supported for zHPF channel programs.

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 188.)

To post the channel program from a PCI appendage to an EXCP request (EXCP V=V), use the procedure described in "SIO Appendage" on page 227.

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 z/OS MVS Programming: Authorized Assembler Services Guide and z/OS MVS Programming: 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:** If you specify the ASCB parameter with MEMREL=NO, 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, because only registers 9 and 14 are 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. If you do not have the ASCB address, you can use the following procedure to obtain the ASCB address:
   a. Issue the EPAR instruction to obtain the ASID. For information on the EPAR instruction, see z/Architecture Principles of Operation.
   b. Issue the LOCASCB macro to obtain the ASCB address. The LOCASCB macro is documented in z/OS MVS Programming: Authorized Assembler Services Reference LLA-SDU.
7. Issue the POST macro, as shown in the following example:

   ```assembler
   POST (11),(10),ASCB=(13),ERRET=(12),ECBKEY=(0),LINKAGE=BRANCH
   ```
8. 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 current extent for the data set, the request is executed; if the new seek address is not within the current extent for the data set, 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.

The simplest way to compare two track or block addresses to see their relative positions on the volume is to use the CLC instruction; however that technique works only if both addresses (CCHH or CCHHR) contain cylinder numbers of less than 65536. To handle any addresses that are more or less than cylinder 65536, use the TRKADDR macro with the COMPARE option. See “Compare two track addresses (TRKADDR COMPARE)” on page 318.

You can use the following optional return addresses:

- Contents of register 14 to return control to the system, causes the ABE appendage to be entered. The system places an end-of-extent error code (X’42’) in the ECB code field of the input/output block for subsequent posting in the ECB.
- 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. The following apply:

- 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 231.
- 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.
- 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.
- 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.
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.

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 IOBIOERR 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.

If the ending address is zero or the subchannel status byte in the IOB (IOBCSTAT) shows any of the following errors: program check, protection check, channel data check, channel control check, interface control check, or chaining check, and your abnormal end appendage determines that the ERP has not yet run, then do not modify IOBSTART. This lets the ERP try to recover. If the ERP has completed and one or more of these six bits is on or the address is zero, then the status of the channel program is not known.

To determine if an error is permanent, check the IOBECBCC field of the IOB for a X'4x' completion code.

To determine the type of error, check the subchannel 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.

The appendage can request that a different type of channel program be started. For example:

- The original request was for a zHPF channel program but the device is no longer enabled for zHPF or does not support the zHPF capabilities required by the new channel program.
- The original request was for a non-zHPF channel program but the new I/O request allows a zHPF channel program to be used.

The channel program type may be changed from non-zHPF to zHPF only if the caller passed an IOBE to EXCP. If the channel program type is changed, the appendage must set or reset the IOBEFMT1 and IOBEZHPF bits correctly to reflect the type of channel program. For example, if the original channel program was a format-1 CCW channel program and the new channel program is a zHPF channel program, then the IOBEFMT1 bit must be reset and the IOBEZHPF bit must be set. The system will call appendages, beginning with the SIO appendage, as if this were a new EXCP or EXCPVR request. Note that the EXCPVR page fix appendage will not be called again.
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 system, 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 IOBIOERR 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 IOBIOERR (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 IOBIOERR 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 the first bullet in “Abnormal-End Appendage” on page 229.

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. If the device is DASD, you can change the IOBSEEK field but the first byte must refer to an extent with the same UCB address. The system will call appendages, beginning with the SIO appendage, as if this were a new EXCP or EXCPVR request. Note that the EXCPVR page fix appendage will not be called again.

The appendage can request that a different type of channel program be started. For example:

- The original request was for a zHPF channel program but the device is no longer enabled for zHPF or does not support the zHPF capabilities required by the new channel program.
- The original request was for a non-zHPF channel program but the new I/O request allows a zHPF channel program to be used.

The channel program type may be changed from non-zHPF to zHPF only if the caller passed an IOBE to EXCP. If the channel program type is changed, the
appendage must set or reset the IOBEFMT1 and IOBEZHPF bits correctly to reflect the type of channel program. For example, if the original channel program was a format-1 CCW channel program and the new channel program is a zHPF channel program, then the IOBEFMT1 bit must be reset and the IOBEZHPF bit must be set.

- 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 [z/OS MVS Programming: Authorized Assembler Services Guide](https://www.ibm.com).)

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 conversion routine has two entry points. One, at the address in the CVTPCNVT field of the communication vector table (CVT), is intended for data sets with up to 65535 tracks. The other, at offset +12 from the address in CVTPCNVT, is intended for any data set. Call the conversion routine with one of the following instructions:

- BALR 14,15 Call +0 entry point for track conversion
- BASR 14,15 Call +0 entry point for track conversion
- BAL 14,12(,15) Call +12 entry point for track conversion
- BAS 14,12(,15) Call +12 entry point for track conversion

IBM does not provide a macro for this conversion routine's invocations.

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:
<table>
<thead>
<tr>
<th>Register</th>
<th>Use</th>
</tr>
</thead>
</table>
| 0        | Must be loaded with a 4-byte value of the relative track number. If entered at CVTPCNVT+0, this value must be in the form TTR\(n\), where:  
| \(TT\)  | The track number relative to the beginning of the data set  
| \(R\)   | The block identification on that track  
| \(n\)   | If the DEB is for a partitioned concatenation, (the DSORG field in the DCB indicates PO), then supply the concatenation number for the data set. A value of 0 indicates the first data set, 1 indicates the second data set and so forth. If your partitioned concatenation includes PDSEs or z/OS UNIX directories, each of them is represented by a dummy extent in the DEB. EXCP and EXCPVR are not valid for those extents. If the data set is not concatenated or the concatenation is not partitioned, set to 0.  
| If entered at CVTPCNVT+12, this value must be in the form TTTR, where:  
| \(TTT\) | The track number relative to the beginning of the data set  
| \(R\)   | The block identification on that track  

This entry point is intended for any type of data set.  
| 1        | Must be loaded with the address of the data extent block (DEB) of the data set. Each DEB resides below 16 MB. When you call the +0 entry point, the called routine clears the high order byte of this register. When you call the +12 entry point, this byte must contain X'00'. Note that if the DEB31UCB bit is zero, the UCB address field is three bytes in DEBUCBA. If the DEB31UCB bit is one, the UCB address field is four bytes in DEBUCBAD.  
| 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  
| \(CC\)  | Low order 16 bits of the cylinder number. The cylinder number is 28 bits on all currently supported DASD.  
| \(HH\)  | The actual track number in the low order four bits and the high order twelve bits of the cylinder number in the high order twelve bits.  
| \(R\)   | The block number.  
| 3–8     | Not used by the conversion routine.  
| 9–12    | Used by the conversion routine and not restored. |
**Table 44. Registers and Their Use for Converting Relative to Actual (continued)**

<table>
<thead>
<tr>
<th>Register</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>Used by the conversion routine and not restored. If you call the +12 entry point, the high order three bytes must contain zero and the low order byte must be as set as follows:</td>
</tr>
<tr>
<td></td>
<td>• If the DEB is for a partitioned concatenation, (the DSORG field in the DCB indicates PO), then supply the concatenation number for the data set. A value of 0 indicates the first data set, 1 indicates the second data set and so forth.</td>
</tr>
<tr>
<td></td>
<td>• If the data set is not concatenated or the concatenation is not partitioned, set to 0.</td>
</tr>
<tr>
<td>14</td>
<td>Must be loaded with the address to which control is to be returned after execution of the conversion routine.</td>
</tr>
<tr>
<td>15</td>
<td>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).</td>
</tr>
</tbody>
</table>

---

**Return Codes from the Relative to Actual Conversion Routine**

When control is returned to your program, register 15 contains one of the following return codes:

**Table 45. Relative to Actual Conversion Routine Return Codes**

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (X'00')</td>
<td>Successful conversion.</td>
</tr>
<tr>
<td>4 (X'04')</td>
<td>The relative track address converts to an actual track address outside the extents defined in the DEB.</td>
</tr>
<tr>
<td>8 (X'08')</td>
<td>Internal access method control blocks are invalid. Call IBM service if the control blocks should be valid.</td>
</tr>
<tr>
<td>12(X'0C')</td>
<td>Internal access method control blocks are invalid. Call IBM service if the control blocks should be valid.</td>
</tr>
<tr>
<td>16 (X'10')</td>
<td>Passed concatenation number is too big for the data set.</td>
</tr>
<tr>
<td>20 (X'14')</td>
<td>The combination of the DEBNMTRK and DEBNMTRKHI fields in the last extent shows that the input track number points into this extent, but the calculated CCHH does not lie within this extent.</td>
</tr>
</tbody>
</table>

---

**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 conversion routine has two entry points. One, at the address in the CVTPRLTV field of the communication vector table (CVT), is intended for data sets with up to 65535 tracks. The other, at offset +12 from the address in CVTPRLTV, is intended for any data set. Call the conversion routine with one of the following instructions:

- BALR 14,15 Call +0 entry point for track conversion
- BASR 14,15 Call +0 entry point for track conversion
- BAL 14,12(,15) Call +12 entry point for track conversion
- BAS 14,12(,15) Call +12 entry point for track conversion

IBM does not provide a macro for this conversion routine's invocations.
The conversion routine will return either TTR0 (if entered at CVTPRLTV+0) or TTTR (if entered at CVTPRLTV+12), where TTR0 and TTTR are as described in Table 44 on page 233.

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:

Table 46. Registers and Their Use for Converting Actual to Relative

<table>
<thead>
<tr>
<th>Register</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Loaded with the resulting TTR0 or TTTR to be passed back to the caller. These two formats are as described for register 0 in Table 44 on page 233.</td>
</tr>
<tr>
<td>1</td>
<td>Must be loaded with the address of the data extent block of the data set. Each DEB resides below 16 MB. When you call the +0 entry point, the called routine clears the high order byte of this register. When you call the +12 entry point, this byte must contain X'00'.</td>
</tr>
<tr>
<td>2</td>
<td>Must be loaded with the address of an 8-byte area containing the actual address to be converted to a TTR0 or TTTR. The actual address is of the form MBBCCHHR.</td>
</tr>
<tr>
<td>3–8</td>
<td>Not used by the conversion routine.</td>
</tr>
<tr>
<td>9–13</td>
<td>Used by the conversion routine and not restored.</td>
</tr>
<tr>
<td>14</td>
<td>Must be loaded with the address to which control is to be returned after execution of the conversion routine.</td>
</tr>
<tr>
<td>15</td>
<td>Used by the conversion routine as a base register and must be loaded with the content of field CVTPRLTV of the CVT.</td>
</tr>
</tbody>
</table>

Return Codes from the Conversion Routine

When control is returned to your program, register 15 contains one of the following return codes:

Table 47. Actual to Relative Conversion Routine Return Codes

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (X'00')</td>
<td>Successful conversion.</td>
</tr>
<tr>
<td>4 (X'04')</td>
<td>CCHH is outside of extent M. Returned relative track number is invalid. If you called the +12 entry point, then it has set the output R byte to X'FE'.</td>
</tr>
<tr>
<td>8 (X'08')</td>
<td>Internal access method control blocks are invalid. Call IBM service if the control blocks should be valid.</td>
</tr>
<tr>
<td>12 (X'0C')</td>
<td>Internal access method control blocks are invalid. Call IBM service if the control blocks should be valid.</td>
</tr>
<tr>
<td>16 (X'10')</td>
<td>Passed extent number M is too big for the data set. If you called the +12 entry point, then it has set the output R byte to X'FE'.</td>
</tr>
<tr>
<td>20 (X'14')</td>
<td>The calculated relative track number is too large to be returned. If you called the +0 entry, then the track number exceeds X'FFFF', so the routine returned X'FFFFFE00'. If you called the +12 entry point, then the track number exceeds X'FFFFFF', so the routine returned X'FFFFFFFE'. The likely causes of this are that the DEB is not valid or (for entry +0) the data set is a large format data set, but the track location is more than X'FFFF' tracks into the data set.</td>
</tr>
</tbody>
</table>
Note: For return codes 4, 16 and 20 with the +12 entry point, this routine returns a different value for the R byte (X'FE') than was passed to it.

---

**Using the IECTRKAD Callable Service or the TRKADDR Macro**

The IECTRKAD callable service or the TRKADDR macro can be used to perform the following operations on both 16-bit and 28-bit cylinder addresses:

- Calculate the relative track number on the volume
- Compare two track addresses
- Extract the 28-bit cylinder number
- Extract the 4-bit track number
- Increment the track address by one track and increment the cylinder number if necessary.
- Normalize cylinder number to permit comparing one *cchh* against another
- Convert a relative track number to a 28-bit cylinder address
- Set the cylinder number in a 28-bit track address
- Convert a normalized track address into an absolute 28-bit track address.

See "Call for converting and comparing 28-bit cylinder addresses (IECTRKAD)" on page 339 and "Perform calculations and conversions with 28-bit cylinder addresses (TRKADDR macro)" on page 317 for more information.

---

**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. The IBM DS8000® and newer storage subsystems accept the Set Sector command, but the command has no effect.

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.

**Note:**

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 data sets that are not DASD or do not have a UCB. Examples include tape, dummy data sets (DD DUMMY), spooled data sets (*, DATA or SYSOUT), TSO terminals and z/OS UNIX files such as zFS and HFS files. You will receive unpredictable results if you use the conversion routine with a data set or file that is not on DASD or does not have a UCB.

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:
### Table 48. Registers and Their Use for A Sector Convert Routine

<table>
<thead>
<tr>
<th>Register</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>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:</td>
</tr>
<tr>
<td></td>
<td>XX</td>
</tr>
<tr>
<td></td>
<td>K</td>
</tr>
<tr>
<td></td>
<td>R</td>
</tr>
<tr>
<td></td>
<td>To indicate fixed-length records, turn off (set to 0) the high-order bit of register 0.</td>
</tr>
<tr>
<td></td>
<td>Passing the track balance: Register 0 must contain a 4-byte value of the track balance of the record preceding the required record.</td>
</tr>
<tr>
<td></td>
<td>For all other cases: Register 0 must contain a 4-byte value in the form BBIR, where:</td>
</tr>
<tr>
<td></td>
<td>BB</td>
</tr>
<tr>
<td></td>
<td>I</td>
</tr>
<tr>
<td></td>
<td>R</td>
</tr>
<tr>
<td>1</td>
<td>Not used by the sector-convert routine.</td>
</tr>
<tr>
<td>2</td>
<td>When called at offset 0 or 8, must contain a 4-byte field where:</td>
</tr>
<tr>
<td></td>
<td>The first byte is the UCB device type code for the device (obtainable from UCB+19)</td>
</tr>
<tr>
<td></td>
<td>The remaining 3 bytes are the address of a 1-byte area that is to receive the sector value.</td>
</tr>
<tr>
<td></td>
<td>If called at offset 16 or 20, must contain the address of a 1-byte area that is to receive the sector value.</td>
</tr>
<tr>
<td>3-8</td>
<td>Not used.</td>
</tr>
<tr>
<td>9-10</td>
<td>Used by the convert routine and not saved or restored.</td>
</tr>
<tr>
<td>11</td>
<td>If called at offset 0 or 8, not used.</td>
</tr>
<tr>
<td></td>
<td>If called at offset 16 or 20, contains 1-byte UCBTBYT4 code in the low-order byte. The remaining three bytes must contain zero.</td>
</tr>
<tr>
<td>12,13</td>
<td>Not used.</td>
</tr>
<tr>
<td>14</td>
<td>Must be loaded with the address in which control is to be returned after execution of the sector conversion routine.</td>
</tr>
<tr>
<td>15</td>
<td>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).</td>
</tr>
</tbody>
</table>
Chapter 5. Using XDAP to Read and Write to Direct Access Devices

The execute direct access program (XDAP) is a macro instruction used to read, verify, or update a block on a direct access volume. This information covers the XDAP macro instruction, including information for compatibility with other IBM operating systems, what the XDAP macro does, and how to use it. It also discusses the macro instructions used with XDAP and the control blocks that are generated. However, IBM suggests using an access method such as VSAM in place of XDAP.

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 "IDAW Requirements for EXCP Requests" on page 177 of this publication. You should also be familiar with the information in z/OS DFSMS [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.

Restriction: XDAP does not support partitioned data set extended (PDSEs), extended-format data sets, UNIX files, UNIX system services 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.

XDAP requires less storage than do the standard access methods. However, XDAP 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.

All virtual addresses that you use with XDAP except the OPEN and CLOSE parameter lists, DCBE, and IOBE must be 24-bit.

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 you should use 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 "IDAW Requirements for EXCP Requests" on page 177.

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.

When the XDAP macro instruction is assembled, a control block and the following 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.

The parameters of the XDAP macro instruction are described in “Executing Direct Access Programs” on page 241.

Macro Instructions Used with XDAP

When using the XDAP macro instruction, you must also code DCB, OPEN, CLOSE and, in some cases, the EOV macro instructions. Special requirements or options for the other required macro instructions are explained in the following:

- “Defining a Data Control Block (DCB)”
- “Initializing a Data Control Block (OPEN)”
- “End of Volume (EOV)” on page 241
- “Restoring a Data Control Block (CLOSE)” on page 241

See “Data Control Block (DCB) Fields” on page 196 for listings of the parameters for DCB, OPEN, CLOSE and EOV.

Defining a Data Control Block (DCB)

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 196.

Initializing a Data Control Block (OPEN)

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 include the following actions:

- Constructing a data extent block
- Transferring 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 z/OS DFSMS Macro Instructions for Data Sets.

**End of Volume (EOV)**

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 “Handling End of Volume and End-Of-Data-Set Conditions” on page 194.

**Note:** Should EOV call DADSM EXTEND to extend on the same volume or a new volume (which implies DASD output), EXTEND will issue an enqueue on SYSVTOC. If the EOV request is issued for a data set on a volume where the application already holds the SYSVTOC enqueue, this will cause abnormal termination. For this case, a circumvention would be to allocate the output data set large enough to not require a secondary extent or request the output data set be on a volume that is different than the one for which it holds the SYSVTOC enqueue.

**Restoring a Data Control Block (CLOSE)**

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. When CLOSE is executed, some of the following procedures are performed:

- Releasing the data extent block
- Removing information that was 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 z/OS DFSMS 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:
Using XDAP

![XDAP event control block diagram]

**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:

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>Read a block.</td>
</tr>
<tr>
<td>W</td>
<td>Update a block.</td>
</tr>
<tr>
<td>V</td>
<td>Verify that the device is able to read the contents of a block, but do not transfer data.</td>
</tr>
<tr>
<td>I</td>
<td>Locate a block by identification. (The key portion, if present, and the data portion of the block are read, updated, or verified.)</td>
</tr>
<tr>
<td>K</td>
<td>Locate a block by key. (Only the data portion of the block is read, updated, or verified.) If you code this value, code the <code>key_addr,keylength_value</code> operands.</td>
</tr>
</tbody>
</table>

**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 that contains 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.

The track address of the block reference (CCHH) may contain 28-bit cylinder numbers for devices with more than 65,520 cylinders. Showing nibbles it is in the form of CCCCCcCH, where ccc represent bits 0-11 of the 28-bit cylinder number and CCCC represents bits 12-27 the 28-bit cylinder number. Use the TRKADDR macro to manipulate 16-bit and 28-bit cylinder numbers correctly.

(For more detailed information, see “Converting a Relative Track Address to an Actual Track Address” on page 232.)

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.

Exception: 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.

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.)

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.
Using XDAP

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 196 contains a diagram of the IOB (Figure 21 on page 212). 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 (ECB) Fields" on page 220 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 that is 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 Table 50.

<table>
<thead>
<tr>
<th>Type of I/O Operation</th>
<th>CCW</th>
<th>Command Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read by identification</td>
<td>1</td>
<td>Search ID Equal</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Transfer in Channel</td>
</tr>
<tr>
<td>Verify by identification</td>
<td>3</td>
<td>Read Key and Data</td>
</tr>
<tr>
<td>Read by key</td>
<td>1</td>
<td>Search Key Equal</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Transfer in Channel</td>
</tr>
<tr>
<td>Verify by key</td>
<td>3</td>
<td>Read Data</td>
</tr>
<tr>
<td>Write by identification</td>
<td>1</td>
<td>Search ID Equal</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Transfer in Channel</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>Write Key and Data</td>
</tr>
<tr>
<td>Write by key</td>
<td>1</td>
<td>Search Key Equal</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Transfer in Channel</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>Write Data</td>
</tr>
</tbody>
</table>

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 Table 50 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 UCBRFS 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. See "Obtaining the Sector Number of a Block on an RPS Device" on page 236.
Chapter 6. Using Password Protected Data Sets

This information covers 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 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 a SAF (system authorization facility)-compliant security product is active and provides protection for the data set, then the system bypasses password protection for that data set. Additionally, the system always bypasses password protection for VSAM and for SMS-managed data sets. The system provides SMS-managed data set and catalog protection through the SAF interface. For more SAF information, see “System Authorization Facility” in z/OS MVS Programming: Assembler Services Guide and z/OS MVS Programming: Assembler Services Reference ABE-HSP.

For information about VSAM data set protection, see z/OS DFSMS Using Data Sets and z/OS DFSMS Access Method Services Command.

The following are some reasons to use SAF instead of password protection:

- If you give a password to someone, you have no control over to whom they choose to give it.
- 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.
- Password protection is not supported on system-managed volumes or on dynamic devices.
Password Protection

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 information discusses only the first two methods. The last two methods are discussed in the publications shown in the following list.

Before using this information, you should be familiar with the contents of the following publications:

- z/OS DFSMS 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:
  - z/OS MVS System Messages, Vol 1 (ABA-AOM)
  - z/OS MVS System Messages, Vol 2 (ARC-ASA)
  - z/OS MVS System Messages, Vol 3 (ASB-BPX)
  - z/OS MVS System Messages, Vol 4 (CBD-DMO)
  - z/OS MVS System Messages, Vol 5 (EDG-GFS)
  - z/OS MVS System Messages, Vol 6 (GOS-IEA)
  - z/OS MVS System Messages, Vol 7 (IEB-IEE)
  - z/OS MVS System Messages, Vol 8 (IEF-IGD)
  - z/OS MVS System Messages, Vol 9 (IGF-IWM)
  - z/OS MVS System Messages, Vol 10 (IXC-IZP)
- z/OS 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.
- z/OS DFSMSdfp Utilities describes how to maintain the PASSWORD data set using the utility control statements of the IEHPROGM utility program.
- z/OS 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.

- 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.
- 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.
- 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 252. Using the IEHPROGM utility program to maintain the PASSWORD data set is described in [z/OS DFSMSdfp 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.

**Restriction:** The PASSWORD data set cannot be large format or extended format.

**Attention:** 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. 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. You can create a data set and set the protection indicator 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 z/OS 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 z/OS DFSMS 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 PWRITE protection-mode indicator.

Protection Feature Operating Characteristics

The topics that follow discuss the protection feature operating characteristics:

- “Terminating the Protection Feature Process” on page 251
- “Password Protection When Switching Volumes” on page 251
Password Protection

- “Password Protection When Concatenating Data Sets”
- “Maintaining the Counter for Password Protection”

**Terminating the Protection Feature 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.

**Password Protection When 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. (An HDR1 label is written with the security indicator indicating password protection.)

**Password Protection When Concatenating Data Sets**

A password is requested for every protected data set that is involved in a concatenation of data sets.

**Password Protection 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 for Password Protection**

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:

<table>
<thead>
<tr>
<th>If control password is:</th>
<th>Secondary password must be:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. PWREAD/PWWRITE or PWREAD/NOWRITE</td>
<td>PWREAD/PWWRITE or PWREAD/NOWRITE</td>
</tr>
<tr>
<td>2. NOPWREAD/PWWRITE</td>
<td>NOPWREAD/PWWRITE</td>
</tr>
</tbody>
</table>
Password Protection

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:

```
label

PROTECT parameter_list_address
```

*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 27 on page 254 through Figure 30 on page 257 for the parameter lists and codes used to identify the functions.

*Requirement:* The parameter lists and the areas addressed by the list must reside below 16 MB virtual. PROTECT will fail the request if the actual UCB of the system residence volume is above 16 MB or if PROTECT tries to update the data set's DSCB and one of its UCBs is above the 16 MB line.
Password Protection

PROTECT Macro Parameter Lists

ADD Function:

<table>
<thead>
<tr>
<th>3</th>
<th>0 X'01'</th>
<th>13 Control password pointer</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>00 00 00</td>
<td>16 Number of volumes</td>
</tr>
<tr>
<td>----</td>
<td>----------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>5</td>
<td>00 00 00</td>
<td>17 Volume list pointer</td>
</tr>
<tr>
<td>----</td>
<td>----------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>8</td>
<td>00</td>
<td>20 Protection code</td>
</tr>
<tr>
<td>----</td>
<td>----------</td>
<td>------------------------</td>
</tr>
<tr>
<td>9</td>
<td>00 00 00</td>
<td>21 New password pointer</td>
</tr>
<tr>
<td>----</td>
<td>----------</td>
<td>------------------------</td>
</tr>
<tr>
<td>10</td>
<td>00</td>
<td>24 String length</td>
</tr>
<tr>
<td></td>
<td></td>
<td>25 String pointer</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 27. 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.
Password Protection

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:**

```
| 3 0 X'02' | 3 13 Control password pointer |
| 3 1 00 00 00 | 3 16 Number of volumes |
| 3 4 Data set name length | 3 17 Volume list pointer |
| 3 5 Data set name pointer | 3 20 Protection code |
| 3 8 00 | 3 21 New password pointer |
| 3 9 Current password pointer | 3 24 String length |
| 3 12 00 | 3 25 String pointer |
```

Figure 28. 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
### Password Protection

A 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:

```
<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>8</td>
<td>12</td>
<td>16</td>
<td>17</td>
<td></td>
</tr>
</tbody>
</table>

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.

---

*Figure 29. Parameter List for DELETE Function*
Password Protection

LIST Function:

| 3 0 X'04' | 3 5 Data set name pointer |
| 3 1 80-byte buffer pointer | 3 8 00 |
| 3 4 Data set name length | 3 9 Current password pointer |

Figure 30. 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:

Table 51. PROTECT Return Codes

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (X'00')</td>
<td>Updating of the PASSWORD data set completed successfully.</td>
</tr>
<tr>
<td>4 (X'04')</td>
<td>The password of the data set name was already in the PASSWORD data set.</td>
</tr>
<tr>
<td>8 (X'08')</td>
<td>The password of the data set name was not in the PASSWORD data set.</td>
</tr>
<tr>
<td>12 (X'0C')</td>
<td>A control password is required or the one supplied is incorrect.</td>
</tr>
<tr>
<td>16 (X'10')</td>
<td>The supplied parameter list was incomplete or incorrect.</td>
</tr>
<tr>
<td>20 (X'14')</td>
<td>There was an I/O error in the PASSWORD data set or the system residence volume (which contains the PASSWORD data set), has an actual UCB that resides above the 16 MB line.</td>
</tr>
<tr>
<td>24 (X'18')</td>
<td>The PASSWORD data set was full.</td>
</tr>
<tr>
<td>28 (X'1C')</td>
<td>The validity check of the buffer address failed.</td>
</tr>
<tr>
<td>32 (X'20')</td>
<td>The LOCATE macro failed. LOCATE's return code is in register 1, and the number of indexes searched is in register 0.</td>
</tr>
<tr>
<td>36 (X'24')</td>
<td>The OBTAIN macro for the user data set failed. OBTAIN's return code is in register 1. The user data set resides on a volume that has an actual UCB that resides above the 16 MB line. Register 1 contains a 4, which simulates an OBTAIN return code.</td>
</tr>
<tr>
<td>40 (X'28')</td>
<td>The DSCB could not be updated.</td>
</tr>
<tr>
<td>44 (X'2C')</td>
<td>The PASSWORD data set does not exist.</td>
</tr>
<tr>
<td>48 (X'30')</td>
<td>Tape data set cannot be protected.</td>
</tr>
<tr>
<td>52 (X'32')</td>
<td>Data set in use.</td>
</tr>
<tr>
<td>56 (X'38')</td>
<td>The data set uses the virtual storage access method (VSAM).</td>
</tr>
</tbody>
</table>
Password Protection

Table 51. PROTECT Return Codes (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>60 (X'3C')</td>
<td>The data set is cataloged as being on more than 20 volumes. The DSCBs on the first 20 volumes have been updated. PROTECT does not support updating the rest.</td>
</tr>
</tbody>
</table>

Note:
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 information covers 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).
- Perform calculations and conversions with 28-bit cylinder addresses (TRKADDR macro).

Some functions of these macros tend to depend on the internal logic of the system.

Before reading this information, you should be familiar with the publications *High Level Assembler/MVS & VM & VSE Language Reference*. They contain the information necessary to code programs in the assembler language.

Ensuring Data Security by Validating the Data Extent Block (DEBCHK macro)

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 or 31-bit addressing mode when you call the DEBCHK macro.

The DEB fields used for EXCP and EXCPVR are illustrated in [Appendix A, “Control Blocks,” on page 443](#) (all the DEB fields are illustrated in [z/OS DFSMSdfp Diagnosis](#)).
DEBCHK Macro Specification

The format of the DEBCHK macro is:

```
  label  DEBCHK cbaddr
       ,TYPE= VERIFY,TYPE= ADD
       ,DELETE,TYPE= PURGE
       ,PURGE,PURGE=FORCE
       ,AM= amtype
       (amaddr)
       ((amreg))
       ,BRANCH= NO
       ,TCBADDR= address
       ,KEYADDR= address
       ,KEYREG= reg
       ,SAVREG= 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.

**Recommendation:** 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 or PURGE,PURGE=FORCE**

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 DEBDDBAD 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.

**PURGE,PURGE=FORCE**

DEBCHK removes the DEB pointer from the DEB table without checking the DCB (or ACB). The caller must be in system key, supervisor state, hold the local lock, and the passed DEB pointer must exist in the DEB table but not represent a valid DEB.

**AM**

Specifies an access method value. Each value corresponds to a particular access method type (note that BPAM and SAM have the same values):

<table>
<thead>
<tr>
<th>Value</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>(X'00')</td>
<td>NONE</td>
</tr>
<tr>
<td>(X'01')</td>
<td>VSAM</td>
</tr>
<tr>
<td>(X'02')</td>
<td>EXCP</td>
</tr>
<tr>
<td>(X'04')</td>
<td>Not supported</td>
</tr>
<tr>
<td>(X'08')</td>
<td>GAM</td>
</tr>
<tr>
<td>(X'10')</td>
<td>TAM</td>
</tr>
<tr>
<td>(X'20')</td>
<td>BPAM</td>
</tr>
<tr>
<td>(X'20')</td>
<td>SAM</td>
</tr>
<tr>
<td>(X'40')</td>
<td>BDAM</td>
</tr>
<tr>
<td>(X'81')</td>
<td>SUBSYS</td>
</tr>
<tr>
<td>(X'84')</td>
<td>Not supported</td>
</tr>
</tbody>
</table>

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: BDAM, SAM, BPAM, TAM (which refers to BTAM only), GAM, EXCP, or VSAM. 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)**

The RS-type address of the access method value. This format cannot be coded when MF=L is used.

**((amreg))**

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.
System Macros

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**

`SAVREG`  
`TCBADDR`  
`KEYADDR`  
`KEYREG`

Specifies branch entry is not to be used. The program ignores operands SAVREG, TCBADDR, KEYADDR, and KEYREG. Your program must be running in TCB mode.

**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:

- Your program can run in TCB or SRB mode.
- 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=`.
- 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 263 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 word 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 that points to the location, of a byte containing the key to be used when accessing the DCB (or ACB). The protection key is in bits 0 to 3. 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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (X'00')</td>
<td>The requested function completed successfully and register 1 contains the address of the DEB.</td>
</tr>
<tr>
<td>4 (X'04')</td>
<td>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.</td>
</tr>
<tr>
<td>8 (X'08')</td>
<td>An invalid TYPE was specified. (The DEBCHK routine was entered by a branch, not by the macro.)</td>
</tr>
<tr>
<td>12 (X'0C')</td>
<td>Your program was not authorized and TYPE was not VERIFY.</td>
</tr>
<tr>
<td>16 (X'10')</td>
<td>DEBDCBAD did not contain the address of the DCB (or ACB) that was passed to the DEBCHK routine.</td>
</tr>
<tr>
<td>20 (X'14')</td>
<td>The AM value does not equal the value in the DEBAMTYP field.</td>
</tr>
<tr>
<td>24 (X'18')</td>
<td>The DEB is not on the DEB chain and TYPE=ADD was specified.</td>
</tr>
<tr>
<td>28 (X'1C')</td>
<td>TYPE=ADD was specified for a DEB that was already entered in the DEB table.</td>
</tr>
<tr>
<td>32 (X'20')</td>
<td>The DEB table exceeded the maximum size and TYPE=ADD.</td>
</tr>
<tr>
<td>36 (X'24')</td>
<td>TYPE=PURGE, PURGE=FORCE was specified but one or more of the required conditions was not satisfied. The caller must be in system key, supervisor state, hold the local lock, and the passed DEB pointer must exist in the DEB table but not represent a valid DEB.</td>
</tr>
</tbody>
</table>

**Obtaining I/O Device Characteristics (DEVTYPE macro)**

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 - see "IHADVA Mapping macro" on page 281.

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 16 MB line.

Table 58 on page 277 shows the output for each device type that results from issuing the DEVTYPE macro without the INFOLIST parameter.
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 (TRKCALC macro)” on page 307.

DEVTYPE Macro Specification

There are four forms of DEVTYPE macro invocation covered here. They are the standard form, execute form, list form, and INFO form.

Restrictions:

- 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 document on any release of DFSMS.
- 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.
- 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 but it will not run on MVS/DFP Version 3. If you code PLISTVER=0, then you can assemble the program on DFSMS 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 FAREAELS During Assembler Macro Phase” on page 323.

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. 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:

`/SM590000/SM590000` DEVTYPE `label ddloc,area_addr,DEVTAB /SM590000/SM630000`

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:

```
label  DEVTYPE ddloc
       UCBLIST=(ucbl_addr,ucbl_num)
       INFOLIST=codel_addr
       PLISTVER=0
```

**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_addr—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. If you code `(area_addr,area_size)` before all the keywords, then code it after one comma. If you code `(area_addr,area_size)` after a keyword, then code `(area_addr,area_size)` after two commas.

**area_size—absolute expression or (2-12)**

The size (in bytes) of the area into which the device information is to be placed. See [Table 52](#).

<table>
<thead>
<tr>
<th>ddloc specified</th>
<th>UCBLIST specified</th>
<th>INFOLIST specified</th>
<th>Minimum size of area</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>No</td>
<td>Omitted or 0</td>
<td>8, 20, or 24 bytes depending on whether DEVTAB and RPS are coded.</td>
</tr>
</tbody>
</table>

Table 52. Minimum size of area
Table 52. Minimum size of area (continued)

<table>
<thead>
<tr>
<th>dloc specified</th>
<th>UCBLIST specified</th>
<th>INFOLIST specified</th>
<th>Minimum size of area</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Sum of the number of bytes returned for each code specified with INFO. See also the INFO keyword.</td>
</tr>
<tr>
<td>No</td>
<td>Yes</td>
<td>Omitted or 0</td>
<td>24 bytes per UCB</td>
</tr>
<tr>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>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.</td>
</tr>
</tbody>
</table>

**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 dloc. 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 and 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.

The DEVTYPE macro will accept a captured UCB or an actual UCB address above or below the 16 MB 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 16 MB line. Unauthorized programs can get a copy of the UCB by using the UCBCSCAN macro and specifying the COPY and UCBAREA keywords. Refer to [z/OS HCD Planning](#) for details.
ucbl_addr—A-type address or (2-12)
Name of an area containing a list of 4-byte UCB addresses.

ucbl_num—absolute expression or (2-12)
Number of 4-byte UCB address entries in the list.

BELOW
The UCB parameter contains addresses of UCBs that reside in storage below 16 MB, or a captured UCB. This is the default. 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. 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 "DEVTYPE" on page 272).

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, which was released in 1996. It can run on any system level that supports the coded parameters. You cannot code this with the INFOLIST or INFO parameter. The parameter list will be 20 bytes long.

1 The program can be assembled and run only on DFSMS/MVS Release 3 or later. The parameter list will not be acceptable for systems 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:
ddloc—RX-type address or (1-12)
Has the same meaning as the standard form of the macro (see also the
UCBLIST parameter below for execute form requirements of the ddloc
parameter).

area_addr—RX-type address or (2-12)
Has the same meaning as the standard form of the macro. This parameter must
be coded on the list and/or the execute form.
This must be coded on the execute form.

area_size—RX-type address or (2-12)
This form has the same meaning as the standard form of the macro. This
parameter must be coded on the list and/or on the execute form.
This must be coded on the execute form.

UCBLIST=(ucbl_addr,ucbl_num,BELLOW or ANY)
This format has the same meaning as the standard form of the macro. You
must specify either the UCBLIST or the ddloc parameter on either the execute
or list form. You can code them on both forms. Either parameter on the execute
form overrides the same parameter on the list form.

ucbl_addr—RX-type address or (2-12)
This format has the same meaning as the standard form of the macro.

ucbl_num—absolute expression or (2-12)
This format has the same meaning as the standard form of the macro
except that in the execute form, the maximum value of an absolute
expression is 4095. You can supply a larger value in a register.

BELOW
This format 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. This is the
default.

ANY
This format 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_addr
This format 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.
System Macros

codel_addr—RX-type address or (2-12)
This format has the same meaning as the standard form of the macro.

PLISTVER=0 or 1 or MAX
This format 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:

```
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
This format has the same meaning as the standard form of the macro.

```
UCBLIST=YES or (ucbl_addr, ucbl_num, BELOW or ANY)
```
This format 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
This format has the same meaning as the standard form of the macro.

```
ucbl_num
```
Absolute expression.
This format has the same meaning as the standard form of the macro.

```
BELOW
```
This format has the same meaning as the standard form of the macro. This is the default.

```
ANY
```
This format 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
This format 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
This format 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
This format has the same meaning as the standard form of the macro.

codel_addr—A-type address or (2-12)
This format has the same meaning as the standard form of the macro.

PLISTVER=0 or 1 or MAX
This format 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:

```
/SM590000/SM590000
DEVTYPE
label
/SM590000
INFO=( )
```

INFO=(AMCAP or 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.

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.

**AMCAP**

Returns 32 bytes for the access method capacity as follows:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0(0)</td>
<td>1</td>
<td>Flags.</td>
</tr>
<tr>
<td>0(0)</td>
<td>1...</td>
<td>BSAM, QSAM, and (if DASD) BPAM support the large block interface and the block size limit is in the next doubleword.</td>
</tr>
<tr>
<td>1(1)</td>
<td>7</td>
<td>Reserved, currently set to zeros.</td>
</tr>
<tr>
<td>8(8)</td>
<td>8</td>
<td>Maximum block size supported. If you specify a DD name to DEVTYPE for a data set concatenation, this value is the largest for any of the DDs. This value might exceed 32760 for a magnetic tape or dummy data set and therefore require EXCP or the access method large block interface. On output, OPEN does not allow a block size that exceeds this value except with EXCP. On certain cartridge tape drives, exceeding this limit can cause bypassing of hardware buffering. In the future, IBM might support values that exceed 32760 for other device types.</td>
</tr>
<tr>
<td>16(10)</td>
<td>8</td>
<td>Recommended maximum block size. This is less than or equal to the maximum block size supported. Above this length the device might be less efficient or less reliable. If you specify a DD name to DEVTYPE for a data set concatenation, this value is the largest for any of the DDs (refer to Table 54). Consult hardware documentation for further information.</td>
</tr>
<tr>
<td>24(18)</td>
<td>8</td>
<td>Maximum unspanned logical record length supported by BSAM, QSAM, or BPAM. Various types of data sets on the device might have various maximum record lengths. Therefore, if UCBLIST was coded on DEVTYPE and not a DD name, this value is the smallest for the possible data set types for BSAM, QSAM, and BPAM.</td>
</tr>
</tbody>
</table>

**Table 54. Optimum and Maximum Block Size Supported When Using EXCP or the Access Method Large Block Interface**

<table>
<thead>
<tr>
<th>Device Type</th>
<th>Optimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>DASD</td>
<td>Half track</td>
<td>32 760</td>
</tr>
</tbody>
</table>

Chapter 7. Using System Macro Instructions  271
### System Macros

**Table 54. Optimum and Maximum Block Size Supported When Using EXCP or the Access Method Large Block Interface (continued)**

<table>
<thead>
<tr>
<th>Device Type</th>
<th>Optimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reel tape</td>
<td>32 760</td>
<td>32 760</td>
</tr>
<tr>
<td>3480, 3490</td>
<td>65 535</td>
<td>65 535</td>
</tr>
<tr>
<td>3490 Emulation (VTS)</td>
<td>262 144 (256 KB)</td>
<td>262 144 (256 KB)</td>
</tr>
<tr>
<td>3590</td>
<td>262 144 (256 KB) except on some older models on which it is 229 376 (224 KB)</td>
<td>262 144 (256 KB)</td>
</tr>
<tr>
<td>DUMMY</td>
<td>16</td>
<td>5 000 000</td>
</tr>
</tbody>
</table>

**DEVTYPE**

Returns a copy of the four-byte UCBTYP field of the UCB. See the description of word 0 in “Device Characteristics Information” on page 273.

**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. This means that the following commands are supported:
    - Define Extent (X'63') at the beginning of the channel program, except with VIO.
    - Locate Record (X'47')
    - Read Multiple Count, Key and Data (X'5E')
    - Write Count, Key and Data Next Track (X'9D')

    For VIO data sets, this bit is on because these commands are always supported. For a non-VIO DASD, this bit also means that the device supports the Define Extent command, but EXCP allows it only at the beginning of your channel program. See “DASD Channel Program Prefix CCW Commands” on page 187.

  - **.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.

  - **...1** Flag DVAIXVLD. Flags DVACYL (byte 8 bit 4) and DVAEADSCB (byte 8 bit 5), along with field DVAVIRSZ (bytes 14-15) valid and possibly zero.

  - **.... 1...** Flag DVACLMG. Cylinder-managed space exists on this volume and begins at DVALCYL (bytes 10-11) in multicylinder units of DVAMCU (byte 9). DVAEADSCB (byte 8 bit 5) is also set with this flag on. Valid when DVAIXVLD (byte 8 bit 3) is set.
**System Macros**

---

**Flag DVAEADSCB.** Extended attribute DSCBs, Format 8 and 9 DSCBs, are allowed on this volume. Valid when DVAIXVLD (byte 8 bit 3) is set.

**Flag DVASSDEV.** The device is solid state.

**Flag DVACRYPT.** Data encrypted device.

**Byte 9** Field DVAMCU. Minimum allocation size in cylinders for cylinder-managed space. Each extent in this space must be a multiple of this value. Also referred to as the multicylinder unit (MCU). This is the smallest unit of disk space in cylinders that can be allocated in cylinder managed space. Valid when DVACYLMG (byte 8 bit 4) is set. This field is zero on releases before z/OS 1.10 or if the status is not yet known. In these two cases DVAIXVLD (byte 8 bit 3) is not set.

**Bytes 10-11** Field DVALCYL. First cylinder address divided by 4095 where space is managed in multicylinder units. Valid when DVACYLMG (byte 8 bit 4) is set. When valid and zero the volume has no cylinder-managed space. This field is zero on releases before z/OS 1.10 or if the status is not yet known. In these two cases DVAIXVLD (byte 8 bit 3) is not set.

**Byte 12** Track set size. Zero if device does not support read-any or write-any. See 3990 Reference for more information.

**Bytes 13** Reserved. DEVTYPE currently returns zeroes but could return something different in a future release.

**Bytes 14-15** Block size of the index data set. Valid when byte 8 bit 3 is on. When valid and zero the volume has no working VTOC index. This field is zero on releases before z/OS 1.10 or if the status is not yet known. In these cases byte 8 bit 3 is not set.

**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.

**Word 0:** Describes the device as defined in the UCBTYP field of the UCB. The IHADVA macro maps these four bytes this way:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>DVAOPTS</td>
<td>Model and option bits that depend on the device</td>
</tr>
</tbody>
</table>
System Macros

Table 55. Simulated Device Characteristics Information (continued)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>DVACLASS</td>
<td>Device class. Exactly one bit is on except that X'41' means a channel-to-channel adapter.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>X'80'=magnetic tape, X'40'=unit record, X'20'=DASD, X'10'=display and X'08'=character reader.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>A value of X'01' indicates a simulated device that does not have a UCB. These meanings are described in Table 56.</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>DVAUNIT</td>
<td>Device type. Depends on the device class.</td>
</tr>
</tbody>
</table>

Simulated Device Characteristics:

Any reference to one of the following types in Table 56 causes the information shown to be placed in word 0 and word 1 of the output area.

Table 56. Simulated Device Characteristics Information

<table>
<thead>
<tr>
<th>Data Set Type</th>
<th>Word 0 in Hexadecimal</th>
<th>Word 1 in Hexadecimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>DUMMY application process queue</td>
<td>0000 0000</td>
<td>0000 0000</td>
</tr>
<tr>
<td>TSO terminal</td>
<td>0000 0101</td>
<td>0000 7FF8</td>
</tr>
<tr>
<td>SYSIN, SYSOUT, or subsystem (SUBSYS=)</td>
<td>0000 0102</td>
<td>0000 7FF8</td>
</tr>
<tr>
<td>UNIX system services (possibly HFS) file</td>
<td>0000 0103</td>
<td>0000 7FF8</td>
</tr>
</tbody>
</table>

The system also uses a copy of the UCBTYP word in these places:

- In catalog entries for DASD and tape data sets. Some of the model and option bits are zero. See output of the IDCAMS LISTCAT command.
- As the device code returned by the LOCATE macro in the volume list. See "Retrieving Information from a Catalog" on page 157. Some of the model and option bits are zero.
- Input to the SCRATCH macro. See "Deleting a Data Set from the VTOC" on page 145 and "SCRATCH and CAMLST Macro Specification" on page 147.
- Input to the RENAME macro. See "Renaming a Data Set in the VTOC" on page 150 and "RENAME and CAMLST Macro Specification" on page 153.

Word 1

Maximum block size without using the large block interface of the access method. The maximum value is 32760 bytes. 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 Table 57 on page 276.

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. Treat this as an unassigned 16-bit number.

Recommendation: 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
System Macros

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.

**Recommendation:** 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. If the number of cylinders on
the volume exceeds 65520, then this bit is on. To retrieve
the number of cylinders for any DASD, you can use the
INFO=DASD operand of the DEVTYPE macro.

**Bit 1**
If on, ECKD supported. This means that the following
commands are supported:
- Define Extent (X'63') at the beginning of the channel
  program, except with VIO.
- Locate Record (X'47')
- Read Multiple Count, Key and Data (X'5E')
- Write Count, Key and Data Next Track (X'9D')

For VIO data sets, this bit is on because these commands
are always supported. For a non-VIO DASD, this bit also
means that the device supports the Define Extent
command, but EXCP allows it only at the beginning of
your channel program. See "DASD Channel Program
Prefix CCW Commands" on page 187.
System Macros

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 in the following order:

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

Table 57 and Table 58 on page 277 show the output for each device type that results from issuing the DEVTYPE macro.

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.

Table 57. Output from DEVTYPE Macro

<table>
<thead>
<tr>
<th>IBM Device</th>
<th>Maximum Access Method Record Size When Not Using Large Block Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>2540 Reader</td>
<td>80</td>
</tr>
<tr>
<td>2540 Punch</td>
<td>80</td>
</tr>
<tr>
<td>2501 Reader</td>
<td>80</td>
</tr>
<tr>
<td>3890 Document Processor</td>
<td>80</td>
</tr>
<tr>
<td>3505 Reader</td>
<td>80</td>
</tr>
</tbody>
</table>
Table 57. Output from DEVTYPE Macro (continued)

<table>
<thead>
<tr>
<th>IBM Device</th>
<th>Maximum Access Method Record Size When Not Using Large Block Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>3525 Punch</td>
<td>80</td>
</tr>
<tr>
<td>1403 Printer</td>
<td>120 ^1</td>
</tr>
<tr>
<td>3203 Model 5 Printer</td>
<td>132</td>
</tr>
<tr>
<td>3211 Printer</td>
<td>132</td>
</tr>
<tr>
<td>3262 Model 5 Printer</td>
<td>132</td>
</tr>
<tr>
<td>4245 Printer</td>
<td>132</td>
</tr>
<tr>
<td>4248 Printer</td>
<td>132</td>
</tr>
<tr>
<td>3800 or 3900 Printing Subsystem</td>
<td>136 ^3</td>
</tr>
<tr>
<td>3410, 3420, 3422, 3424, 3430, 3480, 3490, 3590</td>
<td>32760</td>
</tr>
</tbody>
</table>

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.

Table 58. Output from DEVTYPE Macro — DASD Devices

<table>
<thead>
<tr>
<th>IBM Device</th>
<th>Maximum Record Size (Word 1, Decimal)</th>
<th>DEVTAB (Words 2, 3, and 4, in Hexadecimal)</th>
<th>RPS (Word 5, in Hexadecimal)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3380 Models AD4, AJ4, BD4, BJ4, and CJ2 Disk Storage</td>
<td>32 760</td>
<td>0376 000F BB60 0100 2010 010B</td>
<td>04E0 DED6</td>
</tr>
<tr>
<td>3380 Models AD4, AJ4, BD4, BJ4, Disk Storage (attached to a cache storage control)</td>
<td>32 760</td>
<td>0376 000F BB60 0100 2030 010B</td>
<td>04E0 DED6</td>
</tr>
<tr>
<td>3380 Models AE4 and BE4 Disk Storage</td>
<td>32 760</td>
<td>06EB 000F BB60 0100 2010 010B</td>
<td>04E0 DED6</td>
</tr>
<tr>
<td>3380 Models AE4 and BE4 Disk Storage (attached to a cache storage control)</td>
<td>32 760</td>
<td>06EB 000F BB60 0100 2030 010B</td>
<td>04E0 DED6</td>
</tr>
<tr>
<td>3380 Models AK4 and BK4 Disk Storage</td>
<td>32 760</td>
<td>0A60 000F BB60 0100 2010 010B</td>
<td>04E0 DED6</td>
</tr>
<tr>
<td>3380 Models AK4 and BK4 Disk Storage (attached to a cache storage control)</td>
<td>32 760</td>
<td>0A60 000F BB60 0100 2030 010B</td>
<td>04E0 DED6</td>
</tr>
<tr>
<td>3390 Model 1 (attached to a 3990 Model 2)</td>
<td>32 760</td>
<td>0459 000F E5A2 0000 0052 0000</td>
<td>0594 E000</td>
</tr>
<tr>
<td>3390 Model 1 (attached to a 3990 Model 3)</td>
<td>32 760</td>
<td>0459 000F E5A2 0000 0072 0000</td>
<td>0594 E000</td>
</tr>
<tr>
<td>3390 Model 2 (attached to a 3990 Model 2)</td>
<td>32 760</td>
<td>08B2 000F E5A2 0000 0052 0000</td>
<td>0594 E000</td>
</tr>
<tr>
<td>3390 Model 2 (attached to 3990 Model 3)</td>
<td>32 760</td>
<td>08B2 000F E5A2 0000 0072 0000</td>
<td>0594 E000</td>
</tr>
<tr>
<td>3390 Model 3 (attached to a 3990 Model 2)</td>
<td>32 760</td>
<td>0D0B 000F E5A2 0000 0052 0000</td>
<td>0594 E000</td>
</tr>
<tr>
<td>3390 Model 3 (attached to a 3990 Model 3)</td>
<td>32 760</td>
<td>0D0B 000F E5A2 0000 0072 0000</td>
<td>0594 E000</td>
</tr>
<tr>
<td>3390 Model 3 (attached to a 3990 Model 6)</td>
<td>32 760</td>
<td>0D0B 000F E5A2 0000 0072 0000</td>
<td>0594 E000</td>
</tr>
<tr>
<td>3390 Model 9 (attached to a 3990 Model 2)</td>
<td>32 760</td>
<td>2721 000F E5A2 0000 0052 0000</td>
<td>0594 E000</td>
</tr>
<tr>
<td>3390 Model 9 (attached to a 3990 Model 3)</td>
<td>32 760</td>
<td>2721 000F E5A2 0000 0052 0000</td>
<td>0594 E000</td>
</tr>
<tr>
<td>3390 Model 9 (attached to a 3990 Model 6)</td>
<td>32 760</td>
<td>2721 000F E5A2 0000 0052 0000</td>
<td>0594 E000</td>
</tr>
<tr>
<td>3390 Model 9 (attached to a 3990 Model 3)</td>
<td>32 760</td>
<td>2721 000F E5A2 0000 0052 0000</td>
<td>0594 E000</td>
</tr>
<tr>
<td>9345 Model 1</td>
<td>32 760</td>
<td>05A0 000F BC98 0000 0052 0000</td>
<td>04A0 D500</td>
</tr>
</tbody>
</table>

**Recommendation:** For all currently supported devices, DEVTYPE does not return enough information to perform space calculations. Use the TRKCALC macro and
the sector conversion routine to perform space calculations. For information on using the TRKCALC macro, see “Performing Track Calculations (TRKCALC macro)” on page 307. For information on the sector conversion routine, see “Obtaining the Sector Number of a Block on an RPS Device” on page 236.

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. Registers 2 to 14 contents are unchanged. Register 1 contents are unpredictable. The return codes and their meanings are as follows:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (X'00')</td>
<td>Information has been successfully stored in your work area.</td>
</tr>
</tbody>
</table>

**Reason Code**

<table>
<thead>
<tr>
<th>Reason Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (X'00')</td>
<td>All the information is available.</td>
</tr>
<tr>
<td>4 (X'04')</td>
<td>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.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 (X'04')</td>
<td>Invocation error.</td>
</tr>
</tbody>
</table>

**Reason Code**

<table>
<thead>
<tr>
<th>Reason Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 (X'04')</td>
<td>DD name not defined.</td>
</tr>
<tr>
<td>8 (X'08')</td>
<td>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 coded for the third value of the UCBLIST parameter.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 (X'08')</td>
<td>Unsupported device class.</td>
</tr>
</tbody>
</table>

**Reason Code**

<table>
<thead>
<tr>
<th>Reason Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>12 (X'0C')</td>
<td>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.</td>
</tr>
</tbody>
</table>

DEVTYPE—Example 1—Referring to a DD Statement

```assembly
DEVTYPE MYDD,DEVINFO,DEVTAB
  
  MYDD DC CL8'DATATAB'
  DEVINFO DC 5F'0'
```

System Macros
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**

```assembly
classic
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

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 OF ALIGNMENT FOR TRKCYL
AREA DS OCL20 INFORMATION FROM
       DS CL4 DEVTYP INFO=(DASD,DEVTYPE)
       DS OCL4 INFORMATION FROM
       DS CL4 DEVTYP INFO=DEVTYPE (UCBTYP)
       DS CL4 UCBTYP FROM UCB POINTED TO

*************************************************************************

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 - DEVTYPE clears the extra 16 bytes. 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 position 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 38 on page 324.

**DEVTYPE—Example 3—Building a Parameter List and Using IHADVA**

This example is the same as the previous one, but it uses the IHADVA macro to map the DEVTYPE output and several unusual techniques to demonstrate the possibilities.

```plaintext
MVC DTLIST,KDTLIST Build parameter list in dynamic storage
***********************************************************************
* Retrieve four-byte UCBTYP for SYSUT1 device in 20-byte area.
* **************************************************************************
DEVTYPE ,(DVAIDASD,L'DVAIDASD+LenDEVTYPE),MF=(E,DTLIST)
DTUsing USING DVAUCBTY,DVAIDASD Map DSECT on CSECT
 * *
*******************************************************************************
* 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)
DTUsing USING DVAUCBTY,DVAIDASD+L'DVAIDASD Map DSECT on CSECT
 * *
*******************************************************************************
KDTLIST DEVTYPE FIRSTDD,MF=L,INFOLIST=ILIST1 Read-only parameter list
LDTLIST EQU +-KDTLIST Length of parameter list
FIRSTDD DC CL8'SYSUT1' DD name
ILIST1 DEVTYPE INFO=DEVTYPE DEVTYPE INFO list
ILIST2 DEVTYPE INFO=(DASD,DEVTYPE) Another INFO list
* End of CSECT.
DYNAMIC DSECT **** Dynamic storage for reentrancy
UCBAD DS A Address of UCB
DTLIST DS CL(LDTLIST) DEVTYPE parameter list (modifiable)
 DS OF Alignment for efficiency
* Output from INFO=(DASD,DEVTYPE) begins here.
* Next line defines symbols for DVAIDASD, which is 16 bytes.
 IHADVA DSECT=NO,INFO=DASD Info from DEVTYPE INFO=DASD
 DS CL(LenDEVTYPE) Info from DEVTYPE INFO=DEVTYPE
* Next line defines the DVAUCBTY DSECT, which is 4 bytes.
 IHADVA DSECT=YES,INFO=DEVTYPE Info from DEVTYPE INFO=DEVTYPE
LenDEVTYPE EQU +-DVAUCBTY Length of DSECT (4 b
```

Example 3 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,(DVAIDASD,L'DVAIDASD+LenDEVTYPE),INFOLIST=ILIST1
```

ILIST1 describes four bytes to be returned at the beginning of a 20-byte area. The length attribute of the INFO=DASD area is L'DVAIDASD, which is 16. The constant LenDEVTYPE is the length of the INFO=DEVTYPE area, which is four bytes. DEVTYPE clears the extra 16 bytes. 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.

The DTUsing USING DVAUCBTY,DVAIDASD line is a labeled dependent USING statement. It applies the 4-byte DVAUCBTY DSECT on top of the storage at DVAIDASD. One purpose of the symbol DTUsing could be to allow a later DROP statement to end the addressability but that is not the purpose of the label here. The purpose here is to avoid an assembler warning message ASMA303W. Multiple address resolutions may result from this USING and the USING on statement number nn". It would be on the second labeled USING if it did not have the same label.
In effect, the second specification of DEVTYPE is:

```
DEVTYPE ,,(L’DVAIDASD+LenDEVTYPE),INFOLIST=ILIST2,UCBLIST=(UCBAD,1)
```

The INFOLIST describes 20 bytes to be returned. The second USING with the label DTUsing defines addressability to the last four of the 20 bytes. The first 16 bytes have no USING because they are a part of the DYNAMIC DSECT and already have addressability that is not shown. This technique of using one DSECT (defined by IHADVA DSECT=YES,INFO=DEVTYPE) apply to two places in different areas of the program is allowed by named USINGs without loading another base register.

The IHADVA DSECT=NO,INFO=DASD line defines 16 bytes of variables that DEVTYPE sets. The first symbol is DVAIDASD and its length attribute is 16.

The DS CL(LenDEVTYPE) line defines variables that are described by the DSECT generated by the IHADVA DSECT=YES,INFO=DEVTYPE line.

**IHADVA Mapping macro**

The IHADVA macro supports two parameters:

**DSECT={YES|NO}**

If you code DSECT=YES, you get a single area with a DSECT. This is the default. Its name depends on whether you code INFO= and what you code for it. The DSECT name depends on the first value that you code for INFO=.

The following applies if you code DSECT=NO:

- If you omit INFO= or you code INFO=NONE, then the area begins with the symbol DVAREA and it is not a DSECT.
- If you code any combination of INFO values other than NONE, then DVAREA is not defined and there is no DSECT.

**INFO={NONE|DEVTYPE|DASD|SUFFIX|AMCAP}**

If you omit the INFO keyword, then the mapping is for all of the following at the same origin:

- the minimum type of call
- UCBLIST= without INFO=
- INFO=DASD
- INFO=DEVTYPE
- INFO=SUFFIX

**INFO=NONE.**

This generates the mapping for the minimum type of call or when you code UCBLIST= without INFO=. You cannot code NONE in combination with any other value.

**INFO=DASD**

Generate the mapping for the area returned by coding INFO=DASD. The DSECT name or first symbol is DVAIDASD.

**INFO=DEVTYPE**

Generate the mapping for the area returned by coding INFO=DEVTYPE. The DSECT name or first symbol is DVAUCBTY.

**INFO=SUFFIX**

Generate the mapping for the area returned by coding INFO=SUFFIX. The DSECT name or first symbol is DVASUFFX.
INFO=AMCAP

Generate the mapping for the area returned by coding INFO=AMCAP. The DSECT name or first symbol is DVAAMCAP.

=======================================================================

DEVTYPE return area (mapping macro IHADVA)
THIS MACRO MAPS THE AREA RETURNED TO THE CALLER BY THE DEVTYPE SVC
=======================================================================

<table>
<thead>
<tr>
<th>OFFSET</th>
<th>DEC(HEX)</th>
<th>TYPE</th>
<th>LEN</th>
<th>NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (0)</td>
<td>STRUCTURE</td>
<td>24</td>
<td>DVAREA</td>
<td>Area if no INFOLIST=, DEVTAB or RPS.</td>
<td></td>
</tr>
<tr>
<td>0 (0)</td>
<td>CHARACTER</td>
<td>8</td>
<td>DVAPREFX</td>
<td>Following four bytes are also returned for INFO=DEVTYPE.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OFFSET</th>
<th>DEC(HEX)</th>
<th>TYPE</th>
<th>LEN</th>
<th>NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (0)</td>
<td>CHARACTER</td>
<td>4</td>
<td>DVAUCBTY</td>
<td>UCB TYPE FIELD.</td>
<td></td>
</tr>
<tr>
<td>0 (0)</td>
<td>BITSTRING</td>
<td>2</td>
<td>DVAOPTS</td>
<td>UCB OPTIONS.</td>
<td></td>
</tr>
<tr>
<td>2 (2)</td>
<td>BITSTRING</td>
<td>1</td>
<td>DVACLASS</td>
<td>DEVICE CLASS.</td>
<td></td>
</tr>
<tr>
<td>3 (3)</td>
<td>BITSTRING</td>
<td>1</td>
<td>DVAUNIT</td>
<td>UNIT TYPE.</td>
<td></td>
</tr>
<tr>
<td>4 (4)</td>
<td>SIGNED</td>
<td>4</td>
<td>DVAMAXRC</td>
<td>MAXIMUM RECORD SIZE.</td>
<td></td>
</tr>
<tr>
<td>8 (8)</td>
<td>CHARACTER</td>
<td>12</td>
<td>DVATAB</td>
<td>SECTION INCLUDED BY DEVTAB.</td>
<td></td>
</tr>
<tr>
<td>8 (8)</td>
<td>UNSIGNED</td>
<td>2</td>
<td>DVACYL</td>
<td>PHYS NUMBER CYL PER VOLUME.</td>
<td></td>
</tr>
<tr>
<td>10 (A)</td>
<td>SIGNED</td>
<td>2</td>
<td>DVATRK</td>
<td>NR OF TRACKS PER CYL.</td>
<td></td>
</tr>
<tr>
<td>12 (C)</td>
<td>SIGNED</td>
<td>2</td>
<td>DVATRKLN</td>
<td>TRACK LENGTH (BYTES).</td>
<td></td>
</tr>
<tr>
<td>14 (E)</td>
<td>SIGNED</td>
<td>2</td>
<td>DVAOVHD</td>
<td>BLOCK OVERHEAD IF DVA2BOV IS ON.</td>
<td></td>
</tr>
</tbody>
</table>

IF DVA2BOV IS OFF USE INSTEAD THE FOLLOWING TWO VALUES

<table>
<thead>
<tr>
<th>OFFSET</th>
<th>DEC(HEX)</th>
<th>TYPE</th>
<th>LEN</th>
<th>NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>14 (E)</td>
<td>ADDRESS</td>
<td>1</td>
<td>DVAOVNLB</td>
<td>OVERHEAD NOT LAST BLOCK.</td>
<td></td>
</tr>
<tr>
<td>15 (F)</td>
<td>ADDRESS</td>
<td>1</td>
<td>DVAOVLB</td>
<td>OVERHEAD LAST BLOCK.</td>
<td></td>
</tr>
<tr>
<td>16 (10)</td>
<td>ADDRESS</td>
<td>1</td>
<td>DVAOVNK</td>
<td>OVERHEAD DECR IF NOT KEYED.</td>
<td></td>
</tr>
<tr>
<td>17 (11)</td>
<td>BITSTRING</td>
<td>1</td>
<td>DVAFLAGS</td>
<td>FLAG BYTE.</td>
<td></td>
</tr>
<tr>
<td>1...</td>
<td>DVABCYL</td>
<td>IF 1, DVACYL IS INVALID.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.1...</td>
<td>DVADFLR</td>
<td>DEFINE EXTENT/LOCATE RECORD AND RELATED TRANSFER COMMANDS ARE IMPLEMENTED.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>..1...</td>
<td>DVADFX</td>
<td>DEFINE EXTENT IMPLEMENTED.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...1...</td>
<td>DVAOMODL</td>
<td>IF ON, USE MODULO TRACK ALGORITHM.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>....1...</td>
<td>DVA2BOV</td>
<td>IF ON, USE DVAOVHD ELSE USE DVAOVNLB &amp; DVAOVLB.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>....1...</td>
<td>DVAPAGES</td>
<td>IF ON DEVICE SUPPORTS PAGING CCWS.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>......1.</td>
<td>DVANOALT</td>
<td>NO ALT TRKS AVAILABLE.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>......1.</td>
<td>DVAFTOL</td>
<td>IF ON, APPLY TOLERANCE FACTOR.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18 (12)</td>
<td>SIGNED</td>
<td>2</td>
<td>DVATOL</td>
<td>TOLERANCE FACTOR.</td>
<td></td>
</tr>
</tbody>
</table>

(BLKSI+KEYLE) DVATOL/DVADVSR GIVES THE ADJUSTED BLOCK SIZE TO WHICH APPROPRIATE OVERHEADS ARE THEN ADDED. |

<table>
<thead>
<tr>
<th>OFFSET</th>
<th>DEC(HEX)</th>
<th>TYPE</th>
<th>LEN</th>
<th>NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>20 (14)</td>
<td>CHARACTER</td>
<td>4</td>
<td>DVARPS</td>
<td>RPS SECTION.</td>
<td></td>
</tr>
<tr>
<td>20 (14)</td>
<td>SIGNED</td>
<td>2</td>
<td>DVAOVRO</td>
<td>OVERHEAD BYTES FOR RECORD 0.</td>
<td></td>
</tr>
<tr>
<td>22 (16)</td>
<td>ADDRESS</td>
<td>1</td>
<td>DVASECT</td>
<td>NUMBER OF SECTORS IN FULL TRACK.</td>
<td></td>
</tr>
<tr>
<td>23 (17)</td>
<td>ADDRESS</td>
<td>1</td>
<td>DVASECTD</td>
<td>NUMBER OF DATA SECTORS.</td>
<td></td>
</tr>
</tbody>
</table>

THE FOLLOWING SECTION IS RETURNED BY DEVTYPE FOR INFO=DASD.

<table>
<thead>
<tr>
<th>OFFSET</th>
<th>DEC(HEX)</th>
<th>TYPE</th>
<th>LEN</th>
<th>NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (0)</td>
<td>STRUCTURE</td>
<td>16</td>
<td>DVAIDASD</td>
<td>NUMBER OF CYLINDERS.</td>
<td></td>
</tr>
<tr>
<td>0 (0)</td>
<td>UNSIGNED</td>
<td>4</td>
<td>DVAICYL</td>
<td>TRACKS PER CYLINDER.</td>
<td></td>
</tr>
<tr>
<td>4 (4)</td>
<td>UNSIGNED</td>
<td>4</td>
<td>DVAITRKLN</td>
<td>TRACKS PER CYLINDER.</td>
<td></td>
</tr>
<tr>
<td>8 (8)</td>
<td>UNSIGNED</td>
<td>1</td>
<td>DVAIFLAG</td>
<td>FLAGS.</td>
<td></td>
</tr>
<tr>
<td>Field</td>
<td>Type</td>
<td>Length</td>
<td>Description</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>--------</td>
<td>--------</td>
<td>-------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DVAECKD1</td>
<td>ECKD SUPPORTED, ALSO ON FOR VIO DATA SETS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DVALRE1</td>
<td>LOCATE RECORD EXTENDED SUPPORTED</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DVACACHE1</td>
<td>DEVICE IS Cached</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DVAIXVLD</td>
<td>DVACYLMG, DVAEADSCB, DVAVIRSZ valid.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DVACYLGM</td>
<td>Cylinder-managed space exists on this volume and begins at DVALCYL in multicylinder units of DVAMCU. DVAEADSCB is also set with this flag on. Valid when DVAIXVLD is set.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DVAEADSCB</td>
<td>Extended attribute DSCBs, Format 8 and 9 DSCBs, are allowed on this volume. Valid when DVAIXVLD is set.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DVASSDEV</td>
<td>The device is solid state</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DVACRYPT</td>
<td>Data encrypted device</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DVAMCU</td>
<td>Minimum allocation size in cylinders for cylinder-managed space. Each extent in this space must be a multiple of this value. Also referred to as the multicylinder unit (MCU). This is the smallest unit of disk space in cylinders that can be allocated in cylinder-managed space. Valid when DVACYLMG is set. This field is zero on releases before z/OS 1.10 or if the status is not yet known. In these two cases DVAIXVLD is not set.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DVALCYL</td>
<td>First cylinder address divided by 4095 where space is managed in multicylinder units. Cyl-managed space begins at this address. Valid when DVACYLMG is set. This field is zero on releases before z/OS 1.10 or if the status is not yet known. In these two cases DVAIXVLD is not set.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DVAITSET</td>
<td>TRACK SET SIZE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DVAVIRSZ</td>
<td>Block size of the index data set. Valid when DAVIXVLD is set on. When valid and zero the volume has no working VTOC index. This field is zero on releases before z/OS 1.10 or if the status is not yet known. In these cases DVAIXVLD is not set.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

THE FOLLOWING SECTION IS RETURNED BY DEVTYPE FOR INFO=AMCAP.

---

**System Macros**

1... DVAECKD1 ECKD SUPPORTED, ALSO ON FOR VIO DATA SETS
.1... DVALRE1 LOCATE RECORD EXTENDED SUPPORTED
.1... DVACACHE1 DEVICE IS CACHED
...1 DVAIXVLD DVACYLMG, DVAEADSCB, DVAVIRSZ valid.
.... DVACYLGM Cylinder-managed space exists on this volume and begins at DVALCYL in multicylinder units of DVAMCU. DVAEADSCB is also set with this flag on. Valid when DVAIXVLD is set.
.... DVAEADSCB Extended attribute DSCBs, Format 8 and 9 DSCBs, are allowed on this volume. Valid when DVAIXVLD is set.
.... DVASSDEV The device is solid state
.... DVACRYPT Data encrypted device.
9 (9) UNSIGNED 1 DVAMCU Minimum allocation size in cylinders for cylinder-managed space. Each extent in this space must be a multiple of this value. Also referred to as the multicylinder unit (MCU). This is the smallest unit of disk space in cylinders that can be allocated in cylinder-managed space. Valid when DVACYLMG is set. This field is zero on releases before z/OS 1.10 or if the status is not yet known. In these two cases DVAIXVLD is not set.
10 (A) UNSIGNED 2 DVALCYL First cylinder address divided by 4095 where space is managed in multicylinder units. Cyl-managed space begins at this address. Valid when DVACYLMG is set. This field is zero on releases before z/OS 1.10 or if the status is not yet known. In these two cases DVAIXVLD is not set.
12 (C) UNSIGNED 1 DVAITSET TRACK SET SIZE
13 (D) UNSIGNED 1 * Reserved. DEVTYPE currently returns zeroes but could return something different in a future release.
14 (E) UNSIGNED 2 DVAVIRSZ Block size of the index data set. Valid when DAVIXVLD is set on. When valid and zero the volume has no working VTOC index. This field is zero on releases before z/OS 1.10 or if the status is not yet known. In these cases DVAIXVLD is not set.
SUPPORT THE LARGE BLOCK INTERFACE & THE LIMIT IS IN THE NEXT DOUBLEWORD.

.1.... DVAAM_XTIOT  This data set allocation has an XTIOT. Either all or none of the entries for a concatenation are XTIOT.

..1.... DVAAM_XTIOTAM BSAM, QSAM and BPAM (if DASD) support XTIOT for this device, and the NON_VSAM_XTIOT option in PARMLIB allows it. DEVTYPE will turn this on if the UCB is DASD or tape or the DD is dummy and the PARMLIB option allows it.

....1  DVAAM_31UCB  One or more UCB addresses for this data set allocation (or concatenation) point above the 16 MB and have not been captured for the allocation. If this bit is off, the data set still might be extended to another volume and gain a 31-bit address UCB.

....1.  DVAAM_31UCBAM BSAM, QSAM and BPAM (if DASD) support 31-bit UCB addresses in the DEB and the NON_VSAM_XTIOT option in PARMLIB allows it.

....1..  DVAAM_DSAB  DSAB is above the line.

....1.1.  DVAAM_DSABAM BSAM, QSAM and BPAM (if DASD) support DSAB above the line and the NON_VSAM_XTIOT option in PARMLIB allows it.

1 (1) CHARACTER 7 * RESERVED
8 (8) BITSTRING 8 DVAMAXBLK  MAXIMUM BLOCK SIZE SUPPORTED WITH SAM LBI

16 (10) BITSTRING 8 DVAOPTBLK  RECOMMENDED MAXIMUM BLOCK SIZE LONGER BLOCKS MIGHT BE LESS EFFICIENT OR LESS RELIABLE. LESS THAN OR EQUAL TO PREVIOUS FIELD.

24 (18) BITSTRING 8 DVAMAXLR  MAXIMUM UNSPANNED LOGICAL RECORD LENGTH SUPPORTED BY BSAM, QSAM AND BPAM

====================================================================
THE FOLLOWING SECTION IS RETURNED BY DEVTYPE FOR INFO=SUFFIX.
====================================================================
0 (0) SIGNED 2 DVASUFFX  SUFFIX LENGTH

**Reading and Modifying a Job File Control Block (RDJFCB Macro)**

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). Some information is placed into the JFCB when the data set is allocated, while other information is placed there only when the data set is opened. Which fields are updated and when they are updated will vary depending upon factors such as what is specified in the JCL DD statement, what the application does, whether the data set is SMS managed or not, and so on. Fields that have not been updated yet will contain binary zeroes.

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, let's suppose that 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 288. 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.

If you specify the XTIOT, UCB NOCAPTURE or DSAB-above-the-line options of dynamic allocation, then the system creates an XTIOT. With these options, if the access method is not EXCP and the data set is not VSAM, then RDJFCB requires that you code the LOC=ANY option on the DCBE macro, and the NON_VSAM_XTIOT=YES option in the DEVSUPxx member of SYS1.PARMLIB is in effect.

The RDJFCB macro also allows you to retrieve allocation information 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.

You can retrieve the following items:
- All JFCBs
- All volume serial numbers
- Block size limit
- The path name (PATH=) associated with any DD. In these cases the data set name in the JFCB is a dummy value.

“Type 07 JFCB Exit List Entry” on page 290 describes how you can use RDJFCB to retrieve this information.

Tip: 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).

Your program can also use the SWAREQ macro to access JFCBs and JFCBXs. It requires your program to access the DSAB or DSABs and the TIOT or XTIOTs. You can use the GETDSAB macro for this and you will need several mapping macros. SWAREQ and GETDSAB are documented in “z/OS MVS Programming: Authorized System Macros”.
The RDJFCB macro is designed to be simpler to use and does not require examining system control blocks.

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
  The number-of-volumes field can be modified only to be a value not greater than the total number of volume serial numbers available in the JFCB and any JFCBXs (extensions). The JFCB can have five volume serial numbers. Each JFCBX can have 15 volume serial numbers. Whether or not a JFCBX is required and how many JFCBXs are required is determined during data set allocation. A JFCBX cannot be dynamically created after allocation. Therefore, the maximum value of the number-of-volumes field is based on the JFCB and how many JFCBXs exist. Setting the number-of-volumes field to a value greater than that maximum will not cause a JFCBX to be dynamically created.
- Moving the DCB fields from the DSCB into the JFCB
- Adding volume serial numbers to the JFCB (see "RDJFCB Security" on page 290)
  Volume serial numbers in excess of five are written to the JFCBX (extension). The JFCBX cannot be modified by user programs.
- Modifying the sequence number field of the volume in the JFCB. For DISP=NEW the modified volume sequence will be honored during OPEN TYPE=J only for tape and only when the file sequence number in the JFCB is also modified.
- Modifying the sequence number field of the data set in the JFCB. This specifies the file for a subsequent OPEN TYPE=J to process. You can use this technique to write many data sets on a tape volume or set of volumes. Use RDJFCB and increase the sequence number of the data set by 1 before each OPEN TYPE=J. If the device supports buffered tape marks, you can obtain significantly better performance by requesting the SYNC=NONE option on the DCBE macro. It allows the device to enter "streaming mode" and to write tape marks much faster. A side effect is that any I/O error can be reflected at an unpredictable time. This can result in an ABEND in OPEN or CLOSE for a user data block. It also means that if a device malfunction occurs, it might result in the loss of more than one data set that the application had closed previously. To position rapidly to a tape data set other than the next data set, you can use the technique described in "High-Speed Cartridge Tape Positioning" on page 301.
- Changing the data set name field or member field in the JFCB. See "RDJFCB Security" on page 290 and "RDJFCB Use by Authorized Programs" on page 291.
  You can open a VTOC by reading the JFCB, changing the data set name to 44 bytes of X'04' and then issuing the OPEN macro with TYPE=J. Use BSAM or EXCP. If you use BSAM, also code RECFM=F, KEYLEN=44 and BLKSIZE=96 on the DCB macro. For an extended address volume the DCB macro must point to a DCBE where the EADSCB=OK keyword is specified. You'll find examples of opening an EXCP DCB for a VTOC in "Example of Using the CVAFFILT Macro" on page 106 and "Example of using the CVAFSEQ macro to process a volume in physical sequential order" on page 128.
- Setting bit JFCQDQDSP in field JFCBFLG3 to invoke the tape volume DEQ at demount facility (see "DEQ at Demount Facility for Tape Volumes" on page 299)
- 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 301)
- Setting bit JFCDSCB 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.

- Setting bit JFCBLKSZ to indicate that the JFCB block size field has been set to zero by the application and that OPEN needs to set the block size value in the SIOT extension, if applicable, to zero as well. OPEN will also reset JFCBLKSZ off.

The secondary allocation quantity will be moved from the DSCB into the JFCB unless prevented by the setting of JFCNWRIT or JFCNDCB.

**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:
**Tip**: 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 31. 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 [z/OS DFSMS Macro Instructions for Data Sets](https://www.ibm.com/support/knowledgecenter/SSDM68_2.2.0/com.ibm.zos.zos/dftgtcman.html)), 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.

- You can also use the MF parameter on an RDJFCB macro. Its syntax, use, and effect are the same as documented for the OPEN macro in [z/OS DFSMS Macro Instructions for Data Sets](https://www.ibm.com/support/knowledgecenter/SSDM68_2.2.0/com.ibm.zos.zos/dftgtcman.html). In addition, you can code an execute-form RDJFCB macro that refers to a list-form OPEN macro that does not have MODE=31.
- 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 16 MB, although the calling program can be above 16M.

**Example**

In [Figure 32 on page 289](https://www.ibm.com/support/knowledgecenter/SSDM68_2.2.0/com.ibm.zos.zos/dftgtcman.html) 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.
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 z/OS DFSMS Using Data Sets.

After RDJFCB is performed, register 15 contains one of the following codes:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (X'00')</td>
<td>RDJFCB function completed successfully.</td>
</tr>
</tbody>
</table>

**Figure 32. Example Code Using RDJFCB Macro**

```
EX1   RDJFCB (INVEN,,MASTER),MF=L
  ...

EX2   RDJFCB MF=(E,EX1)
  ...

EX3   OPEN (,(RDBACK,LEAVE)),TYPE=J,MF=(E,EX1)
  ...

INVEN  DCB  EXLST=LST,...
MASTER DCB  EXLST=LSTB,...
LSTA   DS   0F
DC     AL1(EXLLASTE+EXLRJFCB)
DC     AL3(JFCBAREA)
  ...
JFCBAREA DS  0F,176C
  ...
LSTB   DS   0F
  ...
IHAEXLST ,  DCB exit list mapping
```

**System Macros**

Chapter 7. Using System Macro Instructions  289
Table 59. Return Codes from the RDJFCB Macro (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 (X'04')</td>
<td>One or more DCBs encountered one of the following conditions and were skipped. DCBs that were not skipped were processed successfully.</td>
</tr>
<tr>
<td></td>
<td>• The DCB was being processed by Open/Close/EOV or a similar function.</td>
</tr>
<tr>
<td></td>
<td>• No data set with the DDNAME that is in the DCB is allocated.</td>
</tr>
<tr>
<td></td>
<td>• The DCB is not open and its DDNAME is blank.</td>
</tr>
<tr>
<td>8 (X'08')</td>
<td>One or more DCBs had an ARL that could not be processed. Each ARL contains a reason code describing its status.</td>
</tr>
<tr>
<td></td>
<td>One or more DCBs might have encountered a condition described under return code 4. This type of ARL does not contain a reason code.</td>
</tr>
</tbody>
</table>

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:

Table 60.

<table>
<thead>
<tr>
<th>Hexadecimal Code (High-Order Byte)</th>
<th>Contents of Exit List Entry (Three Low-Order Bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>07</td>
<td>Address of a 176-byte area required if the RDJFCB or OPEN (TYPE=J) macro instruction is used.</td>
</tr>
</tbody>
</table>

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.

Requirement: Users running in 31-bit addressing mode must ensure that this area is located below 16 MB 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 z/OS DFSMS 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 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 (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) 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.

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. If the DISP setting is NEW, it does not cause the changed data set name to be created. If OPEN allocates the changed data set name, it is as if the DD had DISP=OLD. To create a data set on DASD, your program should call dynamic allocation by issuing SVC 99 (Requesting Dynamic Allocation Functions) in z/OS MVS Programming: Authorized Assembler Services Guide. Alternately your program might be able to use the REALLOC macro, but note that it requires authorization and carries restrictions. See "Creating (Allocating) a DASD Data Set Using REALLOC" on page 48.

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 z/OS MVS Programming: Authorized Assembler Services Guide and z/OS MVS Programming: Authorized Assembly Services Reference ABE-HSP.

You cannot extend a VTOC by this means.
Restriction: 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 BSAM or EXCP for Random I/O to a Multivolume Data Set: If you open a BDAM DCB for a multivolume data set, OPEN links your program to all volumes simultaneously so that your program can ignore volume boundaries and treat all volumes of the data set as one entity. If you open a BSAM or EXCP DCB for a multivolume data set, OPEN gives your program access to only one volume at a time. This is true for both disk and tape. To switch to another volume, your program issues a CHECK or FE0V macro for BSAM or E0V macro for EXCP. To return to a previous volume, you must close and reopen the data set, which would be slow.

Your program can use RDJFCB and OPEN TYPE=J with one DCB per volume to process all the volumes in parallel. Your program must keep track of which DCB is for each volume. Your program uses the RDJFCB macro to read in the JFCB, and uses OPEN with TYPE=J to open each volume of the data set. The coding example in Figure 33 on page 293 illustrates the procedure with EXCP DCBs.

This technique does not work for a JFCB disposition of NEW because OPEN TYPE=J honors modifications to the JFCB volume sequence number JFCBVLSQ only for tape and only if the JFCB file sequence is also modified.

This technique does not work with a striped data set because OPEN always opens all volumes of a striped data set in parallel as for BDAM.

If you are using BSAM to read non-striped volumes in parallel, you should avoid using the CHECK macro because it can automatically move to the next volume when you reach the end of the current volume. Use WAIT or EVENTS instead of CHECK. Refer to z/OS DFSMS Using Data Sets and z/OS DFSMS Macro Instructions for Data Sets for information about WAIT or EVENTS. If you optimize I/O with the MULTACC parameter of the DCBE macro, you also must issue TRUNC macros.

With tape, you cannot open more than one DCB per allocated drive. You can calculate the number of allocated drives from the TIOT entry length or by issuing the IEFDDSRV macro. IEFDDSRV returns the number of devices in DVAR_NUM_DVENT. Refer to z/OS MVS Programming: Assembler Services Reference [AR-XCT].

If your program does any of the following it will damage the data set:

- If you use BSAM and the OPEN option is not UPDAT and you issue WRITE macros, you might cause the data set to be extended to new tracks or to the next volume. In the latter case you have two DCBs open to one volume and they can interfere with each other.

- If you use BSAM and the OPEN option is not UPDAT and the last operation before CLOSE is WRITE (and CHECK, WAIT or EVENTS), then CLOSE marks that volume as being the last volume of the data set. The result might be every volume marked as being the last one. This is true also if you use EXCP and CLOSE finds that bit 0 of DCBOFLGS is on. For EXCP, see Table 36 on page 194 and “Device-Dependent Parameters” on page 203. If a program later tries to read the volumes of the data set sequentially, such as a program backing it up, that program will not read past the end of this volume. In addition, a later program trying to add records to the end of a volume by opening with the
EXTEND or OUTINX option or with the OUTPUT or OUTIN option with DISP=MOD, can add them to the wrong volume.

If the data set is newly allocated on DASD, space has been allocated only on the first volume unless you used the guaranteed space option of SMS. For both DASD and tape, if the data set has not yet been written on the volume, the OPEN fails.

---

**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. The system will accept both a type X'07' and a X'13' exit list entry. RDJFCB uses the first of each of them.

The format of the type 13 JFCB exit list entry is:

<table>
<thead>
<tr>
<th>Hexadecimal Code (High-Order Byte)</th>
<th>Contents of Exit List Entry (Three Low-Order Bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>Address of an allocation retrieval list.</td>
</tr>
</tbody>
</table>
Using RDJFCB to Retrieve Allocation Information: RDJFCB uses DCB exit list entry type 13 to retrieve allocation information (JFCBs and volume serial numbers) for data sets that might be concatenated. 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 Table 62. If you issue RDJFCB, this DCB exit list entry retrieves all JFCBs for the specified concatenated data sets, and lists of all volume serial numbers for these data sets. The block size, as specified on the DD statement of each data set, is put into the extended information segment following the volume serial numbers. If this block size field is 0, the block size of the data set is in the JFCB. 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 34 on page 298 for an example of usage.

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 16 MB. The allocation retrieval area (ARA), acquired by RDJFCB through a GETMAIN macro, can be above or below 16 MB.

The format of the IHAARL macro is:

```
IHAARL label, DSECT= YES, PREFIX= prefix, DESCR= NO
```

**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.

Table 62 and Table 63 on page 295 describe the formats of the allocation retrieval list and allocation retrieval area, respectively.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Bytes</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (X’00’)</td>
<td>2</td>
<td>ARLLEN</td>
<td>Length of this area. Value should be 36.</td>
</tr>
<tr>
<td>2 (X’02’)</td>
<td>2</td>
<td>ARLIDENT</td>
<td>EBCDIC ’AR’</td>
</tr>
</tbody>
</table>
### Table 62. Format of the Allocation Retrieval List (mapped by the IHAARL macro) (continued)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Bytes</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 (X'04')</td>
<td>1</td>
<td>ARLOPT1</td>
<td>Option byte.</td>
</tr>
<tr>
<td>0.......</td>
<td></td>
<td>ARLANY</td>
<td>ARA must be below 16 MB.</td>
</tr>
<tr>
<td>1... ....</td>
<td></td>
<td>ARA can be above 16 MB.</td>
<td></td>
</tr>
<tr>
<td>.1....</td>
<td></td>
<td>ARLUSS</td>
<td>Request ARA have a path name for each entry for which PATH was coded.</td>
</tr>
<tr>
<td>. xx xxxx</td>
<td></td>
<td>Reserved. Must be zero.</td>
<td></td>
</tr>
<tr>
<td>5 (X'05')</td>
<td>7</td>
<td>ARLRSVD1</td>
<td>Reserved. Must be zero.</td>
</tr>
<tr>
<td>12 (X'0C')</td>
<td>2</td>
<td>ARLRETRV</td>
<td>Number of data sets for which to retrieve information. If 0, retrieve all in the concatenation.</td>
</tr>
<tr>
<td>14 (X'0E')</td>
<td>2</td>
<td>ARLFIRST</td>
<td>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.</td>
</tr>
</tbody>
</table>

The following fields are set by RDJFCB:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Bytes</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 (X'10')</td>
<td>4</td>
<td>ARLAREA</td>
<td>Address of ARA. See Table 63.</td>
</tr>
<tr>
<td>20 (X'14')</td>
<td>1</td>
<td>ARLPOLL</td>
<td>Storage subpool containing ARA.</td>
</tr>
<tr>
<td>21 (X'15')</td>
<td>3</td>
<td>ARLRLEN</td>
<td>Length of ARA.</td>
</tr>
<tr>
<td>24 (X'18')</td>
<td>2</td>
<td>ARLRTRVD</td>
<td>Number of concatenated data sets for which JFCBs were retrieved.</td>
</tr>
<tr>
<td>26 (X'1A')</td>
<td>2</td>
<td>ARLCONC</td>
<td>Number of concatenated data sets. If no concatenation, this value is 1.</td>
</tr>
<tr>
<td>28 (X'1C')</td>
<td>1</td>
<td>ARLRCODE</td>
<td>Reason Code:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0 = Requested information was read.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The following reason codes are related to return code 8:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4 = ARLFIRST is greater than ARLCONC.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>8 = Insufficient storage to read information. ARLPOOL and ARLRLEN describe what RDJFCB needs.</td>
</tr>
<tr>
<td>29 (X'1D')</td>
<td>7</td>
<td>ARLRSVD2</td>
<td>Reserved. Used by RDJFCB.</td>
</tr>
</tbody>
</table>

### Table 63. Format of the Allocation Retrieval Area (mapped by the IHAARA macro)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Bytes</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (X'00')</td>
<td>2</td>
<td>ARALEN</td>
<td>Length of the information for this data set (including this field). 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 concatenation, if requested. Do not use the length field to calculate the number of volumes.</td>
</tr>
<tr>
<td>2 (X'02')</td>
<td>1</td>
<td>ARAFLG</td>
<td>Flags.</td>
</tr>
<tr>
<td>1... ....</td>
<td></td>
<td>ARAXINF</td>
<td>Extended information segment is present.</td>
</tr>
<tr>
<td>. xxx xxxx</td>
<td></td>
<td>Reserved. Must be zero.</td>
<td></td>
</tr>
<tr>
<td>3 (X'03')</td>
<td>1</td>
<td>ARAXINOF</td>
<td>Offset in doublewords from the beginning of the allocation retrieval area for the current data set to the extended information segment.</td>
</tr>
</tbody>
</table>
### System Macros

**Table 63. Format of the Allocation Retrieval Area (mapped by the IHAARA macro) (continued)**

<table>
<thead>
<tr>
<th>Offset</th>
<th>Bytes</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 (X'04')</td>
<td>176(Dec)</td>
<td>ARAJFCB</td>
<td>JFCB</td>
</tr>
<tr>
<td>180 (X'B4')</td>
<td>variable</td>
<td>*</td>
<td>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 determined.</td>
</tr>
</tbody>
</table>

Extended Information Segment. The DSECT name is ARAXINFO.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Bytes</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (X'00')</td>
<td>2</td>
<td>ARAXINLN</td>
<td>Length of the extended information segment (including this field).</td>
</tr>
<tr>
<td>2 (X'02')</td>
<td>2</td>
<td>ARAPATHO</td>
<td>If other than zero, ARAPATHO is the offset from beginning of extended information segment to the path name (ARAPATHNAME); in this case, the data set name in the JFCB is meaningless. If zero, then this entry does not contain a path name.</td>
</tr>
<tr>
<td>4 (X'04')</td>
<td>4</td>
<td>*</td>
<td>Reserved. Must be 0.</td>
</tr>
<tr>
<td>8 (X'08')</td>
<td>8</td>
<td>ARAXLBKS</td>
<td>Block size for this data set or 0. If 0 then block size can be found in the JFCB in ARAJFCB. If JFCBLKSZ is set, this field will be returned with a value of 0.</td>
</tr>
<tr>
<td>16 (X'10')</td>
<td>8</td>
<td>ARABKSLM</td>
<td>The maximum allowed value for system determined block size (BLKSLIM or DFABLKSZ value in DFA). This integer value is meaningful only if block size (BLKSIZE) is omitted from all sources and the application program opens for output. It is the first value available from these sources: 1. BLKSLIM keyword on the DD statement or dynamic allocation. 2. Block size limit in data class. Set by storage administrator. Available even if the data set is not SMS-managed. 3. System default set in TAPEBLKSLIM keyword in DEVSUPxx member in SYS1.PARMLIB by system programmer. Also available in DFA. 4. 32760 The minimum value in the current level of the operating system is 32760.</td>
</tr>
</tbody>
</table>

The following fields are present only if ARAPATHO is non-zero:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Bytes</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>2</td>
<td>ARAPATHLEN</td>
<td>DSECT Length of path name, excluding any trailing blanks. Calculate the value of x by adding the value in the two bytes in ARAPATHO to the address of ARAXINLN.</td>
</tr>
<tr>
<td>x+2</td>
<td>255</td>
<td>ARAPATHNAME</td>
<td>Path name.</td>
</tr>
<tr>
<td></td>
<td>257</td>
<td>ARAPNAMLEN</td>
<td>Symbolic length of maximum path name section.</td>
</tr>
</tbody>
</table>
System Macros

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. The DSECT name is ARAXINFO.

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 Table 59 on page 289.

**Example:** In Figure 34 on page 298, 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.
Figure 34. Sample Code Retrieving Allocation Information Part 1 of 2
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 JFCDDQDSP (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 "DEQ at Demount Facility for Tape Volumes" 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).

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:
  - 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.
  - 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.
  - The dequeued volume cannot 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 the preceding list 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 previously 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 z/OS DFSMSdfp Checkpoint/Rerarrt.

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). High-speed positioning is also available 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. When you write or read on an IBM 3590 Model A60 at the right hardware level it is not important to use the procedure described here. The magnetic tape subsystem gives these performance benefits automatically. This procedure will not degrade performance.

**Tip:** On an IBM 3480, IBM 3490, or older models of IBM 3590, this technique offers significantly better performance than the technique for setting the data set sequence number. In addition, systems with DFSMSrmm use this faster technique automatically for all cartridge tapes. For the IBM 3590 Model A60, both techniques give high performance.

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.

Exceptions:
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 [z/OS DFSMS 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:
Tip: 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 36. 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 that is 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 32 on page 289 for an example of coding that modifies a system-created JFCB.) The DD statement
must specify at least the device allocation (see z/OS MVS JCL User’s Guide for methods of preventing share status) and a ddname corresponding to the associated DCB DCBDDNAM field.

The MODE operand is not shown here because it is not allowed with the TYPE=J operand of the OPEN macro instruction.

Since OPEN with TYPE=J does not accept a JFCBX from the caller, you cannot change volume serials after the first five volumes.

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 (PURGE and RESTORE macros)

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 includes the following procedures:

- 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.

Restriction: 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 z/OS UNIX 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.

Refer to “General-Use Mapping Macros” in z/OS MVS Programming: Authorized Assembler Services Reference ALE-DYN for information about using the 31-bit interface provided by the PURGE function.

The format of the PURGE macro is:
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:

<table>
<thead>
<tr>
<th>Byte</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A byte that specifies the actions of the purge routines. The bit settings and their meanings are:</td>
</tr>
<tr>
<td>1...</td>
<td>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.</td>
</tr>
<tr>
<td>0...</td>
<td>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.</td>
</tr>
<tr>
<td>..1.</td>
<td>Post ECBs associated with purged I/O requests.</td>
</tr>
<tr>
<td>..1.</td>
<td>Halt I/O-request processing. (Quiesce I/O-request processing, if 0.)</td>
</tr>
<tr>
<td>...1</td>
<td>Purge related requests. (Only valid if a data-set purge is requested.)</td>
</tr>
<tr>
<td>....</td>
<td>Reserved—must be zero.</td>
</tr>
<tr>
<td>....</td>
<td>Do not purge the TCB request-block chain of asynchronously scheduled processing.</td>
</tr>
<tr>
<td>....</td>
<td>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.</td>
</tr>
<tr>
<td>....</td>
<td>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.)</td>
</tr>
<tr>
<td>1,2,3</td>
<td>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.)</td>
</tr>
<tr>
<td>4</td>
<td>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.)</td>
</tr>
<tr>
<td>5,6,7</td>
<td>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.</td>
</tr>
<tr>
<td>8</td>
<td>Value of X'00' or X'02' means that EXCP is the owner.</td>
</tr>
</tbody>
</table>
| 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
System Macros

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 37 on page 307.

**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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 (X’04’)</td>
<td>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.</td>
</tr>
<tr>
<td>8 (X’08’)</td>
<td>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.</td>
</tr>
<tr>
<td>20 (X’14’)</td>
<td>Another purge request has preempted your request. You might want to reissue your purge request in a time-controlled loop.</td>
</tr>
</tbody>
</table>

**Exception:** 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 37). 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.

Purge I/O Restore List

POINTER TO THE FIRST IOB. IF 1S, NO I/O REQUEST WAS QUIESCED.

Figure 37. The IOB Chain

RESTORE Macro Specification

The RESTORE macro is used to reprocess I/O requests.

The format of the RESTORE macro is:

```
/restored_macro_address
```

*address_address—RX-type address, (2-12) or (1)*
Address that you specified at byte 9 of the purge parameter list. See "PURGE Macro Specification" on page 304 for information about byte 9.

Performing Track Calculations (TRKCALC macro)

The TRKCALC macro performs DASD track capacity calculations. This macro is intended for EXCP applications and other advanced applications. You can use TRKCALC to determine:

- The number of equal-length records that can be written on a track
The total track capacity
Whether a record can be written in the space remaining on a track and return the new track balance
What the track balance would be if the last record were removed from a track
The length of the longest possible record that can be written to a track.

The TRKCALC routine issues no SVC instructions or I/O. 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.

TRKCALC works equally well for any track. It does not use the address of the track.

**Using TRKCALC**

This information provides an overview of how to use TRKCALC to accomplish various tasks. See "TRKCALC Macro Specification" on page 309 for details on how to code the TRKCALC parameters and on how output is returned.

**Determining the number of equal-length records that can be written on a track**

To determine the number of equal-length records that can be written on a track, code TRKCALC with FUNCTN=TRKCAP. You must specify the number of existing records on the track and the key and data length of the new records (using either the R, K, and DD keywords or the R, K, and DD bytes in the RKDD parameter).

If you wish to regard the track as being empty, specify an R value of 1. Otherwise, specify an R value that is one greater than the number of existing records on the track.

If the length of any existing record differs from the length of the new records (as specified in the DD value), then code the BALANCE parameter. Otherwise, omit the BALANCE parameter.

**Determining the total track capacity**

To determine the total track capacity, code

```
FUNCTN=TRKBAL, REMOVE=YES
```

and either R=1 or the R byte in RKDD set to 1.

**Note:** This value is useful only as input for the BALANCE parameter on later calls to TRKCALC to represent an empty track. You cannot write a record of this size.

**Determine whether a record can be written in the space remaining on a track and return the new track balance**

To determine whether a record can be written in the space remaining on the track and return the new track balance, code

```
FUNCTN=TRKBAL, REMOVE=NO
```

and the BALANCE parameter. You can supply this new track balance with the BALANCE parameter on a later call to TRKCALC.
Determine the track balance if the last record were removed from a track
To determine what the track balance would be if the last record were removed from the track, code FUNCTN=TRKBAL and REMOVE=YES. Use the R, K and DD parameters or the RKDD parameter to identify the record to be removed. It must be the last record on the track.

Determine the length of the longest possible record that can be written on a track
To determine the length of the longest possible record that can be written on a track, code FUNCTN=TRKBAL, REMOVE=NO, MAXSIZE=YES. You must specify the number of existing records on the track (using either the R keyword or the RKDD parameter) and a data length of X'FFFF' (using either the DD keyword or the RKDD parameter). The DD value of X'FFFF' is greater than is supported on any disk. Expect a return code 8, which means that the record does not fit and TRKCALC returned the size of the largest possible record.

If you wish to regard the track as being empty, specify an R value of 1.

If the track is not empty, specify an R value that is one greater than the number of existing records on the track and code the BALANCE parameter. In this case, TRKCALC will give return code 8 and the length of the longest possible record that will fit on the rest of the track.

Note: The value returned might be larger than what is supported by any access method other than EXCP.

Restrictions
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 z/OS UNIX files. You will receive unpredictable results if you use TRKCALC for z/OS UNIX 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:
FUNCTION=TRKBAL or TRKCAP
Specifies the function to be performed. 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, TRKCALC will treat record 1 as if it were being added to an empty track; if R is equal to 1 and REMOVE=YES is specified, TRKCAL will treat record 1 as if it were being 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.

Depending on the value for R, one of the following occurs:

- If R is equal to 1, TRKCALC ignores the BALANCE parameter and makes the calculation as if the track were empty.
- 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.

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 z/OS 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 and 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.
  - 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.
  - 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 STARDBAL. 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.

YES is valid only on a FUNCTION=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).

YES is valid only on a FUNCTION=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
   ,DEVTAB=addr
   ,UCB=addr
   ,TYPE=addr
   ,LOC=ANY
   ,BALANCE=addr
   ,REMOVE=NO
   ,MAXSIZE=YES
   ,RKDD=addr
   ,R=addr
   ,K=addr
   ,DD=addr
   ,REGSAVE=YES
   ,MF=(E,addr)
```

**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.

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 [z/OS HCD Planning](#) for more information.
**System Macros**

**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.

<table>
<thead>
<tr>
<th>E</th>
<th>Coded as shown.</th>
</tr>
</thead>
</table>
| 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:

```
/SM590000/SM590000
TRKCALC MF=L
/SM590000/SM630000
```

**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:

```
label TRKCALC MF=D
```

**Input Register Usage for All Forms of MF**

<table>
<thead>
<tr>
<th>Register</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>0, 2-12, 14</td>
<td>Available to provide input for keywords.</td>
</tr>
<tr>
<td>1</td>
<td>Only to provide the address of the parameter list for an MF=E call.</td>
</tr>
<tr>
<td>13</td>
<td>Input for keywords if REGSAVE=YES is not specified.</td>
</tr>
<tr>
<td>15</td>
<td>Work register to build the TRKCALC parameter list for the MF=E call; it is not available as an input register.</td>
</tr>
</tbody>
</table>

**Output from TRKCALC**

**FUNCTN=TRKBAL:**

<table>
<thead>
<tr>
<th>Output</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>R15=X'00'</td>
<td>The record fits on the track. Register 0 and STARBAL contain the new track balance.</td>
</tr>
<tr>
<td>R15=X'04'</td>
<td>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.</td>
</tr>
<tr>
<td>R15=X'08'</td>
<td>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. The key length is excluded from the count of maximum data bytes.</td>
</tr>
<tr>
<td>R15=X'0C'</td>
<td>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.</td>
</tr>
</tbody>
</table>

**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 :**

<table>
<thead>
<tr>
<th>Output</th>
<th>Meaning</th>
</tr>
</thead>
</table>
System Macros

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 and BALANCE is omitted. STARBAL is updated to the new track balance if the record fits; otherwise, STARBAL is left with the input track balance value when the request specified MAXSIZE=NO or was defaulted to MAXSIZE=NO. If the record does not fit on the track with MAXSIZE=YES specified, STARBAL is set to the maximum number of data bytes of a partial record that can fit on the remainder of the track with the specified key length (Register 15 set to 8) or STARBAL is set to zero when a partial record could not fit on the remainder of the track (Register 15 set to 4).

Return Codes from TRKCALC
The TRKCALC macro passes a return code in register 15. The return codes and their meanings are as follows:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (X'00')</td>
<td>Indicates that register 0 contains the new track balance.</td>
</tr>
<tr>
<td>4 (X'04')</td>
<td>Indicates that the record did not fit (register 0 = 0).</td>
</tr>
<tr>
<td>8 (X'08')</td>
<td>Indicates that the record did not fit. (Register 0 contains the maximum data length that does fit.)</td>
</tr>
<tr>
<td>12 (X'0C')</td>
<td>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.</td>
</tr>
</tbody>
</table>

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:

```assembly
TRKCALC FUNCTN=TRKCAP,TYPE=UTYPE,R=1,K=10,DD=DL, X MF=(E,(1))
ST 0,NUMREC SAVE NUMBER 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.
Perform calculations and conversions with 28-bit cylinder addresses (TRKADDR macro)

TRKADDR is an assembler macro that performs conversion and compare operations on DASD track addresses in the form CCCCcccH, where CCCC is the 16 low order bits of the cylinder number and ccc is the 12 high order bits of the cylinder number. This is referred to as a 28-bit cylinder address. TRKADDR works equally well with track addresses that contain a cylinder number less than or greater than 16 bits. It works with all DASD types that are supported by z/OS. Its functions include:

- Calculate the relative track number on the volume
- Compare two track addresses
- Extract the 28-bit cylinder number
- Extract the 4-bit track number
- Increment the track address by one track and increments the cylinder number if necessary.
- Normalize cylinder number to permit comparing one cchh against another
- Convert a relative track number to a 28-bit cylinder address
- Set the cylinder number in a 28-bit track address
- Convert a normalized track address into an absolute 28-bit track address.

Unless otherwise stated, you can specify any registers from 0 to 15 except that register 0 cannot be used to address storage. TRKADDR does not use any other registers, even register 13. You can invoke TRKADDR in 24-bit, 31-bit or 64-bit mode. If you use the SYSSTATE macro with AMODE64=YES in an earlier source code statement, then TRKADDR might generate more efficient code.

Calculate the relative track number on the volume (TRKADDR ABSTOREL)

The format of the execute form of the TRKADDR ABSTOREL macro is:

```
TRKADDR ABSTOREL label, cchh, (reg), REG=(rega, regb)
```

Converts absolute track address (CCCCcccH) to a relative track number. Calculates the relative track number on the volume and stores the result in the first register. The second register is used as a work register.

**Parameters**

`cchh`

Input: Track address in absolute format
System Macros

(reg)
This is a register from 1 to 15 containing the address of the cchh.

rega
Output: Relative track number

regb
Work register

Compare two track addresses (TRKADDR COMPARE)
The format of the execute form of the TRKADDR COMPARE macro is:

```
/                                      
 label TRKADDR COMPARE cchh1 (reg1) cchh2 (reg2) REG=(rega,regb) / 
```

Compares two track addresses in storage using the two registers as work registers. Sets condition code as for CLC machine instruction. Normalizes the two input values (CCCCcccH to cccCCCCH) and then compares the two normalized values. The input values are returned unchanged.

Parameters

_cchhl_ and _cchh2_
Input: Track addresses in absolute format to be compared. These values are returned unchanged.

(reg1) and (reg2)
This is a register from 1 to 15 containing the address of the cchh.

rega, regb
Work registers

Extract 28-bit cylinder number (TRKADDR EXTRACTCYL)
The format of the execute form of the TRKADDR EXTRACTCYL macro is:

```
/                                      
 label TRKADDR EXTRACTCYL cchh (reg) REG=(rega) / 
```

Extracts the 28-bit cylinder number to a register (CCCCcccH to 0cccCCCC). The input field is returned unchanged.

Parameters

_cchh_
Input: Track address in absolute format

(reg)
This is a register from 1 to 15 containing the address of the cchh.

rega
Output: Cylinder number from the input track address
Extract 4-bit track number (TRKADDR EXTRACTTRK)

The format of the execute form of the TRKADDR EXTRACTTRK macro is:

```
TRKADDR EXTRACTTRK label, cchh, (reg), REG=(rega)
```

Extracts the 4-bit track number to a register (CCCCcccH to 0000000H). The input field is returned unchanged.

**Parameters**

**cchh**
- Input: Track address in absolute format

**(reg)**
- This is a register from 1 to 15 containing the address of the cchh.

**rega**
- Output: Track number from the input track address

Increment track address (TRKADDR NEXTTRACK)

The format of the execute form of the TRKADDR NEXTTRACK macro is:

```
TRKADDR NEXTTRACK label, cchh, (reg), REG=(rega)
```

Increments the track address by one track and increments the cylinder number if necessary. The modified value is returned in the input **cchh** field. The register is used as a work register.

**Parameters**

**cchh**
- Input/Output: Track address in absolute format (CCCCcccH). Upon completion of the operation, this parameter contains the incremented track address in absolute format.

**(reg)**
- This is a register from 1 to 15 containing the address of the cchh.

**rega**
- Work register

Normalize cylinder number (TRKADDR NORMALIZE)

The format of the execute form of the TRKADDR NORMALIZE macro is:
Reverses the 16-bit and 12-bit portions of the cylinder number and stores the result in the 32-bit register with the H digit so you can use a simple unsigned comparison. The CCCCcccH becomes cccCCCCH. Use this when comparing one cchh against another. Normalize each and do an unsigned comparison.

Parameters

cchh
   Input: Track address in absolute format.

(reg)
   This is a register from 1 to 15 containing the address of the cchh.

rega
   Output: Normalized track address

Convert a relative track number to a 28-bit cylinder address (TRKADDR RELTOABS)

The format of the execute form of the TRKADDR RELTOABS macro is:

Converts relative track number to absolute format (CCCCcccH). RELTOABS converts a relative track number to a 28-bit cylinder address form in the passed cchh field. The register must be the first in an even/odd pair. The odd register must contain the relative track number on the volume. The macro modifies both registers. In 24-bit and 31-bit addressing modes these are four-byte registers. In 64-bit mode, they are eight-byte registers.

Parameters

cchh
   Output: Converted track address in absolute format.

(reg)
   This is a register from 1 to 15 containing the address of the cchh.

reg_pair
   Input: The first register of an even/odd pair where the odd register contains the track address to be converted.

Set cylinder number from register (TRKADDR SETCYL)

The format of the execute form of the TRKADDR SETCYL macro is:
Stores the cylinder number from the register to the 28-bits in the \textit{cchh} and sets H to 0 (\textit{ccccCCCC} to \textit{CCCCcccc}). Destroys the register.

**Parameters**

\textit{cchh} \\
Output: Contains the cylinder number

\textit{(reg)} \\
This is a register from 1 to 15 containing the address of the \textit{cchh}.

\textit{rega} \\
Input: Contains the cylinder number to be converted

**Convert normalized track address into an absolute 28-bit track address (TRKADDR NORMTOABS)**

The format of the execute form of the TRKADDR NORMTOABS macro is:

Reverses the 12-bit and 16-bit portions of the cylinder number and stores the result in the 32-bit register with the H digit. The \textit{ccccCCCC} becomes \textit{CCCCccccH}. Use this to convert a normalized track address to an absolute 28-bit track address.

**Parameters**

\textit{cchh} \\
Input: Cylinder address to be converted

\textit{(reg)} \\
This is a register from 1 to 15 containing the address of the \textit{cchh}.

\textit{rega} \\
Output: Contains the converted value

\textit{regb} \\
Work register

**Determining Level and Name of DFSMS**

You can use the IHADFA mapping macro to determine the level and name of DFSMS. 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.
Determining Version, Release, and Modification Level of DFSMS

The first byte of DFARELS contains a binary value that indicates the level of DFSMS on which your program is running:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Your program is not executing on DFSMS; 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.</td>
</tr>
<tr>
<td>1</td>
<td>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.</td>
</tr>
<tr>
<td>2</td>
<td>Your program is running on the level of DFSMS that is exclusive to OS/390® or one of the first two releases of z/OS. A value of X'02020A00' in DFARELS designates DFSMS for OS/390 Version 2 Release 10, Modification level 0. DFSMS was not modified in the first two releases of z/OS, so these releases also have a value of X'02020A00'.</td>
</tr>
<tr>
<td>&gt;3</td>
<td>Your program is running on a level of DFSMS that is part of a hypothetical replacement product after all versions and releases of z/OS. The system never returns this value. This represents IBM's intent in case there is such a product. The following three bytes designate the version, release, and modification level of that product. The value in the other three bytes is X'010100' or higher. It may differ from the level of installed z/OS.</td>
</tr>
<tr>
<td>3</td>
<td>Your program is running on a level of DFSMS that is exclusive to z/OS Version 1 Release 3 or higher. The following three bytes designate the version, release, and modification level of z/OS for which that DFSMS was designed. The value in the other three bytes is X'010300' or higher. It may differ from the level of installed z/OS.</td>
</tr>
</tbody>
</table>

IBM intends that for any future level of the DFA, the 4-byte DFARELS will not contain a value smaller than any previous value. If your purpose in testing DFARELS is to determine whether a particular feature of DFSMS is available, then we suggest 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 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 Level Determination" on page 342 for an alternative method of determining the level of DFSMS.

See "Data Facilities Area (DFA) Fields" on page 449 for a layout of the fields of the Data Facilities Area (DFA) control block.
Determining Name of DFSMS

If the value of DFARELS is '03010300' or greater, it means the system is z/OS Version 1 Release 3.0 or later. This means that field DFAELNMP points to a structure that contains the name of DFSMS. See DFAELNM in “Data Facilities Area (DFA) Fields” on page 449.

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. For example the value for z/OS Version 1 Release 10 is '03011000', '03' represent the name z/OS, '01' represents Version 1, '10' represents release 10, '00' represents modification level 0.

This is an example of a program using &IHADFARELS:
The IHADFA macro as shipped prior to DFSMS/MVS V1R3 did not set &IHADFARELS. 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.

```
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
  * 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.
  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, the ANY has no effect. DFP 3.x will reject it.
  DEVTYPE UCBLIST=(MYLIST,1,ANY),MF=(E,DEVLIST)
  .CONT
  .OLD DEVTYPE UCBLIST=(MYLIST,1),MF=(E,DEVLIST)
  .CONT ANOP
  .
  MYLIST DC A(0)
  DEVTLIST DEVTYPE ,(DEVINFO,24),MF=L

  Figure 38. Sample &IHADFARELS Program
```

The IHADFA macro as shipped prior to DFSMS/MVS V1R3 did not set &IHADFARELS. 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 39. Example of Determining Symbol Definition
```
Chapter 8. Displaying Messages on Cartridge Magnetic Tape Subsystems (MSGDISP macro)

This information covers 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, executes, and list forms of the macro are described here. The six main parameters of the macro and their functions are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOUNT</td>
<td>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.</td>
</tr>
<tr>
<td>VERIFY</td>
<td>Shows that a volume has been accepted by displaying its serial number and label type in positions 2 through 8.</td>
</tr>
<tr>
<td>RDY</td>
<td>Displays text in positions 2 through 7 while a data set is open.</td>
</tr>
<tr>
<td>DEMOUNT</td>
<td>Places a volume disposition indicator in position 1 of the display until a volume is demounted.</td>
</tr>
<tr>
<td>RESET</td>
<td>Clears the display area.</td>
</tr>
<tr>
<td>GEN</td>
<td>Provides the full range of display options, including the option to alternate two messages.</td>
</tr>
</tbody>
</table>

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 z/OS DFSMS Installation Exits for details.
MSGDISP—Displaying a Mount Message

The format of the MSGDISP macro with the MOUNT parameter is:

```
SM590000
MSGDISP MOUNT
  label
    UCB=(reg)
      'S'
        'A'
          'N'
              'X'
                  addr

SM590000
  MF=(E,addr)
    SER='volser'
      MEDIATYPE=n
          INDEX=NO
              TEST=YES
                WAIT=NO

SM590000
```

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.
Cartridge Messages

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.

MEDIATYPE=n
  Specifies what media type to mount for SCRTCH or PRIVAT mounts. The
MEDIATYPE keyword applies only when volumes are to be mounted on
devices that reside in a Manual Tape Library (MTL). If MEDIATYPE is
specified for devices outside of a Manual Tape Library, it is ignored. The value
n can be specified as a literal, the address of a 1 byte field containing the
value, or the name of the addressable field containing the value. Valid values
for MEDIATYPE are the numbers 1 through 8.

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.

Requirement: TEST=YES requires you to include the UCB mapping macro
(IEFUCBOB) in the source code.

Restriction: 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 16 MB line. These programs must pass the captured UCB
address or, if an actual address is passed, the UCB must reside below the
16 MB line.

INDEX=NO or YES
  Specifies whether the automatic cartridge loader (ACL) should be indexed to
satisfy a scratch mount request.

Chapter 8. Displaying Messages on Cartridge Magnetic Tape Subsystems (MSGDISP macro)  327
Cartridge Messages

**NO** Specifies that indexing should not be done regardless of the state of the ACL.

**YES**
- Specifies that indexing should be done if:
  - The ACL is present and loaded, and
  - The request is for SCRTCH or PRIVAT.

**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:

```
/SM590000/SM590000
MSGDISP VERIFY,UCB=(reg),LABEL='A' or 'N' or 'S' or 'X' or addr
,MF=(E,addr),SER='volser',addr,TEST=NO or YES
,WAIT=NO or YES

/SM590000/SM630000
VERIFY
```

**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.


Cartridge Messages

'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.

\textit{addr—RX-type address, A-type address, or (2-12)}
\begin{itemize}
  \item 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.
\end{itemize}

\textbf{MF=L or (E,addr)}
\begin{itemize}
  \item Specifies either the execute or list form of MSGDISP. If you do not specify this parameter, the standard form of the macro is used.
  \item \textbf{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.
  \item \textbf{(E,addr)} Specifies that the execute form of the macro and an existing parameter list is to be used.
\end{itemize}

\textit{SER='volser' or addr}
\begin{itemize}
  \item 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”.
  \item \textit{volser} Specifies the volume serial number as a literal. Express in apostrophes.
  \item \textit{addr—RX-type address, A-type address, or (2-12)} Specifies the address of the parameter list.
\end{itemize}

\textbf{TEST=NO or YES}
\begin{itemize}
  \item 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.
  \item \textbf{NO} Specifies that the macro expansion is not to include code that tests the UCB to determine whether the device supports message display.
  \item \textbf{YES} Specifies testing the UCB by the MSGDISP macro before attempting to invoke the message display service routine.
\end{itemize}

\textbf{Requirement:} TEST=YES requires you to include the UCB mapping macro (IEFUCBOB) in the source code. If this provision is not 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
Cartridge Messages

16 MB line. These programs must pass the captured UCB address or, if an actual address is passed, the UCB must reside below the 16 MB 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 that is 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:

```
SM590000 SM590000
MSGDISP RDY

label
,DCB=addr
,MF=L (E,addr)

'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.

**Tip:** 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 also 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.
Cartridge Messages

**(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:

\[
\text{MSGDISP-DEMOUNT, UCB=(reg), DISP='D', 'K', 'R', \text{addr}}
\]

\[
\text{MF=L, MF= \text{addr}, A', 'N', 'X', \text{addr}}
\]

\[
\text{MSER='volser_to_mount', SER='volser', \text{addr}}
\]

\[
\text{MEDIATYPE=n, INDEX=NO, TEST=YES, \text{addr}}
\]

\[
\text{WAIT=NO}
\]

**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.
Cartridge Messages

**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. “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.

MEDIATYPE=n
  Specifies what media type to demount for SCRTCH or PRIVAT demounts. The MEDIATYPE keyword applies only when volumes are to be demounted on devices that reside in a Manual Tape Library (MTL). If MEDIATYPE is specified for devices outside of a Manual Tape Library, it is ignored. The value n can be specified as a literal, the address of a 1 byte field containing the value, or the name of the addressable field containing the value. Valid values for MEDIATYPE are the numbers 1 through 8.

INDEX=NO or YES
  Specifies whether the ACL should be indexed to satisfy a scratch mount request.

  NO  Specifies that indexing should not be done regardless of the state of the ACL.

  YES  Specifies that indexing should be done if:
      • The ACL is present and loaded, and
      • The request is for SCRTCH or PRIVAT.

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.
Cartridge 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.

**Requirement:** TEST=YES requires you to include the UCB mapping macro (IEFUCBOB) in the source code. If this provision is not 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 16 MB line. These programs must pass the captured UCB address or, if an actual address is passed, the UCB must reside below the 16 MB 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:

```
/SM590000/SM590000
MSGDISP RESET
(label),UCB= (reg),UCBL=addr,MF= (E,addr)

/SM590000/SM630000
RESET
```

**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 internal status of the device (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.

**Requirement:** TEST=YES requires you to include the UCB mapping macro (IEFUCBOB) in the source code. If this provision is not 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 16 MB line. These programs must pass the captured UCB address or, if an actual address is passed, the UCB must reside below the 16 MB 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.

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),UCB=(reg)

,FLASH=STEADY

,INDEX=YES

,TEST=YES

,TXT='msgtxt'

,TXT2='altmsgtxt'

,WAIT=NO
```

**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.

**Hint:** 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.
**Cartridge Messages**

**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.

**INDEX=NO or YES**
- Specifies whether the ACL should be indexed to satisfy a scratch mount request.
  - **NO**
  - Specifies that indexing should not be done regardless of the state of the ACL.
  - **YES**
  - Specifies that indexing should be done if:
    - The ACL is present and loaded, and
    - The request is for SCRTCH or PRIVAT.

**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.

**Requirement**: TEST=YES requires you to include the UCB mapping macro (IEFUCBOB) in the source code. If this provision is not 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 16 MB line. These programs must pass the captured UCB address or, if an actual address is passed, the UCB must reside below the 16 MB 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.
Cartridge Messages

'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 is 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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (X'00')</td>
<td></td>
<td>Successful completion.</td>
</tr>
<tr>
<td>4 (X'04')</td>
<td></td>
<td>Device does not support MSGDISP.</td>
</tr>
<tr>
<td>8 (X'08')</td>
<td>1 (X'01')</td>
<td>Invalid input parameter.</td>
</tr>
<tr>
<td></td>
<td>2 (X'02')</td>
<td>Invalid DCB or DEBCHK error.</td>
</tr>
<tr>
<td></td>
<td>3 (X'03')</td>
<td>Environmental error.</td>
</tr>
</tbody>
</table>
Cartridge Messages

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 (X'04')</td>
<td>Authorization (TESTAUTH) violation.</td>
<td></td>
</tr>
<tr>
<td>5 (X'05')</td>
<td>Invalid UCB. Requires the address of a UCB, not a UCB copy.</td>
<td></td>
</tr>
<tr>
<td>6 (X'06')</td>
<td>Invalid request.</td>
<td></td>
</tr>
<tr>
<td>11 (X'0B')</td>
<td>Unsuccessful ESTAE macro call.</td>
<td></td>
</tr>
<tr>
<td>12 (X'0C')</td>
<td>Unsuccessful GETMAIN request.</td>
<td></td>
</tr>
<tr>
<td>12 (X'0C')</td>
<td>I/O error (The system posted the request for an error).</td>
<td></td>
</tr>
</tbody>
</table>

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 information 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 here with your program, the routine continues to execute correctly on future levels of the operating system.

The DFSMSdfp callable services provide 15 callable system services. They can be invoked by the high-level languages supported by z/OS Language Environment® and by assembler language callers.

Note: Nine of these callable services 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 362), but for detailed information on their use, refer to Character Data Representation Architecture Reference and Registry.

These services enable programs written in assembler language or high-level languages to use:

- IECTRKAD to perform conversions and compares of DASD track addresses that contain 16-bit or 28-bit cylinder numbers.
- 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.
- 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 Table 64 on page 353 for an explanation of the indicators.
- IGWASMS to return certain data set attributes for SMS managed data sets. 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.

  Note: If IGWASMS is called for a non-SMS managed data set, zeros are returned for the DSTYPE attribute.
- IGWASYS to determine the version, release, and modification level of DFSMSdfp on your system, and the status of the SMS subsystem.
- IGWLSHR to determine the DFSMSdfp share attributes in use on the current system.

This information describes calling the service from a nonreentrant program written in assembler language. See z/OS MVS Programming: Assembler Services Reference
Callable Services

**ABE-HSP** 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.

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 the following 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. For an example of using a LINK macro, see "Example" on page 347. For an example of using CALL and LOAD/CALL macros, see "Example" on page 344 and "Example" on page 349.

To invoke the callable services write a set of arguments in a specific order on the invocation. The number of arguments associated with each callable service is fixed, and the types of arguments are restricted to 32-bit binary integers (hereafter referred to as integers) and fixed-length EBCDIC character strings. The CALL statement format is described in the following information:

- "Call for DFSMS Level Determination"
- "Call for Data Set Attribute Retrieval" on page 345
- "Call for Data Set Backup-While-Open Support" on page 347
- "Call for DFSMSdfp Share Attributes" on page 353
- "Call for Record-Level Sharing Query (IGWARLS)" on page 355
- "Call for converting and comparing 28-bit cylinder addresses (IECTRKA1)" on page 359

### Call for DFSMS Level Determination

The DFSMS level determination call (IGWASYS) returns the version, release and modification levels of DFSMS. It also returns a code number to represent the name of the product that contains DFSMS. These four numbers represent the environment that DFSMS was designed to run in. That level might be earlier than the release for the rest of the operating system.

An alternative technique to determine the level of the system is described in "Determining Level and Name of DFSMS" on page 321.

### Format

The format of the system attribute IGWASYS call statement is:
Callable Services

CALL IGWASYS, return_code, ( reason_code, 
level_indicator, 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 354. 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 354. 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 or later. 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 integers:

• Version number
• Release number
• Modification level
• Special indicator

If you pass a value of 1 for level_indicator, DFSMS returns the system_level values as 3, 3, 2 and 1. The first three values reflect the fact that DFSMS 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, IGWASYS sets the fourth system_level word to a code that represents the product name that IGWASYS is part of. On no level of the system will this word contain a value that is smaller than on a prior release. One of the following values is returned:

1 "DFSMS/MVS". With MVS/ESA SP Version 5, it constituted an MVS/ESA system. With OS/390 MVS™, it was part of OS/390 Version 1 or Version 2. The first three system_level words represent DFSMS/MVS.

2 DFSMS is part of OS/390 and not a separate product. The first three system_level words show OS/390 Version 2, Release 10, Modification Level 0, or later. Those words represent the OS/390 level that DFSMS
Callable Services

was designed to support. You will see these values also on z/OS DFSMS Version 1 Release 1 and 2 because DFSMS was not changed from OS/390 2.10.

3 DFSMS is part of z/OS and not a separate product. The first three system-level words show z/OS Version 1, Release 3, Modification Level 0, or later. Those words represent the z/OS level that the DFSMS was designed to support.

>3 DFSMS is part of hypothetical replacement product after all versions and releases of z/OS. The system never returns this value. It represents IBM’s intent in case there is such a product. The other words represent the operating system level that the DFSMS was designed to support.

If the next release of the system does not contain a new level of DFSMS, IGWASYS will continue to return the original DFSMS release. This is to help with diagnosis of configuration problems and service.

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:

<table>
<thead>
<tr>
<th>Third word (modification level)</th>
<th>Fourth, first or second word</th>
</tr>
</thead>
<tbody>
<tr>
<td>Second word (release)</td>
<td>Fourth or first word</td>
</tr>
<tr>
<td>First word (version)</td>
<td>Fourth word</td>
</tr>
</tbody>
</table>

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 Table 65 on page 354 for the IGWASYS return and reason codes.

Example
The following example shows a system attribute call using a 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.
Callable Services

Format
The format of the data set attribute IGWASMS call statement is:

```
label CALL IGWASMS ,
         (return_code , reason_code ,
          prob_det , 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 354. 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 354. 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 354 for more information about
problem determination data. This is an output argument that must be defined
as a two-element array of integers.

dsnamelen
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 on which the IGWASMS service. 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
in the following circumstances:
- 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.
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 Table 65 on page 354 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 41. 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, read_write, dname_length, dname, select,
   bwo_flags, bwo_recov, bwo_resrv)
```

**Parameters**

*return_code*

Return code from IGWABWO. The return code is also returned in register 15.
Callable Services

Return codes are explained in “IGWASYS, IGWASMS, IGWABWO, IGWLSSHR Return and Reason Codes” on page 354. 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, IGWLSSHR Return and Reason Codes” on page 354. This is an output argument that must be defined as an integer.

prob_det
Problem determination data. See “IGWASYS, IGWASMS, IGWABWO, IGWLSSHR Return and Reason Codes” on page 354 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

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.
Callable Services

*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, 0CYYDDDFF, 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, HHMMSSSTFF, where:
  - HH Hours, based on a 24-hour clock
  - MM Minutes
  - SS Seconds
  - T 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 Table 65 on page 354 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.
Using the Backup-While-Open Facility

The following information describes the usage of the backup-while-open (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 347. The BWO indicators are described in Table 64 on page 353.

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.

```
CALL (R2),(RC1,RS1,PROB1,RW1,DSNLEN1,DSN1,SEL1,BWOF1,BWOR1,BWRE)
CALL (R2),(RC2,RS2,PROB2,RW2,DSNLEN2,DSN2,SEL2,BWOF2,BWOR2,BWRE)
```

--Figure 42. Example of IGWABWO Using LOAD and CALL Statements--
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 Table 64 on page 353 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...
Callable Services

has been processed and is consistent. The database manager can then reestablish normal access to the data set (thus opening 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 set) that do not create forward recovery logs, the database manager should clear the BWO1 flag and set the BWO3 flag at close (that is, 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.

Note:

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 by 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.

Table 64 on page 353 describes the meaning of each of the BWO (BWO1, BWO2, and BWO3) indicators.
## 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:

```plaintext
CALL IGWLSHR

(label

(return_code, reason_code)

(prob_det, 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 354. 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 354. This is an output argument that must be defined as an integer.
Callable Services

prob_det
Problem determination data. See "IGWASYS, IGWASMS, IGWABWO, IGWLSHR Return and Reason Codes" 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 Table 65 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.

<table>
<thead>
<tr>
<th>Return Code Dec (Hex)</th>
<th>Reason Code Dec (Hex)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (0)</td>
<td>0 (0)</td>
<td>The operation was successful.</td>
</tr>
<tr>
<td>4 (4)</td>
<td>4 (4)</td>
<td>The operation was successful, but the bwo_recov argument has no valid value for the data set specified in dname. 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.</td>
</tr>
</tbody>
</table>
### Call for Record-Level Sharing Query (IGWARLS)

IGWARLS is used to query the information in the catalog for record-level sharing.

**Format**

The format of the IGWARLS call statement is:

<table>
<thead>
<tr>
<th>Return Code Dec (Hex)</th>
<th>Reason Code Dec (Hex)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 (8)</td>
<td>4 (4)</td>
<td>An invalid <code>dsname_length</code> or <code>share_attr_selector</code> was specified. Correct the argument and retry the request.</td>
</tr>
<tr>
<td>8 (8)</td>
<td>8 (8)</td>
<td>An invalid <code>dsname</code> of blanks or invalid <code>share_attr_array_length</code> was specified. Correct the argument and retry the request.</td>
</tr>
<tr>
<td>8 (8)</td>
<td>12 (C)</td>
<td>An invalid <code>read_write</code> was specified. A value of 0 or 1 must be supplied. Correct the <code>read_write</code> argument and retry the request.</td>
</tr>
<tr>
<td>8 (8)</td>
<td>16 (10)</td>
<td>The values supplied for <code>bwo_flags</code> are not valid. BWO1, BWO2, and BWO3 must have a value of either 0 or 1. Correct the <code>bwo_flags</code> argument and retry the request.</td>
</tr>
<tr>
<td>8 (8)</td>
<td>20 (14)</td>
<td>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 <code>dsname</code> argument and retry the request.</td>
</tr>
<tr>
<td>8 (8)</td>
<td>24 (18)</td>
<td>An invalid <code>level_indicator</code> was specified. Correct the <code>level_indicator</code> argument and retry the request.</td>
</tr>
<tr>
<td>8 (8)</td>
<td>28 (1C)</td>
<td>An invalid <code>select</code> argument was specified. A value between 1 and 3 must be specified. Correct the <code>select</code> argument and retry the request.</td>
</tr>
<tr>
<td>8 (8)</td>
<td>32 (20)</td>
<td>The data set specified in <code>dsname</code> is not an SMS-managed data set. Correct the <code>dsname</code> argument and retry the request.</td>
</tr>
<tr>
<td>12 (C)</td>
<td>8 (8)</td>
<td>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.</td>
</tr>
<tr>
<td>12 (C)</td>
<td>12 (C)</td>
<td>The data set specified in <code>dsname</code> could not be found. Verify that the data set exists and has been correctly specified in <code>dsname</code>.</td>
</tr>
<tr>
<td>12 (C)</td>
<td>16 (10)</td>
<td>The data set specified in <code>dsname</code> is currently in MIGRATE status.</td>
</tr>
<tr>
<td>16 (10)</td>
<td>4 (4)</td>
<td>An error occurred on a call to catalog management. The catalog return code is in the first element of <code>prob_det</code> and the catalog reason code is in the second element of <code>prob_det</code>. See message IDC3009I for an explanation of the catalog return code and reason code. A catalog management return code of 8 indicates that the specified data set was not found. If you get this return code, correct <code>dsname</code> and retry the request.</td>
</tr>
<tr>
<td>20 (14)</td>
<td>4 (4)</td>
<td>A system error occurred during IGWASYS/SMS/BWO or IGWLHSR processing. The elements of <code>prob_det</code> contain additional diagnostic data. Contact IBM for programming assistance and provide them with the IGWASYS/SMS/BWO <code>return_code</code>, <code>reason_code</code>, and <code>prob_det</code> values.</td>
</tr>
<tr>
<td>36 (24)</td>
<td>4 (4)</td>
<td>Linkage cannot be established to the IGWLHSR 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.</td>
</tr>
</tbody>
</table>
Callable Services

```
CALL IGWARLS, (return_code-, reason_code-, prob_det-, 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 357. 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 357. This is an output argument. Define `reason_code` as an integer.

`prob_det`
Problem determination data. See "Return Codes" on page 357 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.
Callable Services

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, 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.

Table 66. IGWARLS Return and Reason Codes

<table>
<thead>
<tr>
<th>Return Code Dec (Hex)</th>
<th>Reason Code Dec (Hex)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (0)</td>
<td>0 (0)</td>
<td>The operation was successful.</td>
</tr>
<tr>
<td>8 (8)</td>
<td>4 (4)</td>
<td>An invalid dsnname_length was specified. Correct the dsnname_length argument and retry the request.</td>
</tr>
<tr>
<td>8 (8)</td>
<td>8 (8)</td>
<td>An invalid dsnname of blanks was specified. Correct the dsnname argument and retry the request.</td>
</tr>
<tr>
<td>8 (8)</td>
<td>20 (14)</td>
<td>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 dsnname argument and retry the request.</td>
</tr>
<tr>
<td>8 (8)</td>
<td>32 (20)</td>
<td>The data set specified in dsnname is not an SMS managed data set. Correct the dsnname argument and retry the request.</td>
</tr>
</tbody>
</table>

Chapter 9. Using DFSMSdfp Callable Services 357
### Table 66. IGWARLS Return and Reason Codes (continued)

<table>
<thead>
<tr>
<th>Return Code Dec (Hex)</th>
<th>Reason Code Dec (Hex)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 (8) 40 (28)</td>
<td></td>
<td>For IGWARLS, the logstreamid_length specified was invalid (&lt;0) or was not large enough to return the requested logstreamid. Correct the logstreamid_length argument and retry the request.</td>
</tr>
<tr>
<td>12 (C) 8 (8)</td>
<td></td>
<td>There is insufficient virtual storage to process the request. Retry the request. If the condition persists, contact IBM for programming assistance.</td>
</tr>
<tr>
<td>12 (C) 12 (C)</td>
<td></td>
<td>The data set specified in dsname could not be found. Verify that the data set exists and has been correctly specified in dsname.</td>
</tr>
<tr>
<td>12 (C) 14 (E)</td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>12 (C) 16 (10)</td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>16 (10) 4 (4)</td>
<td></td>
<td>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. 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.</td>
</tr>
<tr>
<td>20 (14) 4 (4)</td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>36 (24) 4 (4)</td>
<td></td>
<td>Linkage cannot be established to the IGWARLS 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.</td>
</tr>
</tbody>
</table>

### 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,
       LOGSTRML,LOGSTRM,RCVTMG,RCVTML,VSAMQUIS,BWO)
```

| RC1 DC    | F'0'  |
| RS1 DC    | F'0'  |
| PROB1 DC  | F'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 | XLB '00' |
| RCVTML DC | XLB '00' |
| VSAMQUIS DC| F'0'  |
| BWO DC    | F'0'  |

*Figure 43. Example of the IGWARLS Query Call Using LOAD and CALL Statements*
IECTRKAD is a callable service to perform conversions and compares of 28-bit cylinder addresses. The track addresses are in the form CCCCcccH, where CCCC is the 16 low order bits of the cylinder number and ccc is the 12 high order bits of it.

Cobol, PL/I, and C programs can call IECTRKAD without having to write assembler routines to invoke TRKADDR.

The caller requirements of IECTRKAD are:
- Register 1 contains an address to a parameter list. Register 0 is not used.
- Register 13 points to a standard register 18-word save area and registers 14 and 15 have their standard usage.
- The calling program can be in either 24-or 32-bit addressing mode
- The calling program can be executing in any protection key and in either supervisor or problem state

The called routine, IECTRKAD, has the following characteristics:
- No executable macro other than CALL and LINK is provided to call IECTRKAD.
- The called routine resides in SYS1.CSSLIB and is shipped in distribution library ACSSLIB.
- User program can link edit with the called routine to invoke IECTRKAD.
- User program can use the LINK macro or the LOAD and CALL macros to invoke IECTRKAD.
- IECTRKAD processing will use the equivalent TRKADDR function and pass the result back accordingly.
- IECTRKAD is release independent; if you link or bind IECTRKAD with your program, it will run on earlier or later releases of z/OS. This includes releases before the first availability of IECTRKAD.

**Format**

You can adapt the assembler language syntax shown here to your computer language such as C, COBOL or PL/I:

```
label CALL IECTRKAD, (operation, cchh1), (cchh2, number, returncode, reasoncode)
```

**Parameters**

Where the parameters are:

**operation**

Specifies the operation to perform. This parameter is the name of a field that contains character data of length 10. The allowable values are:
Callable Services

**ABSTOREL**
calculates the relative track number on the volume from the passed `cchh1`
track address.

**COMPARE**
compares the two track addresses passed in `cchh1` and `cchh2`.

**EXTRACTCYL**
extracts the 28-bit cylinder number from the passed `cchh1` track address.

**EXTRACTTRK**
extracts the 4-bit track number from the passed `cchh1` track address.

**NEXTTRACK**
increments the track address by one track and increments the cylinder
number if necessary from the passed `cchh1` track address.

**NORMALIZE**
reverses the 16-bit and 12-bit portions of the cylinder number from the
passed `cchh1` track address. The `CCCCcccH` becomes `cccCCCCH`. This could
be used to subsequently perform unsigned comparisons of track addresses.

**NORMTOABS**
reverses the 12-bit and 16-bit portions of the cylinder from the track
address passed in number. The `cccCCCCH` becomes `CCCCcccH`. Use this to
convert a normalized track address to an absolute 28-bit cylinder address.

**RELTOABS**
converts a relative track number, passed in `number`, to a 28-bit cylinder
address.

**SETCYL**
converts a relative cylinder number, passed in `number`, to a 28-bit cylinder
address and sets the head portion to zero.

If you request an operation that is less than 10 characters, it must be padded
on the right with blanks.

**cchh1**
Required parameter, `cchh1`, is a four-byte area containing a track address that
nominally is in the form of CCHH. For all functions except RELTOABS and
SETCYL this is input to the called routine. For RELTOABS, SETCYL, and
NORMTOABS this is output from the called routine.

**cchh2**
Optional positional parameter, `cchh2`, is a four-byte area whose meaning
depends on the operation specified by the first parameter. For all functions
except COMPARE and NEXTTRACK this parameter is ignored.

For COMPARE processing, `cchh2` contains the track address that is to be
compared to the first `cchh1` parameter.

For NEXTTRACK processing, `cchh2`, is the four-byte output area to contain the
track address (CCHH) of the next logical track on the volume. If the input
track number is 0 to 13, the output cylinder number will be the same and the
output track number will be one greater than the input track number. If the
input track number is 14, the output cylinder will be one higher than the
cylinder number in `cchh1` and the output track number will be 0. The called
routine does not check for numeric overflow.
**Callable Services**

**number**
Optional positional parameter, *number*, is a four-byte integer whose meaning depends on the operation specified by the first parameter.

For ABSTOREL processing, *number* is the output area that will contain the relative track number on the volume.

For COMPARE processing, *number* is the output area that will contain the result of the comparison. Zero means the inputs are equal. Negative one means the first one is lower. Positive one means the first one is higher.

For EXTRACTCYL processing, *number* is the output area that will contain the 28-bit cylinder number with the four high order bits set to zero.

For EXTRACTTRK processing, *number* is the output area that will contain the four-bit track number with the 28 high order bits set to zero.

For NEXTTRACK processing, *number* is an ignored parameter. It can be 0 or any valid virtual address. Not checked by the called routine.

For NORMALIZE processing, *number* is the output area that will contain the normalized version of the input $CCHH$. The $CCCCccch$ becomes $cccCCCCCH$. The high order 28-bits are the cylinder number and the low order four bits are the track number. This allows your code to do a more efficient comparison of one track address with many track addresses. Normalize each and do unsigned comparisons.

For NORMTOABS processing, *number* is the input area that contains the normalized track address to be converted. The $cccCCCCCH$ becomes $CCCCcccH$. Use this to convert a normalized track address to an absolute 28-bit cylinder address.

For RELTOABS processing, *number* is the input area that contains the relative track number on the volume. For example the first two tracks on the second cylinder (cylinder 1) have relative track numbers of 15 and 16. The called routine converts the relative track number to a 28-bit nonlinear cylinder address with a 4-bit head value in the low order four bits in the $cchh1$ output area.

For SETCYL processing, *number* is the input area that contains the cylinder number on the volume in the low order 28-bits with the four high order bits set to zero. The called routine splits the 28-bits into the high order 12-bits and low order 16-bits, reverses them in the output field, $cchh1$, with the cylinder address in the high order 28-bits and the low order four bits to zero.

**returncode**
$returncode$ is a 4-byte integer that contains the return code from IECTRKAD processing. $returncode$ is also returned in register 15. The return codes that could be set include the following:

- 0 Successful.
- 4 Successful, but with exceptions
- 8 Request was unsuccessful because of invalid or incorrect input. Refer to $reasoncode$ for more detailed information

**reasoncode**
$reasoncode$ is a 4-byte integer associated with a specific return code. $reasoncode$ is returned in Register 0.

No reason codes are supplied for return code 4:

The following reason codes are supplied for return code 8:
Callable Services

4 The H portion of a track address is not valid on a NEXTTRACK operation
8 The caller specified an invalid operation (the first parameter is not one of the supported operations)
12 The address of a required parameter is zero.

Character Data Representation Architecture (CDRA) APIs
The following CDRA APIs are included in the DFSMS product library. For more detailed description of both the APIs and their use, see Character Data Representation Architecture Reference and Registry (SC09-2190).

<table>
<thead>
<tr>
<th>API</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDRGESP</td>
<td>Get Encoding Scheme, Character Set, and Code Page Elements</td>
</tr>
<tr>
<td>CDRSCSP</td>
<td>Get Short Form (CCSID) from Specified ES (CS, CP)</td>
</tr>
<tr>
<td>CDRGSE</td>
<td>Get Encoding Scheme Element and its Subelements</td>
</tr>
<tr>
<td>CDRGCTL</td>
<td>Get Control Function Definition</td>
</tr>
<tr>
<td>CDRSMXC</td>
<td>Get Short Form (CCSID) with maximal CS for Specified ES, CP</td>
</tr>
<tr>
<td>CDRMSCI</td>
<td>Multiple-Step Convert Initialize</td>
</tr>
<tr>
<td>CDRMSCP</td>
<td>Multiple-Step Convert Perform</td>
</tr>
<tr>
<td>CDRMSCC</td>
<td>Multiple-Step Convert Clean Up</td>
</tr>
<tr>
<td>CDRXSRF</td>
<td>Extract Status and Reason Codes from Feedback Code</td>
</tr>
</tbody>
</table>
Chapter 10. Using the DESERV Exit

The DESERV exit was designed to support those programs which, before the binder, used SVC screening or replacement of the SVC table to trap SVC 21 (STOW) in order to monitor STOWs. DESERV provides an equivalent function to that obtained by SVC Screening or replacing the SVC table entry for SVC 21 (or SVC 12, BLDL). 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 following occurs:

- 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 following occurs:

- 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 member’s 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.
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.

For more detail and explanation of the DESERV functions, see z/OS DFSMS Using Data Sets for their macros, see z/OS DFSMS Macro Instructions for Data Sets.

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 z/OS MVS Programming: Authorized Assembler Services Reference SET-WTO and z/OS MVS Programming: Authorized Assembler Services Guide.

### 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 perform one of the following tasks:

- Not process this DESERV call (that is, 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 (that is, 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 SVCUPDATE 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:
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:

- **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 with 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 DFSMSdfp 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™ or PTF is installed.

If the DESERV FUNC=EXIT interface is called on DFSMS 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.

**Table 67. Installing or Replacing the DESERV Exit**

<table>
<thead>
<tr>
<th>EXIT_OPTION</th>
<th>then:</th>
<th>and these parameters...</th>
</tr>
</thead>
<tbody>
<tr>
<td>REPLACE</td>
<td>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.</td>
<td>EXIT_DST and EXIT_PREV_DSTPTR must be specified.</td>
</tr>
<tr>
<td>NOREPLACE (with no exit currently existing)</td>
<td>The exit definition is activated. A value of zero is returned as the previous DST address.</td>
<td>See REPLACE.</td>
</tr>
<tr>
<td>NOREPLACE (with an exit currently existing)</td>
<td>The currently defined exit remains the active exit. The address of the currently defined DST is returned as the previous DST address.</td>
<td>See REPLACE.</td>
</tr>
</tbody>
</table>

**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 Table 68.

Table 68. DESERV Screen Table Structure

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length or Bit Pattern</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (X'0')</td>
<td>20</td>
<td>DST</td>
<td>(structure)</td>
</tr>
<tr>
<td>0 (X'0')</td>
<td>16</td>
<td>DST_HEADER</td>
<td>(character)</td>
</tr>
<tr>
<td>0 (X'0')</td>
<td>8</td>
<td>DST_ID</td>
<td>Eyecatcher 'IGWDST' (character)</td>
</tr>
<tr>
<td>08 (X'08')</td>
<td>4</td>
<td>DST_LEN</td>
<td>Length of DST</td>
</tr>
<tr>
<td></td>
<td>X'14'</td>
<td>DST_LEN_IV</td>
<td>Constant to be used with DST_LEN</td>
</tr>
<tr>
<td>12 (X'0C')</td>
<td>1</td>
<td>DST_LEV</td>
<td>Control block level (unsigned)</td>
</tr>
<tr>
<td></td>
<td>X'01'</td>
<td>DST_LEV_IV</td>
<td>Constant to be used with DST_LEV</td>
</tr>
<tr>
<td>13 (X'0D')</td>
<td>1</td>
<td>DST_FLAGS</td>
<td>DST flags (unsigned)</td>
</tr>
<tr>
<td></td>
<td>xxxx xxxx.</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td></td>
<td>.... ...1</td>
<td>DST_FLAGS_PROP</td>
<td>Propagate this DST to lower level tasks</td>
</tr>
<tr>
<td>14 (X'0E')</td>
<td>2</td>
<td>DST_RES</td>
<td>Reserved</td>
</tr>
<tr>
<td>16 (X'10')</td>
<td>4</td>
<td>DST_EXIT</td>
<td>Address of exit routine screen table (address)</td>
</tr>
</tbody>
</table>

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:

Table 69. Deleting the DESERV Exit

<table>
<thead>
<tr>
<th>For this parameter</th>
<th>specify this:</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXIT_SCOPE</td>
<td>(application dependent)</td>
</tr>
<tr>
<td>EXIT_OPTION</td>
<td>DELETE</td>
</tr>
<tr>
<td>EXIT_DST</td>
<td>Currently active DST to be deleted</td>
</tr>
<tr>
<td>EXIT_PREV_DSTPTR</td>
<td>The DST to become active</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>For this parameter</th>
<th>choose this:</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXIT_OPTION</td>
<td>QUERY</td>
</tr>
<tr>
<td>EXIT_SCOPE</td>
<td>(application dependant)</td>
</tr>
<tr>
<td>EXIT_PREV_DSTPTR</td>
<td>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.)</td>
</tr>
</tbody>
</table>

Writing the DESERV Exit

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 Table 71.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length or Bit Pattern</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (X'0')</td>
<td>36</td>
<td>DESX</td>
<td>(structure)</td>
</tr>
<tr>
<td>0 (X'0')</td>
<td>16</td>
<td>DESX_HEADER</td>
<td>(character)</td>
</tr>
<tr>
<td>0 (X'0')</td>
<td>8</td>
<td>DESX_ID</td>
<td>Eyecatcher - IGWDESX (character)</td>
</tr>
<tr>
<td>08 (X'08')</td>
<td>4</td>
<td>DESX_LEN</td>
<td>Length of DESX (signed)</td>
</tr>
<tr>
<td>12 (X'0C')</td>
<td>1</td>
<td>DESX_LEV</td>
<td>Control block level (character)</td>
</tr>
<tr>
<td>13 (X'0D')</td>
<td>3</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>16 (X'10')</td>
<td>4</td>
<td>DESX_DESP_PTR</td>
<td>Address of caller’s DESP (address)</td>
</tr>
<tr>
<td>20 (X'14')</td>
<td>4</td>
<td>DESX_DST</td>
<td>Address of DESERV screen table (address)</td>
</tr>
<tr>
<td>24 (X'18')</td>
<td>1</td>
<td>DESX_CALLER_KEY</td>
<td>Key of DESERV caller in bits 0-3 (unsigned)</td>
</tr>
<tr>
<td>25 (X'19')</td>
<td>1</td>
<td>DESX_FLAGS</td>
<td>(bitstring)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DESX_BLDL_BIT</td>
<td>DESERV issues BLDL to process this GET request</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DESX_PREV_BIT</td>
<td>EXIT called before DESERV PUT or GET function</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DESX_POST_BIT</td>
<td>EXIT called after DESERV PUT or GET function</td>
</tr>
<tr>
<td>26 (X'1A')</td>
<td>2</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>28 (X'1C')</td>
<td>4</td>
<td>DESX_RETURN_CODE</td>
<td>Return code to be returned to DESERV caller (unsigned)</td>
</tr>
<tr>
<td>32 (X'20')</td>
<td>4</td>
<td>DESX_REASON_CODE</td>
<td>Reason code to be returned to DESERV caller (unsigned)</td>
</tr>
</tbody>
</table>
DESERV Exit

Note that the DESERV return and reason codes, with the exception of the PUT codes, can be found in [Z/OS DFSMS Using Data Sets](#). See Figure 45 on page 387 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 Table 72 on page 373 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 not 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 (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 Table 73 on page 375.

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.
### DESERV Exit

**Hint:** 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'.

Table 72. Structure of DESP for DESERV GET Invocations

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length or Bit Pattern</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 (X'00')</td>
<td>104 DESP</td>
<td>DE Services parameter list (structure)</td>
<td></td>
</tr>
<tr>
<td>00 (X'00')</td>
<td>16 DESP_HEADER</td>
<td>(character) 'IGWDESP'</td>
<td></td>
</tr>
<tr>
<td>00 (X'00')</td>
<td>8 DESP_ID</td>
<td>Eyecatcher IGWDESP (character)</td>
<td></td>
</tr>
<tr>
<td>08 (X'08')</td>
<td>24 DESP_LEN</td>
<td>Length of DESP (signed)</td>
<td></td>
</tr>
<tr>
<td>X'04'</td>
<td>DESP_LEN_IV</td>
<td>Constant to be used with DESP_LEN</td>
<td></td>
</tr>
<tr>
<td>12 (X'0C')</td>
<td>1 DESP_LEV</td>
<td>Control block level (character)</td>
<td></td>
</tr>
<tr>
<td>X'01'</td>
<td>DESP_LEV_IV</td>
<td>Constant to be used with DESP_LEV</td>
<td></td>
</tr>
<tr>
<td>13 (X'0D')</td>
<td>3 -</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>16 (X'10')</td>
<td>1 DESP_FUNC</td>
<td>Function type (unsigned)</td>
<td></td>
</tr>
<tr>
<td>X'07'</td>
<td>DESP_FUNC_DELETE</td>
<td>Function is DELETE</td>
<td></td>
</tr>
<tr>
<td>X'08'</td>
<td>DESP_FUNC_RENAME</td>
<td>Function is RENAME</td>
<td></td>
</tr>
<tr>
<td>X'09'</td>
<td>DESP_FUNC_UPDATE</td>
<td>Function is UPDATE</td>
<td></td>
</tr>
<tr>
<td>X'04'</td>
<td>DESP_FUNC_PUT</td>
<td>Function is PUT</td>
<td></td>
</tr>
<tr>
<td>X'01'</td>
<td>DESP_FUNC_GET</td>
<td>Function is GET</td>
<td></td>
</tr>
<tr>
<td>X'00'</td>
<td>DESP_FUNC_OMITTED</td>
<td>Function is omitted</td>
<td></td>
</tr>
<tr>
<td>17 (X'11')</td>
<td>3 -</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>20 (X'14')</td>
<td>4 -</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>24 (X'18')</td>
<td>12 DESP_DATA</td>
<td>Function data (character)</td>
<td></td>
</tr>
<tr>
<td>24 (X'18')</td>
<td>2 DESP_FLAGS</td>
<td>Flags (bitstring)</td>
<td></td>
</tr>
<tr>
<td>1...</td>
<td>DESP_BYPASS_LLA</td>
<td>0=USE LLA, 1=BYPASS LLA</td>
<td></td>
</tr>
<tr>
<td>x...</td>
<td>-</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>..1...</td>
<td>DESP_SUBPOOL_FLG</td>
<td>0=SUBPOOL not specified, 1=SUBPOOL specified</td>
<td></td>
</tr>
<tr>
<td>...1...</td>
<td>DESP_C370LIB</td>
<td>1=treat PDSs as C370LIB if @@DC370$ member exists</td>
<td></td>
</tr>
<tr>
<td>....</td>
<td>xx...</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>....</td>
<td>..1. DESP_SYSTEM_DCB</td>
<td>1=treat DCB as a system DCB</td>
<td></td>
</tr>
<tr>
<td>26 (X'1A')</td>
<td>1 -</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>27 (X'1B')</td>
<td>1 -</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>28 (X'1C')</td>
<td>1 DESP_LIBTYPE</td>
<td>Indicates whether a DCB or DEB is input =X'02', DEB input=X'01' (unsigned)</td>
<td></td>
</tr>
<tr>
<td>X'02'</td>
<td>DESP_LIBTYPE_DCBO</td>
<td>Constant to be used with DESP_LIBTYPE</td>
<td></td>
</tr>
<tr>
<td>X'01'</td>
<td>DESP_LIBTYPE_DEBO</td>
<td>Constant to be used with DESP_LIBTYPE</td>
<td></td>
</tr>
<tr>
<td>X'00'</td>
<td>DESP_LIBTYPE_OMITTED</td>
<td>Constant to be used with DESP_LIBTYPE</td>
<td></td>
</tr>
</tbody>
</table>
### DESERV Exit

Table 72. Structure of DESP for DESERV GET Invocations (continued)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length or Bit Pattern</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>29 (X'1D')</td>
<td>1</td>
<td>DESP_GETTYPE</td>
<td>Indicates whether Name List or PDSDE is input. (NAME_LIST input=(X'01', PDSDE input=X'02') (unsigned)</td>
</tr>
<tr>
<td>30 (X'1E')</td>
<td>1</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>31 (X'1F')</td>
<td>1</td>
<td>DESP_SUBPOOL</td>
<td>Subpool number for getting DESB.</td>
</tr>
<tr>
<td>32 (X'20')</td>
<td>1</td>
<td>DESP_CONN_INTENT</td>
<td>Connect intent (unsigned)</td>
</tr>
<tr>
<td>33 (X'21')</td>
<td>1</td>
<td>DESP_NAME_LIST_PTR</td>
<td>Name list address, valid if DESP_GETTYPE=X'01' (address)</td>
</tr>
<tr>
<td>34 (X'22')</td>
<td>1</td>
<td>DESP_CONN_INTENT_INPUT</td>
<td>INPUT</td>
</tr>
<tr>
<td>35 (X'23')</td>
<td>1</td>
<td>DESP_CONN_INTENT_EXEC</td>
<td>EXEC</td>
</tr>
<tr>
<td>36 (X'24')</td>
<td>4</td>
<td>DESP_DCБ_PTR</td>
<td>DCB address, valid if DESP_LIBTYPE=X'02' (address)</td>
</tr>
<tr>
<td>37 (X'25')</td>
<td>4</td>
<td>DESP_CONN_ID_PTR</td>
<td>Connect identifier address</td>
</tr>
<tr>
<td>38 (X'26')</td>
<td>4</td>
<td>DESP_AREA_PTR</td>
<td>Buffer address</td>
</tr>
<tr>
<td>39 (X'27')</td>
<td>4</td>
<td>DESP_AREA2</td>
<td>Buffer length (unsigned)</td>
</tr>
<tr>
<td>40 (X'28')</td>
<td>4</td>
<td>DESP_DEB_PTR</td>
<td>DEB address, valid if DESP_LIBTYPE=X'01' (address)</td>
</tr>
<tr>
<td>44 (X'2C')</td>
<td>4</td>
<td>DESP_CONN_ID_PTR</td>
<td>Connect identifier address</td>
</tr>
<tr>
<td>48 (X'30')</td>
<td>4</td>
<td>DESP_AREA_PTR</td>
<td>Address of output field for buffer address</td>
</tr>
<tr>
<td>52 (X'34')</td>
<td>4</td>
<td>DESP_AREA_PTR</td>
<td>Buffer address</td>
</tr>
<tr>
<td>56 (X'38')</td>
<td>4</td>
<td>DESP_AREA2</td>
<td>Buffer length (unsigned)</td>
</tr>
<tr>
<td>60 (X'3C')</td>
<td>4</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>64 (X'40')</td>
<td>4</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>68 (X'44')</td>
<td>4</td>
<td>DESP_ENTRY_GAP</td>
<td>Entry gap size (signed)</td>
</tr>
<tr>
<td>70 (X'46')</td>
<td>4</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>76 (X'4C')</td>
<td>4</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>80 (X'50')</td>
<td>4</td>
<td>DESP_NAME_LIST_PTR</td>
<td>Name list address, valid if DESP_GETTYPE=X'01' (address)</td>
</tr>
<tr>
<td>84 (X'54')</td>
<td>4</td>
<td>DESP_NAME_LIST2</td>
<td>Input list number of entries, valid if DESP_GETTYPE=X'01' (unsigned)</td>
</tr>
<tr>
<td>88 (X'58')</td>
<td>4</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>92 (X'5C')</td>
<td>4</td>
<td>DESP_PDSDE_PTR</td>
<td>BLDL directory entry address, valid if DESP_GETTYPE=X'02' (address)</td>
</tr>
<tr>
<td>92 (X'5C')</td>
<td>4</td>
<td>DESP_SMDE_PTR</td>
<td>SMDE directory entry address, valid if DESP_GETTYPE=X'03' (address)</td>
</tr>
<tr>
<td>96 (X'60')</td>
<td>4</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>100 (X'64')</td>
<td>4</td>
<td>-</td>
<td>Reserved</td>
</tr>
</tbody>
</table>
### Table 73. DESL Structure

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length or Bit Pattern</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 (X'00')</td>
<td>16</td>
<td>DESL</td>
<td>Name list (structure)</td>
</tr>
<tr>
<td>00 (X'00')</td>
<td>16</td>
<td>DESL_ENTRY</td>
<td>Name list entry (character)</td>
</tr>
<tr>
<td>00 (X'00')</td>
<td>1</td>
<td>DESL_FLAGS</td>
<td>Flags (unsigned)</td>
</tr>
<tr>
<td>1 (X'1')</td>
<td>1</td>
<td>DESL_MODULE_BUFFERED_LLA</td>
<td>Module is staged by LLA</td>
</tr>
<tr>
<td>X'03'</td>
<td></td>
<td>DESL_CODE_NEWNAME_EXISTS</td>
<td>For func=rename, indicates a new name already existed in the PDSE.</td>
</tr>
<tr>
<td>X'02'</td>
<td></td>
<td>DESL_CODE_ERROR</td>
<td>An unexpected error has occurred. The DESL_ERRCODE field is set to a DESRF value</td>
</tr>
<tr>
<td>X'01'</td>
<td></td>
<td>DESL_CODE_NOTFOUND</td>
<td>Entry not found or entry not processed. If func=rename, old name was not found.</td>
</tr>
<tr>
<td>X'00'</td>
<td></td>
<td>DESL_CODE_SUCC</td>
<td>Entry successfully processed</td>
</tr>
<tr>
<td>2 (X'2')</td>
<td>2</td>
<td>DESL_ERRCODE</td>
<td>Error reason code (low order halfword of DESERV reason code if error) (unsigned)</td>
</tr>
<tr>
<td>4 (X'4')</td>
<td>4</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>08 (X'08')</td>
<td>4</td>
<td>DESL_SMDE_PTR</td>
<td>Pointer to SMDE within DESB. Output for GET function, input for UPDATE function (address)</td>
</tr>
<tr>
<td>08 (X'08')</td>
<td>4</td>
<td>DESL_NEW_NAME_PTR</td>
<td>Pointer to new name (DESN) descriptor for RENAME function (address)</td>
</tr>
<tr>
<td>12 (X'0C')</td>
<td>4</td>
<td>DESL_NAME_PTR</td>
<td>Pointer to name (DESN) descriptor for GET and DELETE functions (address)</td>
</tr>
<tr>
<td>12 (X'0C')</td>
<td>4</td>
<td>DESL_OLD_NAME_PTR</td>
<td>Pointer to old name (DESN) descriptor for RENAME function (address)</td>
</tr>
</tbody>
</table>

The DESERV GET caller will have built the DESL to point to variable length names. The DESN DSECT maps these names in the IGVDES macro. See Table 74 for the DESN parameter list.

### Table 74. DESN Parameter List

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length or Bit Pattern</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 (X'00')</td>
<td>variable</td>
<td>DESN</td>
<td>Name record (structure)</td>
</tr>
<tr>
<td>00 (X'00')</td>
<td>2</td>
<td>DESN_LEN</td>
<td>Length of name that follows (unsigned)</td>
</tr>
<tr>
<td>2 (X'02')</td>
<td>variable</td>
<td>DESN_VAL</td>
<td>Name data (character)</td>
</tr>
</tbody>
</table>

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...
DESERV Exit

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 Table 75.

The basic SMDE format is shown in Table 76.

Table 75. DESB Parameter List

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length or Bit Pattern</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 (X'00')</td>
<td>variable</td>
<td>DESB</td>
<td>DEServ buffer header (structure)</td>
</tr>
<tr>
<td>00 (X'00')</td>
<td>40</td>
<td>DESB_FIXED</td>
<td>(character)</td>
</tr>
<tr>
<td>00 (X'00')</td>
<td>16</td>
<td>DESB_HEADER</td>
<td>(character)</td>
</tr>
<tr>
<td>00 (X'00')</td>
<td>8</td>
<td>DESB_ID</td>
<td>Eyecatcher - IGWDES (character)</td>
</tr>
<tr>
<td>08 (X'08')</td>
<td>4</td>
<td>DESB_LEN</td>
<td>Length of buffer (signed)</td>
</tr>
<tr>
<td>12 (X'0C')</td>
<td>1</td>
<td>DESB_LEV</td>
<td>Control block level (character)</td>
</tr>
<tr>
<td></td>
<td>X'01'</td>
<td>DESB_LEV_IV</td>
<td>Constant to be used with DESB_LEV</td>
</tr>
<tr>
<td>13 (X'0D')</td>
<td>3</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>16 (X'10')</td>
<td>4</td>
<td>DESB_NEXT</td>
<td>Next buffer pointer (address)</td>
</tr>
<tr>
<td>20 (X'14')</td>
<td>4</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>24 (X'18')</td>
<td>4</td>
<td>DESB_COUNT</td>
<td>Count of entries in this buffer (unsigned)</td>
</tr>
<tr>
<td>28 (X'1C')</td>
<td>4</td>
<td>DESB_AVAIL</td>
<td>Start of free space in buffer (unsigned)</td>
</tr>
<tr>
<td>32 (X'20')</td>
<td>1</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>33 (X'21')</td>
<td>1</td>
<td>DESB_SUBPOOL</td>
<td>Subpool number (unsigned)</td>
</tr>
<tr>
<td>34 (X'22')</td>
<td>2</td>
<td>DESB_GAP_LEN</td>
<td>Length of user-requested gap (unsigned)</td>
</tr>
<tr>
<td>36 (X'24')</td>
<td>4</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>40 (X'28')</td>
<td>variable</td>
<td>DESB_DATA</td>
<td>Start of data area (character)</td>
</tr>
</tbody>
</table>

Table 76. SMDE Format

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length or Bit Pattern</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 (X'00')</td>
<td>variable</td>
<td>SMDE</td>
<td>Member directory entry (structure)</td>
</tr>
<tr>
<td>00 (X'00')</td>
<td>44</td>
<td>SMDE_BASIC</td>
<td>Start of basic section (character)</td>
</tr>
<tr>
<td>00 (X'00')</td>
<td>16</td>
<td>SMDE_HDR</td>
<td>Header (character)</td>
</tr>
<tr>
<td>00 (X'00')</td>
<td>8</td>
<td>SMDE_ID</td>
<td>Eyecatcher (character)</td>
</tr>
<tr>
<td>08 (X'08')</td>
<td>4</td>
<td>SMDE_LEN</td>
<td>Length of control block. This is the sum of the sizes of the SMDE sections and the size of the user data. (unsigned)</td>
</tr>
</tbody>
</table>
### Table 76. SMDE Format (continued)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length or Bit Pattern</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>12 (X'0C')</td>
<td>1</td>
<td>SMDE_LVL</td>
<td>SMDE version number (unsigned)</td>
</tr>
<tr>
<td>X'01'</td>
<td></td>
<td>SMDE_LVL_VAL</td>
<td>Constant to be used with SMDE_LVL</td>
</tr>
<tr>
<td>13 (X'0D')</td>
<td>3</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>16 (X'10')</td>
<td>1</td>
<td>SMDE_LIBTYPE</td>
<td>Source library type. Possible values are declared below with names like SMDE_LIBTYPE_XXX. (unsigned)</td>
</tr>
<tr>
<td>X'03'</td>
<td></td>
<td>SMDE_C370LIB</td>
<td>Constant to be used with SMDE_LIBTYPE</td>
</tr>
<tr>
<td>X'02'</td>
<td></td>
<td>SMDE_LIBTYPE_HFS</td>
<td>Constant to be used with SMDE_LIBTYPE</td>
</tr>
<tr>
<td>X'01'</td>
<td></td>
<td>SMDE_LIBTYPE_PDSE</td>
<td>Constant to be used with SMDE_LIBTYPE</td>
</tr>
<tr>
<td>X'00'</td>
<td></td>
<td>SMDE_LIBTYPE_PDS</td>
<td>Constant to be used with SMDE_LIBTYPE</td>
</tr>
<tr>
<td>17 (X'11')</td>
<td>1</td>
<td>SMDE_FLAG</td>
<td>Flag byte (bitstring)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SMDE_FLAG_ALIAS</td>
<td>Entry is an alias</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SMDE_FLAG_LMOD</td>
<td>Member is a program</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>18 (X'12')</td>
<td>2</td>
<td>-</td>
<td>Reserved, must be zero</td>
</tr>
<tr>
<td>20 (X'14')</td>
<td>5</td>
<td>-</td>
<td>Extended MLTK (character)</td>
</tr>
<tr>
<td>20 (X'14')</td>
<td>1</td>
<td>-</td>
<td>Reserved, must be zero</td>
</tr>
<tr>
<td>21 (X'15')</td>
<td>4</td>
<td>SMDE_MLTK</td>
<td>MLT and concatenation number (character)</td>
</tr>
<tr>
<td>21 (X'15')</td>
<td>3</td>
<td>SMDE_MLT</td>
<td>MLT of member - zero if HFS (character)</td>
</tr>
<tr>
<td>24 (X'18')</td>
<td>1</td>
<td>SMDE_CNCT</td>
<td>Concatenation number (unsigned)</td>
</tr>
<tr>
<td>25 (X'19')</td>
<td>1</td>
<td>SMDE_LIBF</td>
<td>Library flag - Z-byte (unsigned)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SMDE_LIBF_TASKLIB</td>
<td>Constant to be used with SMDE_LIBF</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SMDE_LIBF_LINKLIB</td>
<td>Constant to be used with SMDE_LIBF</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SMDE_LIBF_PRIVATE</td>
<td>Constant to be used with SMDE_LIBF</td>
</tr>
<tr>
<td>26 (X'1A')</td>
<td>2</td>
<td>SMDE_NAME_OFF</td>
<td>Name offset (signed)</td>
</tr>
<tr>
<td>28 (X'1C')</td>
<td>2</td>
<td>SMDE_USRD_LEN</td>
<td>User data length (signed)</td>
</tr>
<tr>
<td>28 (X'1C')</td>
<td>2</td>
<td>SMDE_PMAR_LEN</td>
<td>Sum of lengths of program management attribute record sections (PMAR, PMARR, PMARL) (signed)</td>
</tr>
<tr>
<td>30 (X'1E')</td>
<td>2</td>
<td>SMDE_USERD_OFF</td>
<td>User data offset (signed)</td>
</tr>
<tr>
<td>30 (X'1E')</td>
<td>2</td>
<td>SMDE_PMAR_OFF</td>
<td>Program management attribute record offset (signed)</td>
</tr>
<tr>
<td>32 (X'20')</td>
<td>2</td>
<td>SMDE_TOKEN_LEN</td>
<td>Token length (signed)</td>
</tr>
<tr>
<td>34 (X'22')</td>
<td>2</td>
<td>SMDE_TOKEN_OFF</td>
<td>Token data offset (signed)</td>
</tr>
<tr>
<td>36 (X'24')</td>
<td>2</td>
<td>SMDE_PNAME_OFF</td>
<td>Primary name offset, zero for non-alias SMDES or if library type is a PDS and this is not a program. (signed)</td>
</tr>
<tr>
<td>38 (X'26')</td>
<td>2</td>
<td>SMDE_NLST_CNT</td>
<td>Number of note list entries that exist at beginning of user data field. Always zero for non-PDS members. (signed)</td>
</tr>
<tr>
<td>40 (X'28')</td>
<td>4</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>44 (X'2C')</td>
<td>variable</td>
<td>SMDE_SECTIONS</td>
<td>Start of entry sections (character)</td>
</tr>
</tbody>
</table>
Table 77 through Table 80 shows the optional SMDE_SECTIONS, or extensions to the SMDE.

**Table 77. Directory Entry Name Section**

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length or Bit Pattern</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 (X'00')</td>
<td>variable</td>
<td>SMDE_NAME</td>
<td>Name descriptor (structure)</td>
</tr>
<tr>
<td>00 (X'00')</td>
<td>2</td>
<td>SMDE_NAME_LEN</td>
<td>Length of entry name (signed)</td>
</tr>
<tr>
<td>2 (X'02')</td>
<td>variable</td>
<td>SMDE_NAME_VAL</td>
<td>Entry name (character)</td>
</tr>
</tbody>
</table>

**Table 78. Directory Entry Notelist Section (PDS Only)**

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length or Bit Pattern</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 (X'00')</td>
<td>variable</td>
<td>SMDE_NLST</td>
<td>Note list extension (structure)</td>
</tr>
<tr>
<td>00 (X'00')</td>
<td>4</td>
<td>SMDE_NLST_ENTRY</td>
<td>Note list entries (character)</td>
</tr>
<tr>
<td>00 (X'00')</td>
<td>3</td>
<td>SMDE_NLST_RLT</td>
<td>Note list record location token (character)</td>
</tr>
<tr>
<td>3 (X'03')</td>
<td>1</td>
<td>SMDE_NLST_NUM</td>
<td>Number of RLT described by this note list block. If 0 this is not a notelist but a data block. (unsigned)</td>
</tr>
</tbody>
</table>

**Table 79. Directory Entry Token Section**

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length or Bit Pattern</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 (X'00')</td>
<td>32</td>
<td>SMDE_TOKEN</td>
<td>(structure)</td>
</tr>
<tr>
<td>00 (X'00')</td>
<td>4</td>
<td>SMDE_TOKEN_CONNID</td>
<td>CONNECT_IDENTIFIER (unsigned)</td>
</tr>
<tr>
<td>4 (X'04')</td>
<td>4</td>
<td>SMDE_TOKEN_ITEMNO</td>
<td>Item number (unsigned)</td>
</tr>
<tr>
<td>08 (X'08')</td>
<td>24</td>
<td>SMDE_TOKEN_FT</td>
<td>File token (character)</td>
</tr>
</tbody>
</table>

**Table 80. Directory Entry Primary Name Section**

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length or Bit Pattern</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 (X'00')</td>
<td>variable</td>
<td>SMDE_PNAME</td>
<td>Primary name descriptor (structure)</td>
</tr>
<tr>
<td>00 (X'00')</td>
<td>2</td>
<td>SMDE_PNAME_LEN</td>
<td>Length of primary name (signed)</td>
</tr>
<tr>
<td>2 (X'02')</td>
<td>variable</td>
<td>SMDE_PNAME_VAL</td>
<td>Primary name (character)</td>
</tr>
</tbody>
</table>

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. Table 81 on page 379 shows the basic PMAR definition. Table 82 on page 381 and Table 83 on page 383 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.
### Table 81. Directory Entry Name Section

Data is always present at offset SMDE_PMAR_OFF in an SMDE.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length or Bit Pattern</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 (X'00')</td>
<td>30</td>
<td>PMAR</td>
<td>Basic section of program user data (structure)</td>
</tr>
<tr>
<td>00 (X'00')</td>
<td>30</td>
<td>PMAR_ENTRY</td>
<td>Alternative name for the PMAR section (character)</td>
</tr>
<tr>
<td>00 (X'00')</td>
<td>2</td>
<td>PMAR_SLEN</td>
<td>Section length (unsigned)</td>
</tr>
<tr>
<td>2 (X'02')</td>
<td>1</td>
<td>PMAR_LVL</td>
<td>PMAR format level (unsigned)</td>
</tr>
<tr>
<td>X'02'</td>
<td></td>
<td>PMAR_LVL_VAL</td>
<td>Constant to be used with PMAR</td>
</tr>
<tr>
<td>X'01'</td>
<td></td>
<td>PMAR_PM1_VAL</td>
<td>Constant to be used with PMAR</td>
</tr>
<tr>
<td>X'02'</td>
<td></td>
<td>PMAR_PM2_VAL</td>
<td>Constant to be used with PMAR</td>
</tr>
<tr>
<td>3 (X'03')</td>
<td>1</td>
<td>PMAR_PLVL</td>
<td>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)</td>
</tr>
<tr>
<td>X'01'</td>
<td></td>
<td>PMAR_PLVL_E_VAL</td>
<td>Constant to be used with PMAR_PLVL</td>
</tr>
<tr>
<td>X'02'</td>
<td></td>
<td>PMAR_PLVL_F_VAL</td>
<td>Constant to be used with PMAR_PLVL</td>
</tr>
<tr>
<td>X'03'</td>
<td></td>
<td>PMAR_PLVL_AOS_VAL</td>
<td>Constant to be used with PMAR_PLVL</td>
</tr>
<tr>
<td>X'04'</td>
<td></td>
<td>PMAR_PLVL_XA_VAL</td>
<td>Constant to be used with PMAR_PLVL</td>
</tr>
<tr>
<td>X'05'</td>
<td></td>
<td>PMAR_PLVL_B1_VAL</td>
<td>Constant to be used with PMAR_PLVL</td>
</tr>
<tr>
<td>X'06'</td>
<td></td>
<td>PMAR_PLVL_B2_VAL</td>
<td>Constant to be used with PMAR_PLVL</td>
</tr>
<tr>
<td>4 (X'04')</td>
<td>4</td>
<td>PMAR_ATR</td>
<td>Attribute bytes (character)</td>
</tr>
<tr>
<td>4 (X'04')</td>
<td>1</td>
<td>PMAR_ATRI</td>
<td>First attribute byte. These flags must be at the same offsets as the corresponding flags in PDS2ATR1 declared by macro IHAPDS. (bitstring)</td>
</tr>
<tr>
<td>1...</td>
<td>PMAR_RENT</td>
<td>Reenterable</td>
<td></td>
</tr>
<tr>
<td>1...</td>
<td>PMAR_REUS</td>
<td>Reusable</td>
<td></td>
</tr>
<tr>
<td>..1...</td>
<td>PMAR_OVLY</td>
<td>Overlay structure</td>
<td></td>
</tr>
<tr>
<td>...1...</td>
<td>PMAR_TEST</td>
<td>Module to be tested - TSO/E TEST</td>
<td></td>
</tr>
<tr>
<td>... 1...</td>
<td>PMAR_LOAD</td>
<td>Only loadable</td>
<td></td>
</tr>
<tr>
<td>... 1...</td>
<td>PMAR_SCTR</td>
<td>Scatter format</td>
<td></td>
</tr>
<tr>
<td>... 1...</td>
<td>PMAR_EXEC</td>
<td>Executable</td>
<td></td>
</tr>
<tr>
<td>... 1...</td>
<td>PMAR_1BLK</td>
<td>Load module contains only one block of text data and has no RLD data.</td>
<td></td>
</tr>
</tbody>
</table>
### DESERV Exit

*Table 81. Directory Entry Name Section (continued).* Data is always present at offset SMDE_PMAR_OFF in an SMDE.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length or Bit Pattern</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 (X'05')</td>
<td>1</td>
<td>PMAR_ATR2</td>
<td>Second attribute byte. These flags must be at the same offsets as the corresponding flags in PDS2ATR2 declared by macro IHAPDS. (bitstring)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PMAR_FLVL</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PMAR_ORGO</td>
<td>Linkage editor assigned origin of first block of text is zero.</td>
</tr>
<tr>
<td></td>
<td>..x.</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td></td>
<td>...1</td>
<td>PMAR_NRLD</td>
<td>Program contains no RLD items</td>
</tr>
<tr>
<td></td>
<td>.... 1...</td>
<td>PMAR_NREP</td>
<td>Module cannot be reprocessed by the linkage editor</td>
</tr>
<tr>
<td></td>
<td>.... .1..</td>
<td>PMAR_TSTN</td>
<td>Module contains TSO/E TEST symbol records</td>
</tr>
<tr>
<td></td>
<td>.... ..x.</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td></td>
<td>.... ...1</td>
<td>PMAR_REFR</td>
<td>Refreshable program</td>
</tr>
<tr>
<td>6 (X'06')</td>
<td>1</td>
<td>PMAR_ATR3</td>
<td>Third attribute byte. (bitstring)</td>
</tr>
<tr>
<td>6 (X'06')</td>
<td>1</td>
<td>PMAR_FTB1</td>
<td>Alternative name for flags byte. These flags must be at the same offsets as the corresponding flags in PDS2FTB1 declared by macro IHAPDS. (bitstring)</td>
</tr>
<tr>
<td></td>
<td>x...</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td></td>
<td>.1...</td>
<td>PMAR_BIG</td>
<td>This program requires 16MB or more of virtual storage.</td>
</tr>
<tr>
<td></td>
<td>..1.</td>
<td>PMAR_PAGA</td>
<td>Page alignment is required</td>
</tr>
<tr>
<td></td>
<td>...1</td>
<td>PMAR_XSSI</td>
<td>SSI information present</td>
</tr>
<tr>
<td></td>
<td>.... 1...</td>
<td>PMAR_XAPF</td>
<td>APF information present</td>
</tr>
<tr>
<td></td>
<td>.... .1..</td>
<td>PMAR_LFMT</td>
<td>PMARL follows PMAR.</td>
</tr>
<tr>
<td></td>
<td>.... ..xx</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>7 (X'07')</td>
<td>1</td>
<td>PMAR_ATR4</td>
<td>Fourth attribute byte (bitstring)</td>
</tr>
</tbody>
</table>
### Table 81. Directory Entry Name Section (continued). Data is always present at offset SMDE_PMAR_OFF in an SMDE.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length or Bit Pattern</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 (X’07’)</td>
<td>1</td>
<td>PMAR_FTB2</td>
<td>Alternative name for flags byte. These flags must be at the same offsets as the corresponding flags in PDS2FTB2 declared by macro IHAPDS. (bitstring)</td>
</tr>
<tr>
<td>1...</td>
<td>...</td>
<td>PMAR_ALTP</td>
<td>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.</td>
</tr>
<tr>
<td><em>xx</em>...</td>
<td>-</td>
<td>PMAR_RMOD</td>
<td>RMODE is ANY.</td>
</tr>
<tr>
<td><em>...1</em></td>
<td>_...</td>
<td>PMAR_AAMD</td>
<td>Alias entry point addressing mode. If B’00’, AMODE is 24. If B’10’, AMODE is 31. If B’11’, AMODE is ANY.</td>
</tr>
<tr>
<td>.... <em>xx</em></td>
<td>_...</td>
<td>PMAR_MAMD</td>
<td>Main entry point addressing mode. If B’00’, AMODE is 24. If B’10’, AMODE is 31. If B’11’, AMODE is ANY.</td>
</tr>
<tr>
<td>08 (X’08’)</td>
<td>1</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>9 (X’09’)</td>
<td>1</td>
<td>PMAR_AC</td>
<td>APF authorization code (unsigned)</td>
</tr>
<tr>
<td>10 (X’0A’)</td>
<td>4</td>
<td>PMAR_STOR</td>
<td>Virtual storage required (unsigned)</td>
</tr>
<tr>
<td>14 (X’0E’)</td>
<td>4</td>
<td>PMAR_EPM</td>
<td>Main entry point offset (unsigned)</td>
</tr>
<tr>
<td>18 (X’12’)</td>
<td>4</td>
<td>PMAR_EPA</td>
<td>This entry point offset (unsigned)</td>
</tr>
<tr>
<td>22 (X’16’)</td>
<td>4</td>
<td>PMAR_SSI</td>
<td>SSI information (bitstring)</td>
</tr>
<tr>
<td>22 (X’16’)</td>
<td>1</td>
<td>PMAR_CHLV</td>
<td>Change level of member (unsigned)</td>
</tr>
<tr>
<td>23 (X’17’)</td>
<td>1</td>
<td>PMAR_SSFB</td>
<td>SSI flag byte (bitstring)</td>
</tr>
<tr>
<td>24 (X’18’)</td>
<td>2</td>
<td>PMAR_MSER</td>
<td>Member serial number (Reserved)</td>
</tr>
<tr>
<td>26 (X’1A’)</td>
<td>4</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>30 (X’1E’)</td>
<td>variable</td>
<td>PMAR_END</td>
<td>End of basic section (character)</td>
</tr>
</tbody>
</table>

### Table 82. LSLoader Attributes Unique to Program Objects. If PMAR_LFMT=ON this section follows the PMAR basic section.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length or Bit Pattern</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 (X’00’)</td>
<td>50</td>
<td>PMARL</td>
<td>LSLoader section for program objects (structure)</td>
</tr>
<tr>
<td>00 (X’00’)</td>
<td>2</td>
<td>PMARL_SLEN</td>
<td>Section length (unsigned)</td>
</tr>
<tr>
<td>2 (X’02’)</td>
<td>48</td>
<td>PMARL_DATA</td>
<td>Section data (character)</td>
</tr>
<tr>
<td>2 (X’02’)</td>
<td>4</td>
<td>PMARL_ATR</td>
<td>Attribute bytes (character)</td>
</tr>
<tr>
<td>2 (X’02’)</td>
<td>1</td>
<td>PMARL_ATR1</td>
<td>Fifth attribute byte (bitstring)</td>
</tr>
<tr>
<td>1...</td>
<td>...</td>
<td>PMARL_NMIG</td>
<td>This program object cannot be converted directly to PDS load module format.</td>
</tr>
<tr>
<td><em>...1</em></td>
<td>_...</td>
<td>PMARL_PRIM</td>
<td>FETCHOPT PRIME option</td>
</tr>
<tr>
<td><em>...1</em></td>
<td>_...</td>
<td>PMARL_PACK</td>
<td>FETCHOPT PACK option</td>
</tr>
<tr>
<td><em>...x</em></td>
<td>_xxxx</td>
<td>-</td>
<td>Reserved</td>
</tr>
</tbody>
</table>
**DESERV Exit**

*Table 82. LSLoader Attributes Unique to Program Objects (continued). If PMAR_LFMT=ON this section follows the PMAR basic section.*

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length or Bit Pattern</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 (X'03')</td>
<td>1</td>
<td>PMARL_ATR2</td>
<td>Sixth attribute byte (bitstring)</td>
</tr>
<tr>
<td>1...</td>
<td>PMARL_CMPR</td>
<td>Compressed format module</td>
<td></td>
</tr>
<tr>
<td>.1...</td>
<td>PMARL_1RMOD</td>
<td>1st segment is RMODE Any, set for PM2-level PO only</td>
<td></td>
</tr>
<tr>
<td>..1...</td>
<td>PMARL_2RMOD</td>
<td>2nd segment is RMODE Any, set for PM2-level PO if there are at least two segments.</td>
<td></td>
</tr>
<tr>
<td>...1</td>
<td>PMARL_SEGM</td>
<td>Loader data includes a Segment Table with &gt;1 loadable entry or a Gas Table, set for PM2-level PO only.</td>
<td></td>
</tr>
<tr>
<td>.... 1...</td>
<td>PMARL_1ALIN</td>
<td>1st segment is page-aligned, set for PM2-level PO only</td>
<td></td>
</tr>
<tr>
<td>.... .1...</td>
<td>PMARL_2ALIN</td>
<td>2nd segment is page-aligned, set for PM2-level PO if there are at least 2 segments.</td>
<td></td>
</tr>
<tr>
<td>.... ..1...</td>
<td>PMARL_FILL</td>
<td>FILL option specified set for PM2-level PO only</td>
<td></td>
</tr>
<tr>
<td>.... ...x</td>
<td>-</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>4 (X'04')</td>
<td>1</td>
<td>PMARL_FILLVAL</td>
<td>FILL character value set for PM2-level PO only</td>
</tr>
<tr>
<td>5 (X'05')</td>
<td>1</td>
<td>-</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

**THE FOLLOWING NOTED FIELDS ARE NOT INTENDED FOR USE. INCLUDED HERE FOR INFORMATION PURPOSES ONLY.**

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length or Bit Pattern</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 (X'06')</td>
<td>4</td>
<td>PMARL_MPGS</td>
<td>Total length of program on DASD in pages (unsigned)</td>
</tr>
<tr>
<td>10 (X'0A')</td>
<td>40</td>
<td>PMARL_MDAT</td>
<td>DASD program descriptors (character)</td>
</tr>
<tr>
<td>10 (X'0A')</td>
<td>4</td>
<td>PMARL_TXTL</td>
<td>Length of text (unsigned)</td>
</tr>
<tr>
<td>14 (X'0E')</td>
<td>4</td>
<td>PMARL_TXTO</td>
<td>Offset to text (address)</td>
</tr>
<tr>
<td>18 (X'12')</td>
<td>4</td>
<td>PMARL_BDRL</td>
<td>Length of binder index (unsigned)</td>
</tr>
<tr>
<td>22 (X'16')</td>
<td>4</td>
<td>PMARL_BDRO</td>
<td>Offset to binder index (address)</td>
</tr>
<tr>
<td>26 (X'1A')</td>
<td>4</td>
<td>PMARL_RDTL</td>
<td>Length of PRDT (unsigned)</td>
</tr>
<tr>
<td>30 (X'1E')</td>
<td>4</td>
<td>PMARL_RDTO</td>
<td>Offset to PRDT (address)</td>
</tr>
<tr>
<td>34 (X'22')</td>
<td>4</td>
<td>PMARL_RATL</td>
<td>Length of PRAT (unsigned)</td>
</tr>
<tr>
<td>38 (X'26')</td>
<td>4</td>
<td>PMARL_RATO</td>
<td>Offset to PRAT (address)</td>
</tr>
<tr>
<td>42 (X'2A')</td>
<td>4</td>
<td>PMARL_NVSPGS</td>
<td>Number of virtual storage pages to contain program object, for PM2-level PO</td>
</tr>
<tr>
<td>42 (X'2A')</td>
<td>4</td>
<td>PMARL_LMDL</td>
<td>Length of LSLoader data (unsigned) for PM1-level PO</td>
</tr>
<tr>
<td>46 (X'2E')</td>
<td>4</td>
<td>PMARL_LMDO</td>
<td>Offset to LSLoader data (address)</td>
</tr>
<tr>
<td>50 (X'32')</td>
<td>2</td>
<td>PMARL_NSEG</td>
<td>Number of loadable segments</td>
</tr>
<tr>
<td>52 (X'34')</td>
<td>2</td>
<td>PMARL_NGAS</td>
<td>Count of entries in Gas Table</td>
</tr>
<tr>
<td>54 (X'36')</td>
<td>4</td>
<td>PMARL_1STOR</td>
<td>Virtual storage required for first loadable segment, valid when PMARL_NSEG &gt; 1.</td>
</tr>
<tr>
<td>58 (X'3A')</td>
<td>4</td>
<td>PMARL_2STOR</td>
<td>Virtual storage required for second loadable segment, valid when PMARL_NSEG &gt; 1.</td>
</tr>
</tbody>
</table>
### Table 82. LSLoader Attributes Unique to Program Objects (continued). If PMAR_LFMT=ON this section follows the PMAR basic section.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length or Bit Pattern</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>62 (X'3E')</td>
<td>4</td>
<td>PMARL_2TXTO</td>
<td>Offset to second txt segment including gas, valid when PMARL_NSEG &gt; 1.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>END INFORMATION ONLY FIELDS.</td>
</tr>
<tr>
<td>66 (X'42')</td>
<td>16</td>
<td>PMARL_TRACE</td>
<td>AUDIT trace data</td>
</tr>
<tr>
<td>66 (X'42')</td>
<td>4</td>
<td>PMARL_DATE</td>
<td>Date saved</td>
</tr>
<tr>
<td>70 (X'46')</td>
<td>4</td>
<td>PMARL_TIME</td>
<td>Time saved</td>
</tr>
<tr>
<td>74 (X'4A')</td>
<td>8</td>
<td>PMARL_USER</td>
<td>User or job identification</td>
</tr>
<tr>
<td>82 (X'52')</td>
<td>variable</td>
<td>PMARL_END</td>
<td>End of LSLoader section (character)</td>
</tr>
</tbody>
</table>

### Table 83. Attributes Unique to Load Modules (PDS only). If PMAR_LFMT=OFF then this section follows the PMAR basic section.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length or Bit Pattern</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 (X'00')</td>
<td>23</td>
<td>PMARR</td>
<td>Load module (PDS) attributes section (structure)</td>
</tr>
<tr>
<td>00 (X'00')</td>
<td>2</td>
<td>PMARR_SLEN</td>
<td>Section length (unsigned)</td>
</tr>
<tr>
<td>2 (X'02')</td>
<td>21</td>
<td>PMARR_DATA</td>
<td>Section data (character)</td>
</tr>
<tr>
<td>2 (X'02')</td>
<td>8</td>
<td>PMARR_TTRS</td>
<td>TTR fields (character)</td>
</tr>
<tr>
<td>2 (X'02')</td>
<td>3</td>
<td>PMARR_TTRT</td>
<td>TTR of first block of text (character)</td>
</tr>
<tr>
<td>5 (X'05')</td>
<td>1</td>
<td>PMARRZERO</td>
<td>Zero (character)</td>
</tr>
<tr>
<td>6 (X'06')</td>
<td>3</td>
<td>PMARR_TTRN</td>
<td>TTR of note list or scatter translation table. Used for modules in scatter load format or overlay structure only. (character)</td>
</tr>
<tr>
<td>9 (X'09')</td>
<td>1</td>
<td>PMARR_NL</td>
<td>Number of entries in note list for scatter format modules and modules in overlay structure otherwise zero. (address)</td>
</tr>
<tr>
<td>10 (X'0A')</td>
<td>2</td>
<td>PMARR_FTBL</td>
<td>Length of first block of text (signed)</td>
</tr>
<tr>
<td>12 (X'0C')</td>
<td>3</td>
<td>PMARR_ORG</td>
<td>Load module origin if 0 (unsigned)</td>
</tr>
<tr>
<td>12 (X'0C')</td>
<td>2</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>14 (X'0E')</td>
<td>1</td>
<td>PMARR_RLDS</td>
<td>Number of RLD/CTL records that follow the first text record</td>
</tr>
<tr>
<td>15 (X'F')</td>
<td>8</td>
<td>PMARR_SCAT</td>
<td>Scatter load information (character)</td>
</tr>
<tr>
<td>15 (X'F')</td>
<td>2</td>
<td>PMARR_SLSZ</td>
<td>Scatter list length (unsigned)</td>
</tr>
<tr>
<td>17 (X'11')</td>
<td>2</td>
<td>PMARR_TTSZ</td>
<td>Translation table length (unsigned)</td>
</tr>
<tr>
<td>19 (X'13')</td>
<td>2</td>
<td>PMARR_ESDT</td>
<td>ESDID of first text block (character)</td>
</tr>
<tr>
<td>21 (X'15')</td>
<td>2</td>
<td>PMARR_ESDC</td>
<td>ESDID of EP control section (character)</td>
</tr>
<tr>
<td>23 (X'17')</td>
<td>variable</td>
<td>PMARR_END</td>
<td>End of load module attributes (character)</td>
</tr>
</tbody>
</table>

### Table 84. Alias in Unformatted Form. Used only as input to the PUT function.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length or Bit Pattern</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 (X'00')</td>
<td>7</td>
<td>PMARA</td>
<td>PMAR alias entry section (structure)</td>
</tr>
</tbody>
</table>
### Table 84. Alias in Unformatted Form (continued). Used only as input to the PUT function.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length or Bit Pattern</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 (X'00')</td>
<td>2</td>
<td>PMARA_LEN</td>
<td>Section length (unsigned)</td>
</tr>
<tr>
<td>2 (X'02')</td>
<td>5</td>
<td>PMARA_DATA</td>
<td>Section data (character)</td>
</tr>
<tr>
<td>2 (X'02')</td>
<td>4</td>
<td>PMARA_EPA</td>
<td>Entry point offset (unsigned)</td>
</tr>
<tr>
<td>6 (X'06')</td>
<td>1</td>
<td>PMARA_ATR</td>
<td>Attribute bytes (character)</td>
</tr>
<tr>
<td>6 (X'06')</td>
<td>1</td>
<td>PMARA_ATR1</td>
<td>First attribute byte (bitstring)</td>
</tr>
<tr>
<td>6 (X'06')</td>
<td>1</td>
<td>PMARA_FTB2</td>
<td>Alternative name for flags byte. These flags must be at the same offsets as the corresponding flags in PDS2FTB2 declared by macro IHAPDS. (bitstring)</td>
</tr>
</tbody>
</table>

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. Table 85 shows the fields of the DESP that pertain to the DESERV PUT invocation.

### Table 85. DESERV PUT DESP Fields

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length or Bit Pattern</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 (X'00')</td>
<td>104</td>
<td>DESP</td>
<td>DE Services parameter list (structure)</td>
</tr>
<tr>
<td>00 (X'00')</td>
<td>16</td>
<td>DESP_HEADER</td>
<td>Standard header (character)</td>
</tr>
<tr>
<td>00 (X'00')</td>
<td>8</td>
<td>DESP_ID</td>
<td>Eyecatcher 'IGWDESP' (character)</td>
</tr>
<tr>
<td>08 (X'08')</td>
<td>4</td>
<td>DESP_LEN</td>
<td>Length of DESP (signed)</td>
</tr>
<tr>
<td>12 (X'0C')</td>
<td>1</td>
<td>DESP_LEV</td>
<td>Control block level (character)</td>
</tr>
<tr>
<td>13 (X'D')</td>
<td>3</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>16 (X'10')</td>
<td>1</td>
<td>DESP_FUNC</td>
<td>Function type (GET=X'01', PUT=X'04', DELETE=X'07', RENAME=X'08', UPDATE=X'09') (unsigned)</td>
</tr>
</tbody>
</table>

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
Table 85. DESERV PUT DESP Fields  (continued)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length or Bit Pattern</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>17 (X'11')</td>
<td>3</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>20 (X'14')</td>
<td>4</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>24 (X'18')</td>
<td>12</td>
<td>DESP_DATA</td>
<td>Function data (character)</td>
</tr>
<tr>
<td>24 (X'18')</td>
<td>2</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>26 (X'1A')</td>
<td>1</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>27 (X'1B')</td>
<td>1</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>28 (X'1C')</td>
<td>1</td>
<td>DESP_LIBTYPE</td>
<td>Indicates whether a DCB or a DEB is input. (unsigned)</td>
</tr>
<tr>
<td>X'02'</td>
<td>DESP_LIBTYPE_DCB</td>
<td>DCB input</td>
<td></td>
</tr>
<tr>
<td>X'01'</td>
<td>DESP_LIBTYPE_DEB</td>
<td>DEB input</td>
<td></td>
</tr>
<tr>
<td>X'00'</td>
<td>DESP_LIBTYPE_OMITTED</td>
<td>Omitted</td>
<td></td>
</tr>
<tr>
<td>29 (X'1D')</td>
<td>1</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>30 (X'1E')</td>
<td>1</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>31 (X'1F')</td>
<td>1</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>32 (X'20')</td>
<td>1</td>
<td>DESP_OPTION</td>
<td>REPLACE option (REPLACE=X'01', NOREPLACE=X'00') (unsigned)</td>
</tr>
<tr>
<td>X'02'</td>
<td>DESP_OPTION_REPLACE_ALIAS</td>
<td>Constant to be used with DESP_OPTION</td>
<td></td>
</tr>
<tr>
<td>X'01'</td>
<td>DESP_OPTION_REPLACE</td>
<td>Constant to be used with DESP_OPTION</td>
<td></td>
</tr>
<tr>
<td>X'00'</td>
<td>DESP_OPTION_NOREPLACE</td>
<td>Constant to be used with DESP_OPTION</td>
<td></td>
</tr>
<tr>
<td>33 (X'21')</td>
<td>1</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>34 (X'22')</td>
<td>1</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>35 (X'23')</td>
<td>1</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>36 (X'24')</td>
<td>4</td>
<td>DESP_DCPtr</td>
<td>DCB address</td>
</tr>
<tr>
<td>40 (X'28')</td>
<td>4</td>
<td>DESP_DEB_PTR</td>
<td>DEB address</td>
</tr>
<tr>
<td>44 (X'2C')</td>
<td>4</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>48 (X'30')</td>
<td>4</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>52 (X'34')</td>
<td>4</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>56 (X'38')</td>
<td>4</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>60 (X'3C')</td>
<td>4</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>64 (X'40')</td>
<td>4</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>68 (X'44')</td>
<td>4</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>72 (X'48')</td>
<td>4</td>
<td>DESP_MEM_DATA_PTR</td>
<td>MEM_DATA_ADDRESS</td>
</tr>
<tr>
<td>76 (X'4C')</td>
<td>4</td>
<td>DESP_MEM_DATA2</td>
<td>MEM_DATA entry count (unsigned)</td>
</tr>
<tr>
<td>80 (X'50')</td>
<td>4</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>84 (X'54')</td>
<td>4</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>88 (X'58')</td>
<td>4</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>92 (X'5C')</td>
<td>4</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>96 (X'60')</td>
<td>4</td>
<td>-</td>
<td>Reserved</td>
</tr>
</tbody>
</table>
## DESERV Exit

### Table 85. DESERV PUT DESP Fields (continued)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length or Bit Pattern</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>100 (X’64’)</td>
<td>4</td>
<td>-</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

**PUT Return and Reason Codes**

[Figure 45 on page 387](#) describes the return and reason codes for the PUT function.
### Return Code

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESRC_SUCC</td>
<td>X'00' Successful processing</td>
</tr>
<tr>
<td>DESRC_INFO 4</td>
<td>Not completely successful</td>
</tr>
<tr>
<td>DESRC_WARN 8</td>
<td>Results questionable</td>
</tr>
<tr>
<td>DESRC_PARM 12</td>
<td>Missing/invalid parameters</td>
</tr>
<tr>
<td>DESRC_CALR 16</td>
<td>Caller has a problem</td>
</tr>
<tr>
<td>DESRC_ENVR 20</td>
<td>Resources unavailable</td>
</tr>
<tr>
<td>DESRC_IOER 24</td>
<td>I/O error</td>
</tr>
<tr>
<td>DESRC_MEDE 28</td>
<td>Media error</td>
</tr>
<tr>
<td>DESRC_DSLE 32</td>
<td>Data set logical error</td>
</tr>
<tr>
<td>DESRC_SEVE 36</td>
<td>Severe error</td>
</tr>
</tbody>
</table>

### Reason Code

<table>
<thead>
<tr>
<th>Reason Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESRS_SUCC</td>
<td>X'00' Successful processing</td>
</tr>
<tr>
<td>DESRS_info</td>
<td>Not completely successful</td>
</tr>
<tr>
<td>DESRS_warn</td>
<td>Results questionable</td>
</tr>
<tr>
<td>DESRS_parm</td>
<td>Missing/invalid parameters</td>
</tr>
<tr>
<td>DESRS_calr</td>
<td>Caller has a problem</td>
</tr>
<tr>
<td>DESRS_envr</td>
<td>Resources unavailable</td>
</tr>
<tr>
<td>DESRS_ioer</td>
<td>I/O error</td>
</tr>
<tr>
<td>DESRS_mede</td>
<td>Media error</td>
</tr>
<tr>
<td>DESRS_dsle</td>
<td>Data set logical error</td>
</tr>
<tr>
<td>DESRS_seve</td>
<td>Severe error</td>
</tr>
</tbody>
</table>

### Reason Code (continued)

<table>
<thead>
<tr>
<th>Reason Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESRS_DST_ALREADY_EXISTS</td>
<td>X'44B' An EXIT exists and DESERV FUNC EXIT with NOREPLACE specified was issued. The current exit is not replaced</td>
</tr>
<tr>
<td>DESRS_DST_COMP_SWAP_FAILED</td>
<td>X'451' An EXIT_OPTION=DELETE specified a DST address that was not current. The compare and swap failed.</td>
</tr>
<tr>
<td>DESRS_EXIT_OPTION_INVALID</td>
<td>X'44F' The EXIT_OPTION specified is not supported</td>
</tr>
<tr>
<td>DESRS_EXIT_SCOPE_INVALID</td>
<td>X'45' The EXIT_SCOPE specified is not supported</td>
</tr>
<tr>
<td>DESRS_EXIT_DST_PTR_ZERO</td>
<td>X'44C' The caller of DESERV FUNC=EXIT supplied a DST address of zero via the EXIT_DST parameter</td>
</tr>
<tr>
<td>DESRS_INVALID_DST_HEADER</td>
<td>X'44D' The DST header is not correct</td>
</tr>
<tr>
<td>DESRS_INVALID_PREVDST_HEADER</td>
<td>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</td>
</tr>
<tr>
<td>DESRS_PREV_DSTPTR_PTR_ZERO</td>
<td>X'452' The pointer to the previous DST is zero. This is checked for EXIT_OPTION=DELETE</td>
</tr>
<tr>
<td>DESRS_INVALID_PARM_LIST_HEADER</td>
<td>X'411' The id, length, or level of the DESP is not valid</td>
</tr>
<tr>
<td>DESRS_UNSUPPORTED_FUNC</td>
<td>X'424' The FUNC value is incorrect</td>
</tr>
<tr>
<td>DESRS_DEB_REQUIRES_AUTH</td>
<td>X'423' To pass the DEB the caller must be in supervisor state or a privileged key</td>
</tr>
<tr>
<td>DESRS_INVALID_DCB_PTR</td>
<td>X'422' The address of the DCB is 0</td>
</tr>
<tr>
<td>DESRS_DCB_NOT_OPEN</td>
<td>X'421' The passed DCB is not opened</td>
</tr>
<tr>
<td>DESRS_INVALID_DEB_PTR</td>
<td>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</td>
</tr>
<tr>
<td>DESRS_DEBCHK_FAILED</td>
<td>X'41D' The DEBCHK macro failed. The DCB or DEB was not valid</td>
</tr>
<tr>
<td>DESRS_AUTH_ERROR</td>
<td>X'449' Caller not supervisor state or system key</td>
</tr>
<tr>
<td>DESRS_ADD_STACK_FAILED</td>
<td>X'437' Non-zero return code from an IGWFINSTK request</td>
</tr>
<tr>
<td>DESRS_SETLOOK_ERR</td>
<td>X'407' Bad return code from SETLOOK</td>
</tr>
<tr>
<td>DESRS_EXTRACT_ERROR</td>
<td>X'406' IGWFTOKM EXTRACT failed</td>
</tr>
<tr>
<td>DESRS_SET_ERROR</td>
<td>X'405' IGWFTOKM SET failed</td>
</tr>
</tbody>
</table>

**Figure 45. 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
DESERV Exit

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 Table 86.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length or Bit Pattern</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 (X'00')</td>
<td>16</td>
<td>DESD</td>
<td>Member data descriptor (structure)</td>
</tr>
<tr>
<td>00 (X'00')</td>
<td>16</td>
<td>DESD_ENTRY</td>
<td>Entry descriptor (character)</td>
</tr>
<tr>
<td>00 (X'00')</td>
<td>1</td>
<td>DESD_FLAG</td>
<td>Flags (bitstring)</td>
</tr>
<tr>
<td>1 (X'01')</td>
<td>1</td>
<td>DESD_FLAG_ALIAS</td>
<td>Alias entry</td>
</tr>
<tr>
<td>1 (X'01')</td>
<td>1</td>
<td>DESD_CODE</td>
<td>Processing code (error=X'02', not processed =X'01', successful =X'00')(unsigned)</td>
</tr>
<tr>
<td>X'02'</td>
<td>DESD_CODE_ERROR</td>
<td>Constant to be used with DESD_CODE</td>
<td></td>
</tr>
<tr>
<td>X'01'</td>
<td>DESD_CODE_NOGO</td>
<td>Constant to be used with DESD_CODE</td>
<td></td>
</tr>
<tr>
<td>X'00'</td>
<td>DESD_CODE_SUCC</td>
<td>Constant to be used with DESD_CODE</td>
<td></td>
</tr>
<tr>
<td>2 (X'02')</td>
<td>2</td>
<td>DESD_ERRCODE</td>
<td>Error code (low order halfword of DESERV reason code if error) (unsigned)</td>
</tr>
<tr>
<td>4 (X'04')</td>
<td>2</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>6 (X'06')</td>
<td>2</td>
<td>DESD_DATA_LEN</td>
<td>Length of data area (unsigned)</td>
</tr>
<tr>
<td>8 (X'08')</td>
<td>4</td>
<td>DESD_DATA_PTR</td>
<td>Address of data (address)</td>
</tr>
<tr>
<td>12 (X'0C')</td>
<td>4</td>
<td>DESD_NAME_PTR</td>
<td>Address of varying length name (address)</td>
</tr>
</tbody>
</table>

The DESD_NAME_PTR points to the DESN structure. 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. Table 84 on page 383 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 (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. Table 87 shows the fields of the DESP that pertain to the DESERV DELETE invocation.

**Table 87. DESERV DELETE DESP Fields**

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length or Bit Pattern</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 (X'00')</td>
<td>104</td>
<td>DESP</td>
<td>DE Services parameter list (structure)</td>
</tr>
<tr>
<td>00 (X'00')</td>
<td>16</td>
<td>DESP_HEADER</td>
<td>Standard header (character)</td>
</tr>
<tr>
<td>00 (X'00')</td>
<td>8</td>
<td>DESP_ID</td>
<td>Eyecatcher IGWDESP (character)</td>
</tr>
<tr>
<td>08 (X'08')</td>
<td>4</td>
<td>DESP_LEN</td>
<td>Length of DESP (signed)</td>
</tr>
<tr>
<td>12 (X'0C')</td>
<td>1</td>
<td>DESP_LEV</td>
<td>Control block level (character)</td>
</tr>
<tr>
<td>16 (X'10')</td>
<td>1</td>
<td>DESP_FUNC</td>
<td>Function type (GET=X'01', PUT=X'04', DELETE=X'07', RENAME=X'08', UPDATE=X'09')</td>
</tr>
<tr>
<td>17 (X'11')</td>
<td>3</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>20 (X'14')</td>
<td>4</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>24 (X'18')</td>
<td>12</td>
<td>DESP_DATA</td>
<td>Function data (character)</td>
</tr>
<tr>
<td>24 (X'18')</td>
<td>2</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>26 (X'1A')</td>
<td>1</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>27 (X'1B')</td>
<td>1</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>28 (X'1C')</td>
<td>1</td>
<td>DESP_LIBTYPE</td>
<td>Indicates whether a DCB or a DEB in input. (DCB input X'02', DEB input X'01') (unsigned)</td>
</tr>
<tr>
<td>29 (X'1D')</td>
<td>1</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>30 (X'1E')</td>
<td>1</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>31 (X'1F')</td>
<td>1</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>32 (X'20')</td>
<td>1</td>
<td>-</td>
<td>Reserved</td>
</tr>
</tbody>
</table>
Table 87. DESERV DELETE DESP Fields (continued)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length or Bit Pattern</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>33 (X'21')</td>
<td>1</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>34 (X'22')</td>
<td>1</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>35 (X'23')</td>
<td>1</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>36 (X'24')</td>
<td>4</td>
<td>DESP_DCB_PTR</td>
<td>DCB address</td>
</tr>
<tr>
<td>40 (X'28')</td>
<td>4</td>
<td>DESP_DEB_PTR</td>
<td>DEB address</td>
</tr>
<tr>
<td>44 (X'2C')</td>
<td>4</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>48 (X'30')</td>
<td>4</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>52 (X'34')</td>
<td>4</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>56 (X'38')</td>
<td>4</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>60 (X'3C')</td>
<td>4</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>64 (X'40')</td>
<td>4</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>68 (X'44')</td>
<td>4</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>72 (X'48')</td>
<td>4</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>76 (X'4C')</td>
<td>4</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>80 (X'50')</td>
<td>4</td>
<td>DESP_NAME_LIST</td>
<td>List of names to be deleted (DESL) (address)</td>
</tr>
<tr>
<td>84 (X'54')</td>
<td>4</td>
<td>DESP_NAME_LIST2</td>
<td>Number of entries in DESL name list (unsigned)</td>
</tr>
<tr>
<td>88 (X'58')</td>
<td>4</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>92 (X'5C')</td>
<td>4</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>96 (X'60')</td>
<td>4</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>100 (X'64')</td>
<td>4</td>
<td>-</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

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 Table 73 on page 373.

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. Table 88 shows the fields of the DESP that pertain to the DESERV RENAME invocation.

Table 88. DESERV RENAME DESP Fields

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length or Bit Pattern</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 (X'00')</td>
<td>104</td>
<td>DESP</td>
<td>DE Services parameter list (structure)</td>
</tr>
<tr>
<td>00 (X'00')</td>
<td>16</td>
<td>DESP_HEADER</td>
<td>Standard header (character)</td>
</tr>
<tr>
<td>00 (X'00')</td>
<td>8</td>
<td>DESP_ID</td>
<td>Eyecatcher IGWDES (character)</td>
</tr>
<tr>
<td>08 (X'08')</td>
<td>4</td>
<td>DESP_LEN</td>
<td>Length of DESP (signed)</td>
</tr>
<tr>
<td></td>
<td>X'04'</td>
<td>DESP_LEN_IV</td>
<td>Constant to be used with DESP_LEN</td>
</tr>
</tbody>
</table>

DESERV Exit
## Table 88. DESERV RENAME DESP Fields (continued)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length or Bit Pattern</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>12 (X'0C')</td>
<td>1</td>
<td>DESP_LEV</td>
<td>Control block level (character)</td>
</tr>
<tr>
<td>X'04'</td>
<td>DESP_LEV_IV</td>
<td>Constant to be used with DESP_LEV</td>
<td></td>
</tr>
<tr>
<td>13 (X'D')</td>
<td>3</td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>16 (X'10')</td>
<td>1</td>
<td>DESP_FUNC</td>
<td>Function type (GET=X'01', PUT=X'04', DELETE=X'07', RENAME=X'08', UPDATE=X'09')</td>
</tr>
<tr>
<td>X'09'</td>
<td>DESP_FUNC_UPDATE</td>
<td>Constant to be used with DESP_FUNC</td>
<td></td>
</tr>
<tr>
<td>X'08'</td>
<td>DESP_FUNC_RENAME</td>
<td>Constant to be used with DESP_FUNC</td>
<td></td>
</tr>
<tr>
<td>X'07'</td>
<td>DESP_FUNC_DELETE</td>
<td>Constant to be used with DESP_FUNC</td>
<td></td>
</tr>
<tr>
<td>X'04'</td>
<td>DESP_FUNC_PUT</td>
<td>Constant to be used with DESP_FUNC</td>
<td></td>
</tr>
<tr>
<td>X'01'</td>
<td>DESP_FUNC_GET</td>
<td>Constant to be used with DESP_FUNC</td>
<td></td>
</tr>
<tr>
<td>X'00'</td>
<td>DESP_FUNC_OMITTED</td>
<td>Constant to be used with DESP_FUNC</td>
<td></td>
</tr>
<tr>
<td>17 (X'11')</td>
<td>3</td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>20 (X'14')</td>
<td>4</td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>24 (X'18')</td>
<td>12</td>
<td>DESP_DATA</td>
<td>Function data (character)</td>
</tr>
<tr>
<td>24 (X'18')</td>
<td>2</td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>26 (X'1A')</td>
<td>1</td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>27 (X'1B')</td>
<td>1</td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>28 (X'1C')</td>
<td>1</td>
<td>DESP_LIBTYPE</td>
<td>Indicates whether a DCB or a DEB is input. (DCB,DEB) (DCB input X'02', DEB input X'01')</td>
</tr>
<tr>
<td>X'02'</td>
<td>DESP_LIBTYPE_DCB</td>
<td>Constant to be used with DESP_LIBTYPE</td>
<td></td>
</tr>
<tr>
<td>X'01'</td>
<td>DESP_LIBTYPE_DEB</td>
<td>Constant to be used with DESP_LIBTYPE</td>
<td></td>
</tr>
<tr>
<td>X'00'</td>
<td>DESP_LIBTYPE_OMITTED</td>
<td>Constant to be used with DESP_LIBTYPE</td>
<td></td>
</tr>
<tr>
<td>29 (X'1D')</td>
<td>1</td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>30 (X'1E')</td>
<td>1</td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>31 (X'1F')</td>
<td>1</td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>32 (X'20')</td>
<td>1</td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>33 (X'21')</td>
<td>1</td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>34 (X'22')</td>
<td>1</td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>35 (X'23')</td>
<td>1</td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>36 (X'24')</td>
<td>4</td>
<td>DESP_DCB_PTR</td>
<td>DCB address</td>
</tr>
<tr>
<td>40 (X'28')</td>
<td>4</td>
<td>DESP_DEB_PTR</td>
<td>DEB address</td>
</tr>
<tr>
<td>44 (X'2C')</td>
<td>4</td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>48 (X'30')</td>
<td>4</td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>52 (X'34')</td>
<td>4</td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>56 (X'38')</td>
<td>4</td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>60 (X'3C')</td>
<td>4</td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>64 (X'40')</td>
<td>4</td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>68 (X'44')</td>
<td>4</td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>72 (X'48')</td>
<td>4</td>
<td></td>
<td>Reserved</td>
</tr>
</tbody>
</table>
DESERV Exit

Table 88. DESERV RENAME DESP Fields (continued)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length or Bit Pattern</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>76 (X'4C')</td>
<td>4</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>80 (X'50')</td>
<td>4</td>
<td>DESP_NAME_LIST</td>
<td>List of pairs of names for rename operations (DESL) (address)</td>
</tr>
<tr>
<td>84 (X'54')</td>
<td>4</td>
<td>DESP_NAME_LIST2</td>
<td>Number of entries in DESL name list (unsigned)</td>
</tr>
<tr>
<td>88 (X'58')</td>
<td>4</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>92 (X'5C')</td>
<td>4</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>96 (X'60')</td>
<td>4</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>100 (X'64')</td>
<td>4</td>
<td>Reserved</td>
<td></td>
</tr>
</tbody>
</table>

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 Table 73 on page 375.

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. Table 89 shows the fields of the DESP that pertain to the DESERV UPDATE invocation.

Table 89. DESERV UPDATE DESP Fields

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length or Bit Pattern</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 (X'00')</td>
<td>104</td>
<td>DESP</td>
<td>DE Services parameter list (structure)</td>
</tr>
<tr>
<td>00 (X'00')</td>
<td>16</td>
<td>DESP_HEADER</td>
<td>Standard header (character)</td>
</tr>
<tr>
<td>00 (X'00')</td>
<td>8</td>
<td>DESP_ID</td>
<td>Eyecatcher 'IGWDES' (character)</td>
</tr>
<tr>
<td>08 (X'08')</td>
<td>4</td>
<td>DESP_LEN</td>
<td>Length of DESP (signed)</td>
</tr>
<tr>
<td>X'04'</td>
<td>DESP_LEN_IV</td>
<td>Constant to be used with DESP_LEN</td>
<td></td>
</tr>
<tr>
<td>12 (X'0C')</td>
<td>1</td>
<td>DESP_LEV</td>
<td>Control block level (character)</td>
</tr>
<tr>
<td>X'04'</td>
<td>DESP_LEV_IV</td>
<td>Constant to be used with DESP_LEV</td>
<td></td>
</tr>
<tr>
<td>13 (X'0D')</td>
<td>3</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>16 (X'10')</td>
<td>1</td>
<td>DESP_FUNC</td>
<td>Function type (GET=X'01', PUT=X'04', DELETE=X'07', RENAME=X'08', UPDATE=X'09')</td>
</tr>
<tr>
<td>X'09'</td>
<td>DESP_FUNC_UPDATE</td>
<td>Constant to be used with DESP_FUNC</td>
<td></td>
</tr>
<tr>
<td>X'08'</td>
<td>DESP_FUNC_RENAME</td>
<td>Constant to be used with DESP_FUNC</td>
<td></td>
</tr>
<tr>
<td>X'07'</td>
<td>DESP_FUNC_DELETE</td>
<td>Constant to be used with DESP_FUNC</td>
<td></td>
</tr>
<tr>
<td>X'04'</td>
<td>DESP_FUNC_PUT</td>
<td>Constant to be used with DESP_FUNC</td>
<td></td>
</tr>
<tr>
<td>X'01'</td>
<td>DESP_FUNC_GET</td>
<td>Constant to be used with DESP_FUNC</td>
<td></td>
</tr>
<tr>
<td>X'00'</td>
<td>DESP_FUNC_OMITTED</td>
<td>Constant to be used with DESP_FUNC</td>
<td></td>
</tr>
<tr>
<td>17 (X'11')</td>
<td>3</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>20 (X'14')</td>
<td>4</td>
<td>-</td>
<td>Reserved</td>
</tr>
</tbody>
</table>
### Table 89. DESERV UPDATE DESP Fields (continued)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length or Bit Pattern</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>24 (X'18')</td>
<td>12</td>
<td>DESP_DATA</td>
<td>Function data (character)</td>
</tr>
<tr>
<td>24 (X'18')</td>
<td>2</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>26 (X'1A')</td>
<td>1</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>27 (X'1B')</td>
<td>1</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>28 (X'1C')</td>
<td>1</td>
<td>DESP_LIBTYPE</td>
<td>Function subtype (DCB, DEB) (DCB input X'02', DEB input X'01') (unsigned)</td>
</tr>
<tr>
<td></td>
<td>X'02'</td>
<td>DESP_LIBTYPE_DCB</td>
<td>Constant to be used with DESP_LIBTYPE</td>
</tr>
<tr>
<td></td>
<td>X'01'</td>
<td>DESP_LIBTYPE_DEB</td>
<td>Constant to be used with DESP_LIBTYPE</td>
</tr>
<tr>
<td></td>
<td>X'00'</td>
<td>DESP_LIBTYPE_OMITTED</td>
<td>Constant to be used with DESP_LIBTYPE</td>
</tr>
<tr>
<td>29 (X'1D')</td>
<td>1</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>30 (X'1E')</td>
<td>1</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>31 (X'1F')</td>
<td>1</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>32 (X'20')</td>
<td>1</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>33 (X'21')</td>
<td>1</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>34 (X'22')</td>
<td>1</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>35 (X'23')</td>
<td>1</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>36 (X'24')</td>
<td>4</td>
<td>DESP_DCB_PTR</td>
<td>DCB address</td>
</tr>
<tr>
<td>40 (X'28')</td>
<td>4</td>
<td>DESP_DEB_PTR</td>
<td>DEB address</td>
</tr>
<tr>
<td>44 (X'2C')</td>
<td>4</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>48 (X'30')</td>
<td>4</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>52 (X'34')</td>
<td>4</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>56 (X'38')</td>
<td>4</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>60 (X'3C')</td>
<td>4</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>64 (X'40')</td>
<td>4</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>68 (X'44')</td>
<td>4</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>72 (X'48')</td>
<td>4</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>76 (X'4C')</td>
<td>4</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>80 (X'50')</td>
<td>4</td>
<td>DESP_NAME_LIST</td>
<td>List of SMDEs with imbedded PMAR (and PMARL) to update the directory information (DESL) (address)</td>
</tr>
<tr>
<td>84 (X'54')</td>
<td>4</td>
<td>DESP_NAME_LIST2</td>
<td>Number of entries in DESL name list (unsigned)</td>
</tr>
<tr>
<td>88 (X'58')</td>
<td>4</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>92 (X'5C')</td>
<td>4</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>96 (X'60')</td>
<td>4</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>100 (X'64')</td>
<td>4</td>
<td>-</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

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 Table 73 on page 375.
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.
   •  For post-processing exit:
DESERV Exit

R15 = 0
Continue processing of this DESERV call (that is, 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 (that is, system diagnostic information). The low order two bytes contain the real reason code as indicated in the macro IGWDES. For additional return and reason codes for the GET, RENAME, DELETE, and UPDATE return and reason codes, refer to z/OS DFSMS Macro Instructions for Data Sets.

- **Restriction**: 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:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>not applicable</td>
</tr>
<tr>
<td>1</td>
<td>address of DESX</td>
</tr>
<tr>
<td>2-12</td>
<td>not applicable</td>
</tr>
<tr>
<td>13</td>
<td>address of register save area</td>
</tr>
<tr>
<td>14</td>
<td>return address</td>
</tr>
<tr>
<td>15</td>
<td>address of DESERV exit entry point</td>
</tr>
</tbody>
</table>

**Registers on Return from the DESERV Exit**

When you return control to DESERV, the register contents must be set up as follows:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-14</td>
<td>restored to contents at entry</td>
</tr>
<tr>
<td>15</td>
<td>return code</td>
</tr>
</tbody>
</table>

**DESERV Exit Return and Reason Codes**

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 (X'00')</td>
<td>Continue with DESERV function</td>
</tr>
<tr>
<td>04 (X'04')</td>
<td>Return immediately to the DESERV caller and return the return and reason codes as defined by DESX_RETURN_CODE and DESX_REASON_CODE respectively.</td>
</tr>
</tbody>
</table>

**DESERV FUNC=EXIT Return and Reason Codes**

The formats of the return and reason codes are:
DESERV Exit

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).

### Table 90. Return and Reason Codes for the Exit DESERV Function

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Symbolic name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'0'</td>
<td></td>
<td>DESRC_SUCC</td>
<td>Successful</td>
</tr>
<tr>
<td>X'00'</td>
<td></td>
<td>DESRS_SUCC</td>
<td>Successful</td>
</tr>
<tr>
<td>X'4'</td>
<td></td>
<td>DESRC_INFO</td>
<td>Not completely successful.</td>
</tr>
<tr>
<td>X'400'</td>
<td></td>
<td>DESRS_NAME_ NOT_DEFINED</td>
<td>Name to be replaced did not previously exist</td>
</tr>
<tr>
<td>X'8'</td>
<td></td>
<td>DESRC_WARN</td>
<td>Results questionable</td>
</tr>
<tr>
<td>X'12'</td>
<td></td>
<td>DESRC_PARM</td>
<td>Missing or invalid parameters</td>
</tr>
<tr>
<td>X'411'</td>
<td></td>
<td>DESRS_INVALID_ PARM_LIST_HEADER</td>
<td>The id, length, or level of the DESP is not valid</td>
</tr>
<tr>
<td>X'415'</td>
<td></td>
<td>DESRS_PDS_ NOT_SUPPORTED</td>
<td>This function requires a PDSE data set</td>
</tr>
<tr>
<td>X'41E'</td>
<td></td>
<td>DESRS_INVALID_ DEB_PTR</td>
<td>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</td>
</tr>
<tr>
<td>X'41F'</td>
<td></td>
<td>DESRS_DCB_NOT_ OPEN_OUTPUT</td>
<td>With function PUT the DCB must have been opened for output</td>
</tr>
<tr>
<td>X'421'</td>
<td></td>
<td>DESRS_DCB_ NOT_OPEN</td>
<td>The passed DCB is not opened</td>
</tr>
<tr>
<td>X'422'</td>
<td></td>
<td>DESRS_INVALID_ DCB_PTR</td>
<td>The address of the DCB is 0</td>
</tr>
<tr>
<td>X'423'</td>
<td></td>
<td>DESRS_DEB_ REQUIRES_AUTH</td>
<td>To pass the DEB, the caller must be in supervisor state or a privileged key</td>
</tr>
<tr>
<td>X'424'</td>
<td></td>
<td>DESRS_ UNSUPPORTED_FUNC</td>
<td>The FUNC value is incorrect</td>
</tr>
<tr>
<td>X'427'</td>
<td></td>
<td>DESRS_INVALID_ MEM_DATA_CNT</td>
<td>The count of entries in the MEM_DATA block is 0</td>
</tr>
<tr>
<td>X'428'</td>
<td></td>
<td>DESRSINVALID_ MEM_DATA_PTR</td>
<td>The address of the MEM_DATA block is 0</td>
</tr>
</tbody>
</table>
### Table 90. Return and Reason Codes for the Exit DESERV Function (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Symbolic name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'429'</td>
<td>DESRS_INVALID_PUT_OPTION</td>
<td>The PUT function requires that the OPTION field be specified</td>
<td></td>
</tr>
<tr>
<td>X'16'</td>
<td>DESRC_CALR</td>
<td>Caller has a problem</td>
<td></td>
</tr>
<tr>
<td>X'3FD'</td>
<td>DESRS_PRI_NM_THIS_FILE</td>
<td>Alias name is same name as primary name for this member</td>
<td></td>
</tr>
<tr>
<td>X'400'</td>
<td>DESRS_NAME_ALREADY_EXISTS</td>
<td>Name to be replaced did not previously exist</td>
<td></td>
</tr>
<tr>
<td>X'40E'</td>
<td>DESRS_NAME_ALREADY_EXISTS</td>
<td>The PUT failed because of a name conflict</td>
<td></td>
</tr>
<tr>
<td>X'40F'</td>
<td>DESRS_NAME_PRIMARY_NAME</td>
<td>The MEM_DATA must have one member designated as primary</td>
<td></td>
</tr>
<tr>
<td>X'410'</td>
<td>DESRS_INVALID_NAME_PREFIX</td>
<td>The first 8 bytes of a name were all X'FF'</td>
<td></td>
</tr>
<tr>
<td>X'412'</td>
<td>DESRS_MORE_THAN_1_PRIMARY</td>
<td>The MEM_DATA must have only one member designated as primary</td>
<td></td>
</tr>
<tr>
<td>X'413'</td>
<td>DESRS_INVALID_MLT</td>
<td>MLT is not valid</td>
<td></td>
</tr>
<tr>
<td>X'414'</td>
<td>DESRS_INVALID_CT</td>
<td>Connect token is not valid</td>
<td></td>
</tr>
<tr>
<td>X'41D'</td>
<td>DESRS_DEBCHK_FAILED</td>
<td>The DEBCHK macro failed, the DCB or DEB was not valid</td>
<td></td>
</tr>
<tr>
<td>X'425'</td>
<td>DESRS_INVALID_NAME_LENGTH</td>
<td>The length of an alias name was either 0 or greater than 8</td>
<td></td>
</tr>
<tr>
<td>X'43A'</td>
<td>DESRS_DATA_LENGTH_ERROR</td>
<td>The DESD data length is not valid, data length must be greater than 0 and less than 108 bytes</td>
<td></td>
</tr>
<tr>
<td>X'43C'</td>
<td>DESRS_NAME_IS_PRIMARY_NAME</td>
<td>The alias name specified is a primary name and the options did not allow for deleting primary name</td>
<td></td>
</tr>
<tr>
<td>X'20'</td>
<td>DESRC_ENVR</td>
<td>Resources unavailable</td>
<td></td>
</tr>
<tr>
<td>X'24'</td>
<td>DESRC_IOER</td>
<td>I/O error</td>
<td></td>
</tr>
<tr>
<td>X'28'</td>
<td>DESRC_MEDE</td>
<td>Media Error</td>
<td></td>
</tr>
<tr>
<td>X'32'</td>
<td>DESRC_DSLE</td>
<td>Data Set logical error</td>
<td></td>
</tr>
<tr>
<td>X'36'</td>
<td>DESRC_SEVE</td>
<td>Severe error</td>
<td></td>
</tr>
<tr>
<td>X'407'</td>
<td>DESRS_SETLOOK_ERR</td>
<td>Bad return code from SETLOOK</td>
<td></td>
</tr>
<tr>
<td>X'437'</td>
<td>DESRS_ADD_STACK_FAILED</td>
<td>Non-zero return code from an IGWFESTK request</td>
<td></td>
</tr>
</tbody>
</table>
### DESERV Exit

**Table 90. Return and Reason Codes for the Exit DESERV Function (continued)**

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Symbolic name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'447'</td>
<td></td>
<td>DESRS_UNKNOWN</td>
<td>Issued by the DESERV recovery routine when entered for an unknown reason (for example, a program check) while the exit routine was in control. Most likely an exit error.</td>
</tr>
<tr>
<td>X'0C'</td>
<td>X'469'</td>
<td>DESRS_DST_EXIT_PTR_NOT</td>
<td>Global exit address is not in common storage.</td>
</tr>
</tbody>
</table>

**Additional Return and Reason Codes:** For the GET, RENAME, DELETE, and UPDATE return and reason codes, refer to [z/OS DFSMS 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.
DESERV Exit

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 THIS 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 STATE
* 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 46. Establishing and Deleting a Task Level DESERV Exit Part 1 of 2
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. Note that this sample exit is not reentrant, so it assumes there is only one subtask.

Figure 47. Establishing and Deleting a Task Level DESERV Exit Part 2 of 2
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 (that is, 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 NOPREVDEST 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 48. Sample DESERV Exit Routine Part 1 of 3
DESERV Exit

NOREV DST 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
* post-processing exit call.
    L 6,DESX_DESP_PTR get the DESERV parameter list (DESP)
    USING DESP,6 map DESP
    CLI DESP_FUNC,DESX 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
    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 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 49. Sample DESERV Exit Routine Part 2 of 3
* 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 GETMAIN 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
IGWIDES DESERV MAPPINGS
IGWSMDE DIRECTORY ENTRY
IEWPMAR PROGRAM MANAGEMENT ATTRIBUTE RECORD
* THE PMAR IS A SUB RECORD OF THE SMDE
END

Figure 50. Sample DESERV Exit Routine Part 3 of 3
Chapter 11. Managing Hierarchical File System Data Sets

A hierarchical file system (HFS) data set is a data set that contains a POSIX-compliant hierarchical file system, which is a collection of files and directories organized in a hierarchical structure that can be accessed using z/OS UNIX system services. You can use many of the standard BSAM, QSAM, BPAM, and VSAM interfaces to access data in z/OS UNIX HFS files. Most applications that use these access methods can access HFS files without reassembly or recompliation.

The contents of HFS data sets are structured like a tree, based on a root directory with various subdirectories. The files within an HFS data set are identified by their path and file names.

DFSMShsm can automatically back up HFS data sets if it is using DFSMSdss as its data mover, but it cannot back up individual files within an HFS data set.

Once you evaluate z/OS UNIX and decide to install it in your enterprise, you may proceed with the following planning tasks:

- Deciding which DASD devices will contain the HFS data sets
- Deciding how to control access to them
- Structuring the file systems
- Determining backup, restore, and expiration date policies
- Determining HFS naming policies (file names can be up to 255 characters long and path names can be up to 1023 characters)
- Defining data classes, management classes, and coding ACS routines and JCL statements for HFS data sets

Creating Hierarchical File System Data Sets

You specify HFS in the DSNTYPE parameter to allocate an HFS data set. You can also define a data class for HFS data sets. Both cataloged and uncataloged HFS data sets can reside on a single non-SMS-managed volume. A cataloged HFS data set can also be a multivolume if it resides on Storage Management Subsystem managed volumes. This type of data set can expand to as many as 255 extents of direct access storage device (DASD) space on multiple volumes (59 volumes maximum with 123 extents per volume).

RACF or an equivalent security product must be installed and active on your system to use z/OS UNIX or HFS data sets. z/OS UNIX maintains system security by verifying user identities and file access control information.

This information covers the following:

- “Defining the Root File System” on page 406
- “Creating and Mounting the Root File System” on page 406
- “Creating Additional File Systems and Directories” on page 406
- “Adding and Mounting File Systems to the Root File System” on page 407
Defining the Root File System

During installation of z/OS UNIX, a system programmer or storage administrator codes the ROOT statement in the BPXPRMxx member of SYS1.PARMLIB. This identifies the HFS data set containing the root file system for the system to logically mount when it starts z/OS UNIX system services.

```
ROOT	FILESYSTEM('OMVS.ROOT')
    TYPE(HFS)
    MODE(RDWR)
```

The root file system is the starting point of the overall file structure. It consists of the root directory and any related HFS files or subdirectories. After installation is complete and the MVS system is running, you can create (allocate) an HFS data set, which contains the root file system.

See z/OS MVS Initialization and Tuning Reference for information about member BPXPRMxx.

Creating and Mounting the Root File System

Create an HFS data set to contain the root file system by running a job to allocate the data set. During allocation, z/OS UNIX builds a basic root directory, which you can alter to meet your specific needs. You specify DSNTYPE=HFS to designate an HFS data set.

```
//STEP1 EXEC PGM=IEFBR14
//MKFS DD DSN=OMVS.ROOT,DISP=(NEW,CATLG),
//    DSNTYPE=HFS,SPACE=(CYL,(1,1,1))
```

Tip: To make the HFS data set SMS-managed, use the ACS routines or specify the STORCLAS parameter in the JCL. HFS data sets do not have to be SMS-managed.

Write operations present the greatest exposure to file system damage. For this reason the root file system should be small, minimizing the amount of write activity to it, thus offering the least exposure to damage.

Additionally, if all users' files are in file systems that are mounted on the root file system, rather than defined as part of the root itself, and users are denied write access to the root, the root is further protected from inadvertent damage. Damaged user directories or files can be unmounted and replaced without causing z/OS UNIX system services to fail.

Creating Additional File Systems and Directories

After allocating an HFS data set for the root file system and logging on as a TSO/E user, you can define additional directories in the root file system using the MKDIR command. For example, to create the /u/joe directory, issue:

```
MKDIR '/u'
MKDIR '/u/joe'
```

These directories can be used as mount points for additional mountable file systems. You can use an IBM-supplied program that creates directories, pseudo-TTY pairs, and device files. Interactive users and application programs can then add files to those additional file systems.

The MKDIR command requires written permission on the parent directory of the directory to be created.
Adding and Mounting File Systems to the Root File System

If you have appropriate authority, you can create other mountable file systems with their own directory and data file structures, and mount them on a directory in the root file system or in another file system. Each file system can be logically mounted to a directory (mount point) in another file system by using the TSO/E MOUNT command. Use the UNMOUNT command to unmount a file system. To create and mount additional file systems:

1. Allocate an additional HFS data set by using either the TSO/E ALLOCATE command as shown in the following example or a JCL DD statement similar to that shown in "Creating and Mounting the Root File System" on page 406.

   ALLOCATE DNAME('OMVS.USER.JOE') NEW DSNTYPE(HFS) BLKSIZE(0)
   LRECL(0) RECFM(U) DSORG(P0) SPACE(1,1) CYLINDERS

   The new data set is allocated with an empty root directory.

2. Have an authorized user enter a TSO/E MOUNT command to logically mount the new file system in the directory of an existing file system.

   MOUNT FILESYSTEM('OMVS.USER.JOE') TYPE(HFS) MOUNTPOINT('/u/joe')

You can specify additional file systems to be logically mounted automatically every time z/OS UNIX is started by adding MOUNT commands to the BPXPRMxx member of SYS1.PARMLIB. The following restrictions apply to mounting file systems:

- The mount point must be a directory.
- Any files in the directory are not accessible while the file system is mounted.
- Only one mount can be active at any time for a mount point.
- A file system can be mounted on only one directory at a time.

You can also create special HFS files to perform the following tasks:

- Represent hardware devices (character special files).
- Allow the use of alias names for HFS files (symbolic links).
- Send data from one process to another so that the receiving process reads the data first-in-first-out (FIFO special files, also called named pipes). The term process as used here is defined as either a program that is created by the fork function, or a program that requests z/OS UNIX system services.

Managing File System Size

File system size increases as users add files and extend existing files. Eventually, a file system can outgrow the space on its volume. In this case, the storage administrator or system programmer responsible for HFS data sets can either make more space available on the volume by moving individual HFS files to other file systems that have space available, or do one of the following:

- Move the entire file system to another set of volumes as follows:
  1. Have an authorized user enter a TSO/E UNMOUNT command to logically unmount the file system.
  2. Allocate an HFS data set with a different data set name on a volume, or set of volumes, that has adequate space available.
  3. Use the DFSMSdss DUMP function to logically dump the old file system.
  4. Use the DFSMSdss RESTORE function to restore the dumped file system with a new name to a volume, or set of volumes, that has sufficient space. If you want to maintain the original file system name, delete the existing file system first, and then restore it using DFSMSdss without renaming it.
Planning for HFS

- Remove files from the file system by either deleting them or by moving them to another file system. If it is impossible to remove the chosen files from a particular directory in the file system, it may be possible to remove other files from a different directory in the same file system. The objective is to reduce the size of the file system.
- Create a new file system on another volume, or set of volumes, and move some files from the full file system to the new file system. To avoid problems that can result from this approach, define symbolic links using the original names.
- Add another volume to the file system candidate volume list with the IDCAMS ALTER ADDVOLUMES command. The file system must be unmounted and remounted for the additional volumes to be usable by the HFS.
- The storage administrator or system programmer can monitor the space in a file system by mounting an HFS with parm FSFULL. For example, mount parm('FSFULL(70,10)') will cause HFS to issue message IGW023A when the file system is 70 percent full and then issues an additional IGW023I messages when the file system is 80 and 90 percent full.

The BPX1PCT callable service can be used to extend the file system (see “Using pfctl (BPX1PCT) Physical File System Control for HFS” on page 411).

Managing File System Activity

If activity for a file system becomes so extensive that accesses are slow, you can do one of the following tasks:
- Move the file system to a volume that processes I/O more quickly, for example, a volume that has a faster channel or a cached storage control.
- Move a subtree from the full file system into a new file system on a different volume. Mount the new file system on the now-empty directory, which was the head of the subtree. This divides I/O activity between two volumes. To avoid failures from this action, define symbolic links using the original names.

Accessing HFS Data Set Attributes

Any application (for example, Interactive System Productivity Facility (ISPF) option 3.4I or DCOLLECT) that requests size information about HFS file system requires OMVS to service that request. This means that the user ID must be defined to OMVS and have Resource Access Control Facility (RACF) authority to access the data set. The HFS file system must be either mounted on this file system or mountable (that is, not already mounted READ/WRITE on another system).

Transporting a File System

You might also want to copy a data set to a storage medium that can be physically transported to another location. In order to do that, perform one of the following tasks:
- Use either the PAX, CPIO, or TAR shell commands to copy the file system in tape archive (TAR) format.
- Have an authorized user logically unmount the file system, allocate an HFS data set with a different data set name, and use the DFSMSdss dump utility to copy the old file system to the new data set.
Removing (Deleting) a File System

If you mount the file system (HFS data set) to a different file system or you want
to delete it, first logically unmount it using the TSO/E UNMOUNT command
against the HFS data set containing it. Then you can use either or both of the
MOUNT commands (see “Adding and Mounting File Systems to the Root File
System” on page 407).

You can eliminate the file system by using any of the following techniques:
• Use the DELETE command (IDCAMS or TSO).
• Execute an IEFBR14 job with DISP=(OLD,DELETE) specified for the HFS data
  set.
• ISPF option 3.x.
• Use the SCRATCH and UNCATLG commands of IEHPROGM.

Migrating a File System

If a file system is unmounted and remains so for a predetermined time, the system
can migrate it to a lower priority storage medium. The system automatically recalls
a migrated file system from migration storage if a mount command is issued for
the file system.

If you plan to migrate HFS data sets, consider migrating them only to level 1
(DASD) storage. Recalling a data set that was migrated to tape could adversely
affect performance because of the time required to physically mount the tape
volume.

If the tape is in an automated library, then recalling it should be much faster than
if the system has to request an operator to mount it.

Backing Up File Systems

DFSMShsm provides automatic backup facilities for HFS data sets. You can back
up mountable file systems by periodically backing up the HFS data sets that
contain them; the data sets can be restored if necessary. DFSMShsm is also used for
migrating and restoring unmounted file systems.

You can manually back up a mountable file system, including the root file system,
periodically with DFSMSdss data set dumps. To do this, issue the DFSMSdss
DUMP command. This quiesces activity against the specified HFS data set, then
invokes backup processing. When the backup is complete, the file system is
unquiesced and user activity can resume.

Retain periodic DFSMSdss data set dumps of file systems in case a program fails
and damages files and directories. Keep this backup in another area or different
building in case the computer site experiences physical damage.

There is no facility for automatically backing up individual files within an HFS
data set. You can manually back up files with the PAX, CPIO, and TAR commands.

For more information on using DFSMSdss, refer to the DFSMSdss section of z/OS
DFSMShsm Storage Administration.
Recovering a Backed-Up File System

If a file system is damaged, you can recover by replacing it with a saved version that was created from an earlier backup. To recover a backed up file system, you must perform the following tasks:

1. Notify all users to stop all activity on the damaged file system.
2. Have an authorized user enter the TSO/E UNMOUNT command with the IMMEDIATE option to logically unmount the damaged file system. If the unmount fails, reenter the UNMOUNT command with the FORCE option.
3. Use the DFSMSdss dump utility to restore the backed up file system to a replacement file system (HFS data set). You can do this simultaneously with the previous steps.
4. Have an authorized user enter a TSO/E MOUNT command to logically mount the replacement file system. Ensure that the MOUNT commands in the BPXPRMxx member in SYS1.PARMLIB are consistent with this MOUNT command.
5. Issue a broadcast message to all users or a message to all z/OS UNIX users when they invoke the shell, telling them that you have mounted a back-level file system and informing them of the mount point. Users must recreate and add any files added since the file system was backed up.

You might choose a more disruptive method to recover a backed up file system for the root file system. Refer to z/OS UNIX System Services Planning, “Shutting Down z/OS Unix”, for more information on how to perform a shutdown.

HFS Deferred File System Synchronization

Normal HFS disk hardening executes under a sync daemon that runs periodically to write out all file and metadata changes that have occurred since the last time the sync daemon ran. If a large number of files are created, modified, or deleted within the time span that is between the file system sync intervals, then with the next file system sync, these file changes are collected and the disk version of the file system is sync'ed with a greatly reduced number of media manager calls. This reduces the actual I/O activity.

The reduced I/O can have a dramatic effect on performance, because the default sync daemon interval is one minute. The sync daemon runs in the OMVS address space and is independent of any user request to the file system.

When the sync daemon runs, all the HFS changes are batched into one large I/O request that gets passed to the media manager. The intent is to perform one long I/O operation to the HFS on disk. However, even with optimal conditions, where the HFS resides on one volume and is contained within a single extent, it is necessary to have a few media manager calls during the disk hardening I/O operation. If the HFS has multiple extents or resides on multiple volumes, multiple channel programs must be built, because a single I/O operation cannot span multiple extents.

In addition to the sync daemon interval, HFS also supports the individual file sync (fsync). The fsync can be performed at any time by the application program. However, be aware that an fsync will result in the entire file system being sync'ed. It is very important to note that during the HFS file system sync operation, the sync task runs independently of the users that are currently accessing the HFS. When the sync task runs, it obtains an exclusive latch for the file system being sync'ed. This latch is held for the duration of the file system sync operation. While
this latch is held UNIX system services users of the HFS file system will not be able to access the HFS. When the sync task runs, the HFS must:
1. Update all of the new or changed files' metadata
2. Delete structures for any removed files
3. Update the internal HFS storage maps within the HFS attributes directory
4. Call the media manager to perform the I/O operation that makes the changes to the disk version of the HFS data set.

The I/O operation is synchronous, which means that HFS sync task will be suspended for the duration of the media manager call. Normally, this file system sync lockout condition is brief, lasting for a few seconds, and is rarely noticed by the end user. However, in some situations the file system sync lockout could last much longer. The duration of the lockout depends on the sync interval, the amount of work that must be performed by the SYNC task, and the performance of the I/O subsystem.

To minimize the effect on applications from the file system sync operations that take a long time to complete, you can do one or more of the following:
1. Mount the affected HFS with a shorter sync interval. This will reduce overall HFS performance, but the sync lockout will not be as long.
2. Split the HFS into two or more smaller HFS data sets.
3. Avoid high impact commands such as “rm -R” on large directories.

How to specify a SYNC value

The default HFS SYNC daemon interval value is 60 seconds. You may override this default by changing the SYNCDEFAULT option of the FILESYSTYPE parameter in the BPXPRMxx member of SYS1.PARMLIB. Individual HFS data sets can have an overriding SYNC value. This can be accomplished by specifying a SYNC value on the ROOT parameter, or a SYNC value on the MOUNT parameter in the BPXPRMxx member of SYS1.PARMLIB.

Refer to IBM z/OS MVS Initialization and Tuning Reference for details on how to specify FILESYSTYPE, ROOT, and MOUNT parameters. Additionally, individual HFS data sets can be mounted by the user with the MOUNT command that specifies a SYNC value.

Using pfsctl (BPX1PCT) Physical File System Control for HFS

The following information describes the use of pfsctl, BPX1PCT callable service, for HFS.

The pfsctl callable service, BPX1PCT, conveys a command and an argument to a physical file system. The meaning of the command and argument are specific to the physical file system and are defined by the physical file system.

Format:

```
CALL BPX1PCT,(File_System_type,
              Command,
              Argument_length,
              Argument,
              Return_value,
              Return_code,
              Reason_code)
```

File_System_type:
Planning for HFS

**Type:** Character string  
**Length:** 8 bytes  
**Character string:** HFS

**Command:**

- **Type:** Integer  
- **Length:** Fullword

If the physical file system is an HFS data set, you can code one of these values:

- X'40000001' DisplayBufferLimits
- X'00000002' ChangeBufferLimits
- X'40000003' DisplayGlobalStats
- X'40000004' DisplayFSStats
- X'00000005' ExtendFS

These HFS-defined commands and the data areas for each are defined in GFUMPCTL, located in SYS1.MODGEN.

**Argument_Length:** A 4-byte integer. See below for the valid values.

**Argument:** Structure as described below and in Table 92 on page 416, Table 93 on page 417, Table 94 on page 418, or Table 95 on page 419.

**Return Value:** Fullword

**Return_Code:** A 4-byte integer that BPX1PCT sets to one of the values described in Table 91 on page 415.

**Reason_Code:** A 4-byte integer that BPX1PCT sets to one of the values described in Table 91 on page 415.

If the physical file system is an HFS data set, then the following apply:

- "DisplayBufferLimits Command"
- "Output from TRKCALC” on page 315
- "ChangeBufferLimits Command” on page 413
- "DisplayGlobalStats Command” on page 413
- "DisplayFSStats Command” on page 414
- "BPX1PCT Return and Reason Codes” on page 415

**DisplayBufferLimits Command**

**Function:**

The DisplayBufferLimits command returns the storage limits in the data area, specifically the VMAX and FMIN values for the HFS buffers currently in effect.

**Command:**

X'40000001'

**Argument_Length:**

The argument length is the length of the data area. The data area must be at least the length of PCTL_BFRLIMITS_TYPE. See Table 92 on page 416.

**Argument:**

PCTL_BFRLIMITS_TYPE
Usage Notes
1. The caller does not have to be authorized.
2. VMAX and FMIN values are both in megabytes (MB).
3. This command is equivalent to the `confighfs` shell command:
   ```
   confighfs -l
   ```

ChangeBufferLimits Command

Function:
The ChangeBufferLimits command modifies the storage limits, specifically the VMAX and FMIN values, for HFS buffers as requested in the data area.

Command:
```X'00000002'
```

Argument_length:
The argument length is the length of the data area. The data area must be at least the length of PCTL_BFRLIMITS_TYPE. See Table 92 on page 416.

Argument:
```PCTL_BFRLIMITS_TYPE```

Usage Notes
1. The following fields must be set within this structure:
   - The PCTL_BL_ACTION flag must be set.
   - Modify VMAX by setting the PCTL_BL_VMAX flag and specifying the new VMAX value in the field PCTL_BL_VMAX_VAL.
   - Modify FMIN by setting the PCTL_BL_FMIN flag and specifying the new FMIN value in the field PCTL_BL_FMIN_VAL.
2. The caller must be a superuser.
3. A request to modify both VMAX and FMIN can be made in one call.
4. The VMAX and FMIN values are in megabytes (MB).
5. This command is equivalent to the `confighfs` shell commands. The following `confighfs` command sets virtual storage maximum (VMAX) to n, where n is in MB:
   ```
   confighfs -v n
   ```

   The following `confighfs` command sets virtual storage minimum (FMIN) to n, where n is in MB:
   ```
   confighfs -f n
   ```

DisplayGlobalStats Command

Function:
The DisplayGlobalStats command returns HFS global system level statistics in the data area.

Command:
```X'40000003'
```

Argument_Length:
The argument length is the length of the data area. The data area must be at least the length of PCTL_GLOBALSTATS_TYPE. See Table 93 on page 417.
Planning for HFS

Argument:
  PCTL_GLOBALSTATS_TYPE

Usage Notes
1. The caller does not have to be authorized.
2. This command is equivalent to the command:
   \texttt{confighfs -q}

DisplayFSStats Command

Function:
The DisplayFSStats command returns HFS file system level statistics in the data area. The values returned for file system statistics are associated with the current mount of the specified file system.

Command:
X'40000004'

Argument Length:
The argument length is the length of the data area. The data area must be the length of PCTL_FSSTATS_TYPE. See Table 94 on page 418.

Argument:
PCTL_FSSTATS_TYPE

Usage Notes
1. The file system name must be supplied as input in the field PCTL_FS_NAME within the PCTL_FSSTATS_TYPE structure.
2. The caller does not have to be authorized.
3. This command is equivalent to the confighfs shell command:
   \texttt{confighfs pathname}
   where the pathname is an absolute or relative pathname that identifies the HFS.

ExtendFS Command

Function:
The ExtendFS command attempts to extend (allocate additional disk space for) the specified file system by the requested amount as specified in the data area.

Command:
X'00000005'

Argument Length:
The argument length is the length of the data area. The length of the data area must be at least the length of PCTL_EXTENDFS_TYPE. See Table 95 on page 419.

Argument:
PCTL_EXTENDFS_TYPE

Usage Notes
- The following fields must be set within this structure:
  - Set PCTL_EXT_FSNNAME to the name of the file system to be extended.
  - If your request is to allocate space on a new volume, set the PCTL_EXT_NEW_VOL flag on.
  - Set PCTL_EXT_UNIT to one of the following units to extend:
1. ExtendFS_UNIT_MB to indicate that the amount to extend is in megabytes (MB).
2. ExtendFS_UNIT_TRK to indicate that the amount to extend is in tracks.
3. ExtendFS_UNIT_CYL to indicate that the amount to extend is in cylinders.

- Set PCTL_EXT_AMT to the amount to be extended.
- The caller must be a superuser.
- If the flag PCTL_EXT_NEW_VOL = off, an attempt to obtain additional space is only made on the last volume of the HFS. The HFS will not be extended to a new volume.
- If the flag PCTL_EXT_NEW_VOL = on, an attempt to obtain additional space is only made on a new volume. In this case, candidate volumes must exist for the HFS, and the candidate volumes must have existed at the time of the current mount of the HFS.
- If a secondary space amount exists for the HFS, it is ignored.
- If the entire amount requested cannot be obtained, the operation fails.

• This command is equivalent to the confighfs shell command:
  
  `confighfs -x size pathname` To extend on the same volume

  `confighfs -xn size pathname` To extend to a new volume

  where size is the amount to be extended suffixed by the extend unit of M (megabytes), T (tracks), or C (cylinders) and pathname is an absolute or relative pathname that identifies the HFS.

**BPX1PCT Return and Reason Codes**

Using BPX1PCT for HFS returns a non-zero return code and reason code only if the return value is -1. Table 91 contains a list of return and reason codes returned by BPX1PCT for the HFS DisplayBufferLimits, ChangeBufferLimits, DisplayGlobalStats, DisplayFSStats, and ExtendFS commands.

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Description</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Successful completion</td>
<td>ExtendFS only</td>
</tr>
<tr>
<td>EINVAL</td>
<td>5B360106</td>
<td>The HFS pfsctl service ExtendFS function was called with an invalid PCTL_EXT_UNIT parameter.</td>
<td>ExtendFS only</td>
</tr>
<tr>
<td>EFAULT</td>
<td>5B360102</td>
<td>The HFS pfsctl service was called with an unsupported command code.</td>
<td>ExtendFS only</td>
</tr>
<tr>
<td></td>
<td>5B360103</td>
<td>The data area address provided for the HFS pfsctl command is zero.</td>
<td>ExtendFS only</td>
</tr>
<tr>
<td></td>
<td>5B360107</td>
<td>The HFS pfsctl ChangeBufferLimits or ExtendFS function was called by an unauthorized caller.</td>
<td>ExtendFS only</td>
</tr>
</tbody>
</table>
Table 91. BPX1PCT - Return Codes and Reason Codes (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Description</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMVSERR</td>
<td>5B360101</td>
<td>An error occurred during the add local recovery routine function.</td>
<td></td>
</tr>
<tr>
<td>ENOBUSFS</td>
<td>5B360104</td>
<td>The data area length provided for the HFS psctl command was less than the minimum size required to complete the request.</td>
<td></td>
</tr>
<tr>
<td>ENOENT</td>
<td>5B360105</td>
<td>An HFS psctl service was called with an unmounted HFS.unted DisplayFSStats and ExtendFS only.</td>
<td></td>
</tr>
<tr>
<td>ENOSPC</td>
<td>5B27C005</td>
<td>DADSM error occurred; no space available to extend the HFS data set</td>
<td>ExtendFS only</td>
</tr>
<tr>
<td></td>
<td>&gt;80000000</td>
<td>File system was in an HFS Out of Space error state and the extend amount was insufficient to meet the requirement to complete the sync shadow write. The reason code contains the 2's complement of the additional number of tracks required.</td>
<td>ExtendFS only</td>
</tr>
<tr>
<td>EIO</td>
<td>5B27C00A</td>
<td>DADSM error occurred; I/O error VTOC</td>
<td>ExtendFS only</td>
</tr>
<tr>
<td>EMVSPFSERM</td>
<td>5B27xxxx</td>
<td>Unexpected error encountered</td>
<td>ExtendFS only</td>
</tr>
<tr>
<td></td>
<td>5BB3xxxx</td>
<td>Unexpected error encountered</td>
<td>ChangeBufferLimits only</td>
</tr>
<tr>
<td>EMVSPARM</td>
<td>5B36000E</td>
<td>VMAX below minimum (VMAX set to minimum)</td>
<td>ChangeBufferLimits only</td>
</tr>
<tr>
<td></td>
<td>5B36000F</td>
<td>FMIN exceeded VMAX (FMIN set to VMAX)</td>
<td>ChangeBufferLimits only</td>
</tr>
<tr>
<td></td>
<td>5B360010</td>
<td>FMIN at maximum available (FMIN set to maximum)</td>
<td>ChangeBufferLimits only</td>
</tr>
<tr>
<td></td>
<td>5B360014</td>
<td>VMAX below FMIN (VMAX set to FMIN)</td>
<td>ChangeBufferLimits only</td>
</tr>
</tbody>
</table>

Table 92 shows the GFUMPCTL structure for the DisplayBufferLimits and the ChangeBufferLimits commands:

Table 92. Structure for the DisplayBufferLimits and ChangeBufferLimits Commands (GFUMPCTL)

<table>
<thead>
<tr>
<th>Offsets</th>
<th>Type</th>
<th>Length</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (0)</td>
<td>STRUCTURE</td>
<td>28</td>
<td>PCTL_BFRLIMITS_TYPE</td>
<td></td>
</tr>
<tr>
<td>0 (0)</td>
<td>CHARACTER</td>
<td>1</td>
<td>PCTL_BL_FLGS</td>
<td>Flags</td>
</tr>
</tbody>
</table>
### Table 92. Structure for the DisplayBufferLimits and ChangeBufferLimits Commands (GFUMPCTL) (continued)

<table>
<thead>
<tr>
<th>Offsets</th>
<th>Type</th>
<th>Length</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (1)</td>
<td>CHARACTER</td>
<td>1</td>
<td>PCTL_BL_ACTION</td>
<td>on: display buffer limits</td>
</tr>
<tr>
<td>1 (1)</td>
<td>CHARACTER</td>
<td>1</td>
<td>PCTL_BL_LIMIT</td>
<td>Limits to be modified</td>
</tr>
<tr>
<td>1 (1)</td>
<td>CHARACTER</td>
<td>1</td>
<td>PCTL_BL_VMAX</td>
<td>Modify VMAX request</td>
</tr>
<tr>
<td>1 (1)</td>
<td>CHARACTER</td>
<td>1</td>
<td>PCTL_BL_FMIN</td>
<td>Modify FMIN request</td>
</tr>
<tr>
<td>2 (2)</td>
<td>CHARACTER</td>
<td>1</td>
<td>*</td>
<td>Reserved</td>
</tr>
<tr>
<td>4 (4)</td>
<td>SIGNED</td>
<td>4</td>
<td>PCTL_BL_VMAX_VAL</td>
<td>Modify VMAX value</td>
</tr>
<tr>
<td>8 (8)</td>
<td>SIGNED</td>
<td>4</td>
<td>PCTL_BL_FMIN_VAL</td>
<td>Modify FMIN value</td>
</tr>
<tr>
<td>12 (1C)</td>
<td>CHARACTER</td>
<td>16</td>
<td>*</td>
<td>Reserved</td>
</tr>
<tr>
<td>28 (1C)</td>
<td>CHARACTER</td>
<td></td>
<td>PCTL_BL_END</td>
<td></td>
</tr>
</tbody>
</table>

Table 93 shows the GFUMPCTL structure for the DisplayGlobalStats command:

### Table 93. Structure for the DisplayGlobalStats Command (GFUMPCTL)

<table>
<thead>
<tr>
<th>Offsets</th>
<th>Type</th>
<th>Length</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (0)</td>
<td>STRUCTURE</td>
<td>200</td>
<td>PCTL_GLOBALSTATS_TYPE</td>
<td></td>
</tr>
<tr>
<td>0 (0)</td>
<td>CHARACTER</td>
<td>8</td>
<td>PCTL_GS_CURR_TIME</td>
<td>Current timestamp</td>
</tr>
<tr>
<td>8 (8)</td>
<td>CHARACTER</td>
<td>8</td>
<td>*</td>
<td>Reserved</td>
</tr>
<tr>
<td>16 (10)</td>
<td>SIGNED</td>
<td>4</td>
<td>PCTL_GS_TOTVIRT</td>
<td>Combined total virtual (in pages) in use by all buffer pools</td>
</tr>
<tr>
<td>20 (14)</td>
<td>SIGNED</td>
<td>4</td>
<td>PCTL_GS_TOTFIX</td>
<td>Combined total fixed (in pages) in use by all buffer pools</td>
</tr>
<tr>
<td>24 (18)</td>
<td>CHARACTER</td>
<td>28</td>
<td>PCTL_GS_BP1</td>
<td>Stats for buffer pool 1</td>
</tr>
<tr>
<td>24 (18)</td>
<td>UNSIGNED</td>
<td>2</td>
<td>PCTL_GS_BP1_BFRSIZE</td>
<td>Size of each buffer (in pages)</td>
</tr>
<tr>
<td>26 (1A)</td>
<td>UNSIGNED</td>
<td>2</td>
<td>PCTL_GS_BP1_DSCNT</td>
<td>Number of data spaces in pool</td>
</tr>
<tr>
<td>28 (1C)</td>
<td>SIGNED</td>
<td>4</td>
<td>PCTL_GS_BP1_TOTVIRT</td>
<td>Total virtual used by this pool</td>
</tr>
<tr>
<td>32 (20)</td>
<td>SIGNED</td>
<td>4</td>
<td>PCTL_GS_BP1_TOTFIX</td>
<td>Total fixed used by this pool</td>
</tr>
<tr>
<td>36 (24)</td>
<td>UNSIGNED</td>
<td>8</td>
<td>PCTL_GS_BP1_FIXD_Y</td>
<td>Times a buffer was already fixed prior to an I/O request</td>
</tr>
<tr>
<td>44 (2C)</td>
<td>UNSIGNED</td>
<td>8</td>
<td>PCTL_GS_BP1_FIXD_N</td>
<td>Times a buffer was not already fixed prior to an I/O request</td>
</tr>
<tr>
<td>52 (34)</td>
<td>CHARACTER</td>
<td>28</td>
<td>PCTL_GS_BP2</td>
<td>Stats for buffer pool 2</td>
</tr>
<tr>
<td>52 (34)</td>
<td>UNSIGNED</td>
<td>2</td>
<td>PCTL_GS_BP2_BFRSIZE</td>
<td>Size of each buffer (in pages)</td>
</tr>
<tr>
<td>54 (36)</td>
<td>UNSIGNED</td>
<td>2</td>
<td>PCTL_GS_BP2_DSCNT</td>
<td>Number of data spaces in pool</td>
</tr>
<tr>
<td>56 (38)</td>
<td>SIGNED</td>
<td>4</td>
<td>PCTL_GS_BP2_TOTVIRT</td>
<td>Total virtual used by this pool</td>
</tr>
<tr>
<td>60 (3C)</td>
<td>SIGNED</td>
<td>4</td>
<td>PCTL_GS_BP2_TOTFIX</td>
<td>Total fixed used by this pool</td>
</tr>
<tr>
<td>64 (40)</td>
<td>UNSIGNED</td>
<td>8</td>
<td>PCTL_GS_BP2_FIXD_Y</td>
<td>Times a buffer was already fixed prior to an I/O request</td>
</tr>
<tr>
<td>72 (48)</td>
<td>UNSIGNED</td>
<td>8</td>
<td>PCTL_GS_BP2_FIXD_N</td>
<td>Times a buffer was not already fixed prior to an I/O request</td>
</tr>
<tr>
<td>80 (50)</td>
<td>CHARACTER</td>
<td>28</td>
<td>PCTL_GS_BP3</td>
<td>Stats for buffer pool 3</td>
</tr>
<tr>
<td>80 (50)</td>
<td>UNSIGNED</td>
<td>2</td>
<td>PCTL_GS_BP3_BFRSIZE</td>
<td>Size of each buffer (in pages)</td>
</tr>
<tr>
<td>82 (52)</td>
<td>UNSIGNED</td>
<td>2</td>
<td>PCTL_GS_BP3_DSCNT</td>
<td>Number of data spaces in pool</td>
</tr>
<tr>
<td>84 (54)</td>
<td>SIGNED</td>
<td>4</td>
<td>PCTL_GS_BP3_TOTVIRT</td>
<td>Total virtual used by this pool</td>
</tr>
<tr>
<td>88 (58)</td>
<td>SIGNED</td>
<td>4</td>
<td>PCTL_GS_BP3_TOTFIX</td>
<td>Total fixed used by this pool</td>
</tr>
<tr>
<td>92 (5C)</td>
<td>UNSIGNED</td>
<td>8</td>
<td>PCTL_GS_BP3_FIXD_Y</td>
<td>Times a buffer was already fixed prior to an I/O request</td>
</tr>
<tr>
<td>100 (64)</td>
<td>UNSIGNED</td>
<td>8</td>
<td>PCTL_GS_BP3_FIXD_N</td>
<td>Times a buffer was not already fixed prior to an I/O request</td>
</tr>
<tr>
<td>108 (6C)</td>
<td>CHARACTER</td>
<td>28</td>
<td>PCTL_GS_BP4</td>
<td>Stats for buffer pool 4</td>
</tr>
<tr>
<td>108 (6C)</td>
<td>UNSIGNED</td>
<td>2</td>
<td>PCTL_GS_BP4_BFRSIZE</td>
<td>Size of each buffer (in pages)</td>
</tr>
<tr>
<td>110 (6E)</td>
<td>UNSIGNED</td>
<td>2</td>
<td>PCTL_GS_BP4_DSCNT</td>
<td>Number of data spaces in pool</td>
</tr>
<tr>
<td>112 (70)</td>
<td>SIGNED</td>
<td>4</td>
<td>PCTL_GS_BP4_TOTVIRT</td>
<td>Total virtual used by this pool</td>
</tr>
</tbody>
</table>
### Planning for HFS

#### Table 93. Structure for the DisplayGlobalStats Command (GFUMPCTL) (continued)

<table>
<thead>
<tr>
<th>Offsets</th>
<th>Type</th>
<th>Length</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>116 (74)</td>
<td>SIGNED</td>
<td>4</td>
<td>PCTL_GS_BP4_TOTFIX</td>
<td>Total fixed used by this pool</td>
</tr>
<tr>
<td>120 (78)</td>
<td>CHARACTER</td>
<td>8</td>
<td>PCTL_GS_BP4_FIXD_Y</td>
<td>Times a buffer was already fixed prior to an I/O request</td>
</tr>
<tr>
<td>128 (80)</td>
<td>CHARACTER</td>
<td>8</td>
<td>PCTL_GS_BP4_FIXD_N</td>
<td>Times a buffer was not already fixed prior to an I/O request</td>
</tr>
<tr>
<td>136 (88)</td>
<td>CHARACTER</td>
<td>8</td>
<td>PCTL_GS_META_Y</td>
<td>Times metadata in cache on a lookup</td>
</tr>
<tr>
<td>144 (90)</td>
<td>CHARACTER</td>
<td>8</td>
<td>PCTL_GS_META_N</td>
<td>Times metadata not in cache on a lookup</td>
</tr>
<tr>
<td>152 (98)</td>
<td>CHARACTER</td>
<td>8</td>
<td>PCTL_GS_RPN0_Y</td>
<td>Times RPN0 in cache on read/write</td>
</tr>
<tr>
<td>160 (A0)</td>
<td>CHARACTER</td>
<td>8</td>
<td>PCTL_GS_RPN0_N</td>
<td>Times RPN0 not in cache on read/write</td>
</tr>
<tr>
<td>168 (A8)</td>
<td>CHARACTER</td>
<td>32</td>
<td>*</td>
<td>Reserved</td>
</tr>
<tr>
<td>200 (C8)</td>
<td>CHARACTER</td>
<td></td>
<td>PCTL_GS_END</td>
<td></td>
</tr>
</tbody>
</table>

#### Table 94 contains the GFUMPCTL structure DisplayFSStats command:

#### Table 94. Structure for the DisplayFSStats Command (GFUMPCTL)

<table>
<thead>
<tr>
<th>Offsets</th>
<th>Type</th>
<th>Length</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (0)</td>
<td>STRUCTURE</td>
<td>228</td>
<td>PCTL_FSSTATS_TYPE</td>
<td>File system name</td>
</tr>
<tr>
<td>0 (0)</td>
<td>CHARACTER</td>
<td>44</td>
<td>PCTL_FS_NAME</td>
<td>Flags from RFS (an internal HFS control block)</td>
</tr>
<tr>
<td>44 (2C)</td>
<td>BITSTRING</td>
<td>4</td>
<td>PCTL_FS_FLAGS</td>
<td>MTAB flags</td>
</tr>
<tr>
<td>44 (2C)</td>
<td>BITSTRING</td>
<td>1</td>
<td>PCTL_FS_RFS_MTAB_FLAGS</td>
<td>RFS flags</td>
</tr>
<tr>
<td>45 (2D)</td>
<td>BITSTRING</td>
<td>1</td>
<td>PCTL_FS_RFS_FLAGS</td>
<td>RFS error flags</td>
</tr>
<tr>
<td>46 (2E)</td>
<td>BITSTRING</td>
<td>1</td>
<td>PCTL_FS_RFS_SYNC_ERR</td>
<td>Reserved</td>
</tr>
<tr>
<td>47 (2F)</td>
<td>BITSTRING</td>
<td>1</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>48 (30)</td>
<td>CHARACTER</td>
<td>8</td>
<td>PCTL_FS_CURR_TIME</td>
<td>Current timestamp</td>
</tr>
<tr>
<td>56 (38)</td>
<td>CHARACTER</td>
<td>8</td>
<td>PCTL_FS_MOUNT_TIME</td>
<td>Mount timestamp</td>
</tr>
<tr>
<td>64 (40)</td>
<td>UNSIGNED</td>
<td>2</td>
<td>PCTL_FS_SYNC</td>
<td>SYNC interval (in seconds)</td>
</tr>
<tr>
<td>66 (42)</td>
<td>CHARACTER</td>
<td>2</td>
<td>*</td>
<td>Reserved</td>
</tr>
<tr>
<td>68 (44)</td>
<td>SIGNED</td>
<td>4</td>
<td>PCTL_FS_SIZE</td>
<td>File system size (in pages)</td>
</tr>
<tr>
<td>72 (48)</td>
<td>SIGNED</td>
<td>4</td>
<td>PCTL_FS_USED</td>
<td>Number of pages used in FS</td>
</tr>
<tr>
<td>76 (4C)</td>
<td>SIGNED</td>
<td>4</td>
<td>PCTL_FS_AD_ALLOC</td>
<td>Number of pages allocated to AD in FS</td>
</tr>
<tr>
<td>80 (50)</td>
<td>UNSIGNED</td>
<td>8</td>
<td>PCTL_FS_SEQ_IO</td>
<td>Number of sequential I/O requests for data</td>
</tr>
<tr>
<td>88 (58)</td>
<td>UNSIGNED</td>
<td>8</td>
<td>PCTL_FS_RANDOM_IO</td>
<td>Number of random I/O requests for data</td>
</tr>
<tr>
<td>96 (60)</td>
<td>UNSIGNED</td>
<td>8</td>
<td>PCTL_FS_META_Y</td>
<td>Times metadata in cache on a lookup</td>
</tr>
<tr>
<td>104 (68)</td>
<td>UNSIGNED</td>
<td>8</td>
<td>PCTL_FS_META_N</td>
<td>Times metadata not in cache on a lookup</td>
</tr>
<tr>
<td>112 (70)</td>
<td>UNSIGNED</td>
<td>8</td>
<td>PCTL_FS_RPN0_Y</td>
<td>Times RPN0 in cache on read/write</td>
</tr>
<tr>
<td>120 (78)</td>
<td>UNSIGNED</td>
<td>8</td>
<td>PCTL_FS_RPN0_N</td>
<td>Times RPN0 NOT in cache on read/write</td>
</tr>
<tr>
<td>128 (80)</td>
<td>UNSIGNED</td>
<td>8</td>
<td>PCTL_FS_IX_TOPS</td>
<td>Number of index new tops</td>
</tr>
<tr>
<td>136 (88)</td>
<td>UNSIGNED</td>
<td>8</td>
<td>PCTL_FS_IX_SPLITS</td>
<td>Number of index splits</td>
</tr>
<tr>
<td>144 (90)</td>
<td>UNSIGNED</td>
<td>8</td>
<td>PCTL_FS_IX_JOINS</td>
<td>Number of index joins</td>
</tr>
<tr>
<td>152 (98)</td>
<td>UNSIGNED</td>
<td>8</td>
<td>PCTL_FS_IX_RDHIT</td>
<td>Number of index page read hits</td>
</tr>
<tr>
<td>160 (A0)</td>
<td>UNSIGNED</td>
<td>8</td>
<td>PCTL_FS_IX_RDMISS</td>
<td>Number of index page read misses</td>
</tr>
<tr>
<td>168 (A8)</td>
<td>UNSIGNED</td>
<td>8</td>
<td>PCTL_FS_IX_WRHIT</td>
<td>Number of index page write hits</td>
</tr>
<tr>
<td>176 (B0)</td>
<td>UNSIGNED</td>
<td>8</td>
<td>PCTL_FS_IX_WRMISS</td>
<td>Number of index page write misses</td>
</tr>
<tr>
<td>184 (B8)</td>
<td>UNSIGNED</td>
<td>4</td>
<td>PCTL_FS_PGS_CACHED</td>
<td>Number of data buffer pages cached by this file system</td>
</tr>
<tr>
<td>188 (BC)</td>
<td>CHARACTER</td>
<td>4</td>
<td>PCTL_FS_HFRFN</td>
<td>HFRFN from DMIB</td>
</tr>
</tbody>
</table>
### Table 94. Structure for the DisplayFSStats Command (GFUMPCTL) (continued)

<table>
<thead>
<tr>
<th>Offsets</th>
<th>Type</th>
<th>Length</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>192 (C0)</td>
<td>UNSIGNED</td>
<td>4</td>
<td>PCTL_FS_MEM_CNT</td>
<td>Member count from RFS</td>
</tr>
<tr>
<td>196 (C4)</td>
<td>UNSIGNED</td>
<td>4</td>
<td>PCTL_FS_APM_CNT</td>
<td>Number of available page maps</td>
</tr>
<tr>
<td>200 (C8)</td>
<td>CHARACTER</td>
<td>28</td>
<td>*</td>
<td>Reserved</td>
</tr>
<tr>
<td>228 (E4)</td>
<td>CHARACTER</td>
<td>*</td>
<td>PCTL_FS_END</td>
<td></td>
</tr>
</tbody>
</table>

### Table 95. Structure for the ExtendFS Command (GFUMPCTL)

<table>
<thead>
<tr>
<th>Offsets</th>
<th>Type</th>
<th>Length</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (0)</td>
<td>STRUCTURE</td>
<td>68</td>
<td>PCTL_EXTENDFS_TYPE</td>
<td></td>
</tr>
<tr>
<td>0 (0)</td>
<td>CHARACTER</td>
<td>44</td>
<td>PCTL_EXT_FSNAME</td>
<td>File system name</td>
</tr>
<tr>
<td>44 (2C)</td>
<td>CHARACTER</td>
<td>1</td>
<td>PCTL_EXT_FLGS</td>
<td>Flags</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PCTL_EXT_NEW_VOL</td>
<td>Extend to new volume</td>
</tr>
<tr>
<td>45 (2D)</td>
<td>CHARACTER</td>
<td>1</td>
<td>PCTL_EXT_UNIT</td>
<td>Unit of amount to extend ('M', 'T' or 'C')</td>
</tr>
<tr>
<td>46 (2E)</td>
<td>CHARACTER</td>
<td>2</td>
<td>*</td>
<td>Reserved</td>
</tr>
<tr>
<td>48 (30)</td>
<td>SIGNED</td>
<td>4</td>
<td>PCTL_EXT_AMT</td>
<td>Amount to be extended</td>
</tr>
<tr>
<td>52 (34)</td>
<td>CHARACTER</td>
<td>16</td>
<td>*</td>
<td>Reserved</td>
</tr>
<tr>
<td>68 (44)</td>
<td>CHARACTER</td>
<td>*</td>
<td>PCTL_EXT_END</td>
<td></td>
</tr>
</tbody>
</table>

### Table 96. Constants for Extend Units Supported

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHARACTER</td>
<td>1</td>
<td>EXTENDFS_UNIT_MB</td>
<td>Constant ('M', Megabytes (MB) for pctl_ext_unit</td>
</tr>
<tr>
<td>CHARACTER</td>
<td>1</td>
<td>EXTENDFS_UNIT_TRK</td>
<td>Constant ('T', Tracks) for pctl_ext_unit</td>
</tr>
<tr>
<td>CHARACTER</td>
<td>1</td>
<td>EXTENDFS_UNIT_CYL</td>
<td>Constant ('C', Cylinders) for pctl_ext_unit</td>
</tr>
</tbody>
</table>
Planning for HFS
Chapter 12. User Access to Subsystem Statistics, Status, and Counts Information

This information documents programming interfaces provided by IDCAMS.

Your program can call a system service that returns the information that the access method services LISTDATA command returns. This information is subsystem statistics, status and count information. To obtain this information, your program must be APF-authorized. Your program will call IDCSS01 with the LINK macro. You pass to IDCSS01 a three-word parameter list pointed to by register 1. If your program is not authorized and attempts to link to IDCSS01, the system issues ABEND with system code 047. The following shows an example of the IDCSS01 call:

```
ST R5,AddrSGARGL Set address of SSGARGL
LINK EP=IDCSS01,PARAM=(,MyArgL,MyRetCode) Call LISTDATA
SR R15,R15 Prepare for ICM
ICM R15,'8'0011',MyRetCode Obtain and test return code
BNZ ListDataFail Branch if call failed
...
MyArgL DC A(AddrSSGARGL) Address of SSGARGL address
AddrSSGARGL DC A(0) Address of SSGARGL
MyRetcode DC H'0' Return code from service
IDCDEF70 , Map the parameter list
```

Your program must point register 13 to a standard 18-word save area when calling IDCSS01.

Register 1 Parameter List

Word 1
must be zero.

Word 2
contains an address of a pointer to the argument list SSGARGL (detailed below), which IDCSS01 requires. Within this argument list is a field named SSGOADR, which points to the buffer area in which IDCSS01 returns subsystem statistics, status, or counts information. The buffer area may be obtained by the caller or may be left for IDCSS01 to obtain.

Word 3
points to an area to receive a 2-byte binary return code. Possible return codes are listed at the end of SSGARGL. IDCSS01 also returns this return code in the low order bytes of register 15.

Passed Argument List -- SSGARGL

The following describes SSGARGL. SSGARGL is the area pointed to by the word that Word 2 points to. Word 2 is part of the parameter list passed to IDCSS01. The caller must establish some fields; IDCSS01 establishes other fields. The caller must set an option flag to indicate whether information requested is status information (SSGRSS), counts information (SSGRPD), Space Efficient Volume status (SSGSEV), or Extent Pool Configuration status (SSGEP). If the caller passes SSGADDN, IDCSS01 establishes SSGAVOL and SSGUNIT.
If counts information is requested, the caller must indicate whether it applies to all subsystems (SSGALL), a specific subsystem (SSG1SS), or a specific device (SSGDEV). In addition, the caller must either pass the ddname (through SSGADDN) of a DD statement that allocates a caching subsystem volume, or the caller must identify the volume and unit (through SSGAVOL and SSGUNIT) of a caching subsystem volume for which information is being requested.

If Space Efficient Volume status, or Extent Pool Configuration status is requested, the caller must follow the instruction described in the mapping for the space efficient volume status output buffer, or mapping for the extent pool configuration status output buffer described in later of this section.

**Note:** The SSGOLN may be larger than the data returned in the SSGBUFR buffer. To detect end of data, the following should be considered:
- If “current index into the SSGBUFR buffer” is within SSGOLN-4 bytes, then the full six bytes of SSGDAVOL being zero is the way to determine the premature “end of data”.
- If “current index into the SSGBUFR buffer” is beyond SSGOLN-4 bytes of the data, then they are at “end of data”.

The following list contains other fields that IDCSS01 establishes:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Type</th>
<th>Length</th>
<th>Name (Dim)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>STRUCTURE</td>
<td>96</td>
<td>SSGARGL</td>
<td>SSGARGL IDENTIFIER</td>
</tr>
<tr>
<td>0</td>
<td>CHARACTER</td>
<td>8</td>
<td>SSGHEAD</td>
<td>SSGARGL IDENTIFIER</td>
</tr>
<tr>
<td>8</td>
<td>ADDRESS</td>
<td>4</td>
<td>SSGADDN</td>
<td>DDNAME ADDRESS</td>
</tr>
<tr>
<td>12</td>
<td>ADDRESS</td>
<td>4</td>
<td>SSGAVOL</td>
<td>VOLUME ADDRESS</td>
</tr>
<tr>
<td>16</td>
<td>BITSTRING</td>
<td>4</td>
<td>SSGUNIT</td>
<td>DEVICE TYPE</td>
</tr>
<tr>
<td>16</td>
<td>BITSTRING</td>
<td>1</td>
<td>SSGUNIT1</td>
<td>GENERAL FLAGS</td>
</tr>
<tr>
<td>17</td>
<td>BITSTRING</td>
<td>1</td>
<td>SSGUNIT2</td>
<td>GENERAL FLAGS</td>
</tr>
<tr>
<td>18</td>
<td>BITSTRING</td>
<td>1</td>
<td>SSGUNIT3</td>
<td>DEVICE CLASS</td>
</tr>
<tr>
<td>19</td>
<td>BITSTRING</td>
<td>1</td>
<td>SSGUNIT4</td>
<td>DEVICE TYPE WITHIN THE CLASS</td>
</tr>
<tr>
<td>20</td>
<td>SIGNED</td>
<td>4</td>
<td>SSGOLN</td>
<td>LENGTH OF OUTPUT BUFFER</td>
</tr>
<tr>
<td>24</td>
<td>ADDRESS</td>
<td>4</td>
<td>SSGADDR</td>
<td>ADDRESS OF OUTPUT BUFFER</td>
</tr>
<tr>
<td>28</td>
<td>BITSTRING</td>
<td>2</td>
<td>SSGOPT</td>
<td>OPTIONS BYTE</td>
</tr>
<tr>
<td>1</td>
<td>.1...</td>
<td>.1...</td>
<td>SSGRPD</td>
<td>SENSE SUBSYSTEM COUNTS</td>
</tr>
<tr>
<td>1</td>
<td>.1...</td>
<td>.1...</td>
<td>SSGRSS</td>
<td>SENSE SUBSYSTEM STATUS</td>
</tr>
<tr>
<td>1</td>
<td>.1...</td>
<td>.1...</td>
<td>SSGCACHE</td>
<td>ON=CACHING,OFF=PAGING</td>
</tr>
<tr>
<td>1</td>
<td>.1...</td>
<td>.1...</td>
<td>SSGSDS</td>
<td>ON=1 SD, OFF=2 SD’S</td>
</tr>
<tr>
<td>1</td>
<td>.1...</td>
<td>.1...</td>
<td>SSGALL</td>
<td>SENSE FOR ALL SUBSYSTEMS</td>
</tr>
<tr>
<td>1</td>
<td>.1...</td>
<td>.1...</td>
<td>SSGISS</td>
<td>SENSE FOR SPECIFIED SUBSYSTEM</td>
</tr>
<tr>
<td>1</td>
<td>.1...</td>
<td>.1...</td>
<td>SSGDEV</td>
<td>SENSE FOR SPECIFIED DEVICE</td>
</tr>
<tr>
<td>1</td>
<td>.1...</td>
<td>.1...</td>
<td>SSGAMD</td>
<td>PTR TO 3880 MODEL PASSED INSTEAD OF PTR TO VOLUME</td>
</tr>
<tr>
<td>29</td>
<td>1...</td>
<td>1...</td>
<td>SSG2SD</td>
<td>PRINT 2 SD</td>
</tr>
<tr>
<td>1</td>
<td>.1...</td>
<td>.1...</td>
<td>SSGACD</td>
<td>ACCESSCODE</td>
</tr>
<tr>
<td>1</td>
<td>.1...</td>
<td>.1...</td>
<td>SSGOFFL</td>
<td>SSGAVOL POINTS TO DEVICE ID OF AN OFFLINE DEVICE</td>
</tr>
<tr>
<td>1</td>
<td>.1...</td>
<td>.1...</td>
<td>SSGARGL2</td>
<td>this flag indicates that the parameter list is extended by the 16 bytes defined as SSGARGLX</td>
</tr>
</tbody>
</table>
| 1      | .1...             | .1...  | SSGLINKP   | Link performance statistics supported by caller and request return of statistics when request for Subsystem counts are made (SSGPRP = ON) for one or more Subsystems (SSGISS=ON or SSGALL = ON). Results passed back in area pointed to by SSSLMOAR Extended size statistics are provided if SSGEXRQ is specified. The returned field, SSSLSET, will be set to the size for each set of link statistics.
Parameter List Extension.

Note: IDCSS01 will not reference fields contained in SSGARGLY unless the bit defined as SSGARGL3 is set to '1'.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSGARGL3</td>
<td>BITSTRING</td>
<td>this flag indicates that the parameter list is extended with area defined as SSGARGLY</td>
</tr>
<tr>
<td>SSGMEMLID</td>
<td>CHARACTER</td>
<td>CU MODEL IDENTIFIER</td>
</tr>
<tr>
<td>SSGRCIOS</td>
<td>CHARACTER</td>
<td>'old' parameter list end</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSGATIME</td>
<td>UNSIGNED</td>
<td>I/O timeout value in seconds. When SSGATIME is not zero, it will be stored into IOSXTIME in the IOSB. If the I/O is active or queued longer that SSGATIME, the I/O will be terminated and IDCSS01 will return a return code of 60. Hex values for SSGATIME support seconds from 1 to 255.</td>
</tr>
<tr>
<td>SSGADEYN</td>
<td>CHARACTER</td>
<td>Binary device number that received an I/O (RC8) or timeout (RC60) from IDCSS01</td>
</tr>
<tr>
<td>SGLPRET</td>
<td>UNSIGNED</td>
<td>Reason code from link performance statistics processing</td>
</tr>
<tr>
<td>SSGLPOAR</td>
<td>ADDRESS</td>
<td>Address of link performance statistics output buffer queue. When zero on return statistics not available for devices found on counts request or statistics not not requested by the caller</td>
</tr>
</tbody>
</table>

Note: IDCSS01 will not reference fields contained in SSGARGLY unless the bit defined as SSGARGL3 is set to '1'. Entire section added

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSGARGLY</td>
<td>CHARACTER</td>
<td>Parm list extension 3</td>
</tr>
<tr>
<td>SSGRANKP</td>
<td>BITSTRING</td>
<td>RANK performance statistics are supported by caller and request return of statistics when request for Subsystem counts are made (SSGRPD = ON) for one or more Subsystems (SSG1SS=ON or SSGALL = ON). Results passed back in area pointed to by SSGROAR</td>
</tr>
<tr>
<td>SSGSEGMP</td>
<td>BITSTRING</td>
<td>Segment pool statistics are supported by caller and request return of statistics when request for Subsystem counts are made (SSGRPD = ON) for one or more Subsystems (SSG1SS=ON or SSGALL = ON). Results passed back in area pointed to by SSGPOAR</td>
</tr>
<tr>
<td>SSGSEV</td>
<td>BITSTRING</td>
<td>Space Efficient Volume (SEVOL) status is supported by caller and request return of status in SSGSEBUF pointed by SSGSEDR when request for SEVOL are made (SSGSEV =ON) for the scope of single device (SSGSEV=ON), online SEVOL devices in the subsystem (SSG1SS=ON), or all online SEVOL devices (SSGALL=ON). For single offline SEVOL status, caller set address of the MVS addr character in SSVAVOL and set</td>
</tr>
</tbody>
</table>
offline (SSGOFFL=ON). The Fixed Block (FB) SEVOL device status is returned when the reporting scope is single (SSGDEV=ON), the FB user request is active (SSGURFBD=ON), FB LSS number is in SSGFBLS, and FB device number is in SSGFBDEV. FB SEVOL status is available through CKD I/O device that resides in the same Storage Facility Image as FB SEVOL and resides in the same Even or Odd number LSS as even or odd FB LSS. SEVOL status is not supported when other statistics or status is requested by caller.

....1.... SSGEPC
Extent Pool Config (EPC) status is supported by caller and request return of status in SSGEPBUF pointed by SSGEOAR when request for EPC of Storage Facility Image oriented by I/O device in SSGADN or SSGAVOL are made (SSGEP=ON). When Extent Pool ID (SSGEPID) is set as 'FFFF'X by caller, EPC summary status is returned. When even numbered Extent Pool ID is specified in SSGEPID with the I/O device in the even LSS, detail status of EPC for the specified Extent Pool is returned. When odd numbered Extent Pool ID is specified with the I/O device in the odd LSS, detail status of EPC for the specified Extent Pool is returned. SSGURBMP is internal use only. When Extent Pool ID (SSGEPID) is set as 'FFFF'X with SSGURBMP is ON, branch entry caller receives error. Whether ON or OFF of SSGURBMP does not make difference in EPC detail status for branch entry caller. EPC status report is not supported when other statistics or status is requested by caller.

....1.... SSGURFBD
Request Fixed Block SEVOL status by turning SSGURFBD flag ON.

....1.... SSGURBMP
IDCAMS internal use. Refer SSGEPC bit description.

....1.... SSGEXRQ
By setting this attribute bit along with other request(s), caller supports and requests the return of extended length statistics. The extended length is any set of statistics that is larger than the standard length of statistics. The extended length statistics will be returned if target storage
facility supports it, otherwise the standard size statistics will be returned. Ignored this attribute bit for requests where the statistics do not support the extended length and where caller does not make any request.

Current request that supports the extended length is:
- Link performance statistics

<table>
<thead>
<tr>
<th>Offset</th>
<th>Type</th>
<th>Length</th>
<th>Name (Dim)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>52</td>
<td>CHARACTER</td>
<td>8</td>
<td>SSGRETY</td>
<td>Extended return codes</td>
</tr>
<tr>
<td>53</td>
<td>UNSIGNED</td>
<td>1</td>
<td>SSGRSRET</td>
<td>Reason code from Rank Performance Statistics</td>
</tr>
<tr>
<td>60</td>
<td>ADDRESS</td>
<td>4</td>
<td>SSGROAR</td>
<td>Address of Rank performance statistics output buffer queue. When zero on return statistics not available for devices found on counts request or statistics not not requested by the caller</td>
</tr>
<tr>
<td>64</td>
<td>ADDRESS</td>
<td>4</td>
<td>SSGPOAR</td>
<td>Address of segment pool performance Statistics output buffer queue. When zero on return statistics not available for devices found on counts request or statistics not not requested by the caller</td>
</tr>
<tr>
<td>68</td>
<td>ADDRESS</td>
<td>4</td>
<td>SSGEOAR</td>
<td>Address of SE Volume status output buffer queue. When zero on return status not available for devices found on SEVOL status request or status not not requested by the caller</td>
</tr>
<tr>
<td>72</td>
<td>ADDRESS</td>
<td>4</td>
<td>SSGPOAR</td>
<td>Address of Extent Pool Config (EPC) status output buffer. When zero on return status not available for Storage Facility Image found on EPC status request or status not not requested by the caller</td>
</tr>
<tr>
<td>76</td>
<td>UNSIGNED</td>
<td>2</td>
<td>SSGEPID</td>
<td>Ext Pool ID. ‘FFFF’X indicates no Ext Pool ID is provided by caller.</td>
</tr>
<tr>
<td>78</td>
<td>UNSIGNED</td>
<td>2</td>
<td>SSGFBLS5</td>
<td>Fix Block device number</td>
</tr>
<tr>
<td>80</td>
<td>UNSIGNED</td>
<td>2</td>
<td>SSGFBEOL</td>
<td>Fix Block LSS number</td>
</tr>
<tr>
<td>82</td>
<td>CHARACTER</td>
<td>14</td>
<td>*</td>
<td>reserved for future use</td>
</tr>
</tbody>
</table>
MAPPPING FOR THE SUBSYSTEM GET OUTPUT BUFFER
FOR LINK PERFORMANCE STATISTICS
Output buffer is a chain, anchored by SSGLPOAR, of link
performance statistics tables. Each table entry represents
data read from a single storage subsystem box. The
statistics from a single box may be described with more than
table entry. The chain is a single threaded queue. Each
chain element needs to be freed by the caller of this
interface. Output buffer is built for storage boxes
that support link performance statistics and are returned
only if requested by caller (SSGLINKP) on a request for
performance data (SSGRPD) for requests with scope of
of subsystem (SSGISS) or all subsystems (SSGALL).
The parameter list extension must also be passed as
indicated with SSGALGZ.

<table>
<thead>
<tr>
<th>OFFSET</th>
<th>DECIMAL</th>
<th>HEX</th>
<th>TYPE</th>
<th>LENGTH</th>
<th>NAME (DIM)</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>STRUCTURE</td>
<td>0</td>
<td>* SSGLBUFR</td>
<td>Queue element header</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>CHARACTER</td>
<td>56</td>
<td>SSGLHDR</td>
<td>Length of this queue element, header plus data</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>UNSIGNED</td>
<td>4</td>
<td>SSGLQDAL</td>
<td>Offset to the start of the statistics sets</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
<td>UNSIGNED</td>
<td>4</td>
<td>SSGLQDAO</td>
<td>Pointer to next queue element, when zero end of queue</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>8</td>
<td>ADDRESS</td>
<td>4</td>
<td>SSGLFWDP</td>
<td></td>
</tr>
</tbody>
</table>

Orientation to subsystem that stats pertain

<table>
<thead>
<tr>
<th>OFFSET</th>
<th>DECIMAL</th>
<th>HEX</th>
<th>TYPE</th>
<th>LENGTH</th>
<th>NAME (DIM)</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>12</td>
<td>12</td>
<td>CHARACTER</td>
<td>6</td>
<td>SSSLVOL</td>
<td>Volume serial of device that stats where read from</td>
</tr>
<tr>
<td>18</td>
<td>18</td>
<td>18</td>
<td>CHARACTER</td>
<td>2</td>
<td>SSSLDEVN</td>
<td>Device number of device that Stats where read from</td>
</tr>
<tr>
<td>20</td>
<td>20</td>
<td>20</td>
<td>CHARACTER</td>
<td>8</td>
<td>*</td>
<td>Reserved</td>
</tr>
<tr>
<td>28</td>
<td>28</td>
<td>28</td>
<td>CHARACTER</td>
<td>28</td>
<td>SSSLHDR</td>
<td>Link stats info</td>
</tr>
<tr>
<td>28</td>
<td>28</td>
<td>28</td>
<td>UNSIGNED</td>
<td>2</td>
<td>SSSLNSET</td>
<td>Num of link stats sets read</td>
</tr>
<tr>
<td>30</td>
<td>30</td>
<td>30</td>
<td>UNSIGNED</td>
<td>2</td>
<td>SSSLSET</td>
<td>Size of each set</td>
</tr>
<tr>
<td>32</td>
<td>32</td>
<td>32</td>
<td>CHARACTER</td>
<td>6</td>
<td>SGLCUT</td>
<td>Control unit type</td>
</tr>
<tr>
<td>38</td>
<td>38</td>
<td>38</td>
<td>CHARACTER</td>
<td>3</td>
<td>SGLCUM</td>
<td>Control unit model</td>
</tr>
<tr>
<td>41</td>
<td>41</td>
<td>41</td>
<td>CHARACTER</td>
<td>10</td>
<td>SGLSEQ</td>
<td>Control unit sequence number</td>
</tr>
<tr>
<td>51</td>
<td>51</td>
<td>51</td>
<td>CHARACTER</td>
<td>1</td>
<td>SGLVER</td>
<td>Version of link statistics. When value of version is X'00', it indicates that the link statistics are defined in the 96 byte format. When value of version is X'01', it indicates that the link statistics are defined in the 156 byte format. Not used</td>
</tr>
<tr>
<td>52</td>
<td>52</td>
<td>52</td>
<td>CHARACTER</td>
<td>4</td>
<td>*</td>
<td>Link performance stats read</td>
</tr>
<tr>
<td>56</td>
<td>56</td>
<td>56</td>
<td>CHARACTER</td>
<td>96</td>
<td>SSSLSTA(+)</td>
<td>Stats entry</td>
</tr>
</tbody>
</table>

SSGLDADA consists of link statistics sets with Interface ID containing one or more table information entries. Each entry can be 96 bytes defined in SSSLSTA or 156 bytes defined in SGLXSTA depending on version number in SGLVER.
<table>
<thead>
<tr>
<th>OFFSET</th>
<th>DECIMAL</th>
<th>HEX</th>
<th>TYPE</th>
<th>LENGTH</th>
<th>NAME (DIM)</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>56 (38)</td>
<td>CHARACTER</td>
<td>1</td>
<td>SSGLLTYP</td>
<td>Link type</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01 - ESCON</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>02 - Fibre Channel 1 Gb/s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>03 - Fibre Channel 2 Gb/s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>04 - Fibre Channel 4 Gb/s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>05 - Fibre Channel 8 Gb/s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>06 - Fibre Channel 16 Gb/s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>57 (39)</td>
<td>UNSIGNED</td>
<td>1</td>
<td>SSGLLYI</td>
<td>Unused</td>
<td></td>
<td></td>
</tr>
<tr>
<td>58 (3A)</td>
<td>UNSIGNED</td>
<td>1</td>
<td>SSGLTMI</td>
<td>Unused</td>
<td></td>
<td></td>
</tr>
<tr>
<td>60 (3C)</td>
<td>CHARACTER</td>
<td>2</td>
<td>SSGLIID</td>
<td>Interface id</td>
<td></td>
<td></td>
</tr>
<tr>
<td>64 (40)</td>
<td>CHARACTER</td>
<td>88</td>
<td>SSGLSAT</td>
<td>Link statistics</td>
<td></td>
<td></td>
</tr>
<tr>
<td>64 (40)</td>
<td>UNSIGNED</td>
<td>4</td>
<td>SSGLERDB</td>
<td>Eckd read bytes in byte increment</td>
<td></td>
<td></td>
</tr>
<tr>
<td>68 (44)</td>
<td>UNSIGNED</td>
<td>4</td>
<td>SSGLERDO</td>
<td>Eckd read operations. for escon ports, one count per chain which transfers data to the host. for ficon ports, one count per command which transfers data to the host</td>
<td></td>
<td></td>
</tr>
<tr>
<td>76 (44)</td>
<td>UNSIGNED</td>
<td>4</td>
<td>SSGLERDT</td>
<td>Eckd read accumulated time on channel. the active processing time for each command is accumulated based on increment value</td>
<td></td>
<td></td>
</tr>
<tr>
<td>84 (44)</td>
<td>UNSIGNED</td>
<td>4</td>
<td>SSGLERRT</td>
<td>Eckd write operations. for escon ports, one count per chain which transfers data to the host. for ficon ports, one count per command which transfers data to the host</td>
<td></td>
<td></td>
</tr>
<tr>
<td>88 (48)</td>
<td>UNSIGNED</td>
<td>4</td>
<td>SSGLPDRB</td>
<td>Pprc send bytes in byte increment</td>
<td></td>
<td></td>
</tr>
<tr>
<td>92 (4C)</td>
<td>UNSIGNED</td>
<td>4</td>
<td>SSGLPRDO</td>
<td>Pprc send operations. each pprc write command sent by the pprc primary</td>
<td></td>
<td></td>
</tr>
<tr>
<td>96 (50)</td>
<td>UNSIGNED</td>
<td>4</td>
<td>SSGLPDRD</td>
<td>Pprc send operations. each pprc write command sent by the pprc secondary</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100 (54)</td>
<td>UNSIGNED</td>
<td>4</td>
<td>SSGLPWRB</td>
<td>Pprc received bytes in byte increment</td>
<td></td>
<td></td>
</tr>
<tr>
<td>104 (58)</td>
<td>UNSIGNED</td>
<td>4</td>
<td>SSGLPWRT</td>
<td>Pprc received accumulated time based on increment value</td>
<td></td>
<td></td>
</tr>
<tr>
<td>108 (62)</td>
<td>UNSIGNED</td>
<td>4</td>
<td>SSGLPWRD</td>
<td>Pprc received accumulated time based on increment value</td>
<td></td>
<td></td>
</tr>
<tr>
<td>112 (66)</td>
<td>UNSIGNED</td>
<td>4</td>
<td>SSGLSROB</td>
<td>Scsi read bytes in byte increment</td>
<td></td>
<td></td>
</tr>
<tr>
<td>116 (70)</td>
<td>UNSIGNED</td>
<td>4</td>
<td>SSGLSRDO</td>
<td>Scsi read operations. each scsi read is counted</td>
<td></td>
<td></td>
</tr>
<tr>
<td>120 (74)</td>
<td>UNSIGNED</td>
<td>4</td>
<td>SSGLSROD</td>
<td>Scsi read operations. each scsi write is counted</td>
<td></td>
<td></td>
</tr>
<tr>
<td>124 (78)</td>
<td>UNSIGNED</td>
<td>4</td>
<td>SSGLSRDT</td>
<td>Scsi read accumulated time based on increment value</td>
<td></td>
<td></td>
</tr>
<tr>
<td>128 (82)</td>
<td>UNSIGNED</td>
<td>4</td>
<td>SSGLSWRB</td>
<td>Scsi write bytes in byte increment</td>
<td></td>
<td></td>
</tr>
<tr>
<td>132 (86)</td>
<td>UNSIGNED</td>
<td>4</td>
<td>SSGLSWRT</td>
<td>Scsi write accumulated time based on increment value</td>
<td></td>
<td></td>
</tr>
<tr>
<td>136 (90)</td>
<td>CHARACTER</td>
<td>16</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

SSGLDADA consists of table entries defined by SSGLXSTA shown below when SSGLVER is decimal value of 1.
errors, such as unexpected NOS received or a link state machine failure detected.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Type</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>156</td>
<td>9C</td>
<td>4</td>
<td>Fibre Channel Loss of Synchronization Error Count. This count is the number of loss of synchronization errors where it is a confirmed and persistent synchronization loss on the fibre channel link.</td>
</tr>
<tr>
<td>160</td>
<td>A0</td>
<td>4</td>
<td>Fibre Channel Loss of Signal Error Count. This count is the number of times that a loss of signal was detected on the fibre channel link when a signal was previously detected.</td>
</tr>
<tr>
<td>164</td>
<td>A4</td>
<td>4</td>
<td>Fibre Channel Primitive Sequence Error Count. This count is the number of primitive sequence protocol error counts where an unexpected primitive sequence was received.</td>
</tr>
<tr>
<td>168</td>
<td>A8</td>
<td>4</td>
<td>Fibre Channel Invalid Transmission Word Error Count. This count is the number of times a &quot;bit&quot; error was detected. Examples of a &quot;bit&quot; errors are a code violation, invalid special code alignment, or disparity errors.</td>
</tr>
<tr>
<td>172</td>
<td>AC</td>
<td>4</td>
<td>Fibre Channel CRC Error Count. This count is the number of times a received frame's CRC is in error.</td>
</tr>
<tr>
<td>176</td>
<td>B0</td>
<td>4</td>
<td>Fibre Channel Link Recovery (LR) Sent Count Count. This count is the number of times the port has transitioned from an active (AC) state to a Link Recovery (LR1) state.</td>
</tr>
<tr>
<td>180</td>
<td>B4</td>
<td>4</td>
<td>Fibre Channel Link Recovery (LR) Received Count Recovery Count. This count is the number of times the port has transitioned from an active (AC) state to a Link Recovery (LR2) state.</td>
</tr>
<tr>
<td>184</td>
<td>B8</td>
<td>4</td>
<td>Fibre Channel Illegal Frame Count. This count is the number of frames that violated Fibre Channel protocol. One example is an invalid frame header. One common reason is when the first frame of data sequence is missing and a subsequent data frame is detected as illegal.</td>
</tr>
<tr>
<td>188</td>
<td>BC</td>
<td>4</td>
<td>Fibre Channel Out of Order Data Count. This count is the number of times that an out of order frame is detected. The frame is either missing from a data sequence or it is received beyond the port's sequence reassembly threshold.</td>
</tr>
<tr>
<td>192</td>
<td>C0</td>
<td>4</td>
<td>Fibre Channel Out of Order ACK Count. This count is the number of times that an out of order ACK frame is detected. The frame is either missing from a data sequence or it is received beyond the port's sequence reassembly threshold.</td>
</tr>
<tr>
<td>196</td>
<td>C4</td>
<td>4</td>
<td>Fibre Channel Duplicate Frame Count. This count is the number of times a frame was received that has been detected as...</td>
</tr>
</tbody>
</table>
MAP\ing FOR THE SUBSYSTEM GET OUTPUT BUFFER  
FOR RANK PERFORMANCE STATISTICS  
Output buffer is a queue, anchored by SSGRBUFR, of rank performance statistics. Each queue element represents data read from a single storage subsystem box that describes sets of statistics. The queue is a single threaded queue. Each queue element needs to be freed by the caller of this interface. Output buffer built for storage subsystem boxes that support rank performance statistics and are returned only if requested by caller (SSGRRANKP) on a request for performance data (SSGRRPD) for requests with scope of of subsystem (SSGIS) or all subsystems (SSGALL).  

<table>
<thead>
<tr>
<th>OFFSET</th>
<th>DECIMAL</th>
<th>TYPE</th>
<th>LENGTH</th>
<th>NAME (DIM)</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(0)</td>
<td>STRUCTURE</td>
<td>*</td>
<td>SSGRBUFR</td>
<td>Queue element header</td>
</tr>
<tr>
<td>0</td>
<td>(0)</td>
<td>CHARACTER</td>
<td>56</td>
<td>SSGRHDR</td>
<td>Length of this queue element, header plus data</td>
</tr>
<tr>
<td>4</td>
<td>(4)</td>
<td>UNSIGNED</td>
<td>4</td>
<td>SSGRQDAO</td>
<td>Offset to the start of the statistics sets</td>
</tr>
<tr>
<td>8</td>
<td>(8)</td>
<td>ADDRESS</td>
<td>4</td>
<td>SSGRFNDP</td>
<td>Pointer to next queue element, when zero end of queue</td>
</tr>
</tbody>
</table>

Orientation to subsystem that stats pertain  

- **12 (C) CHARACTER 6 SSGRVL**  Volume serial of device that stats were read from  
- **18 (12) CHARACTER 2 SSGRDEVN**  Device number of device that stats were read from  
- **20 (14) CHARACTER 8 * SSGRDEVN**  Reserved  
- **28 (1C) CHARACTER 28 SSGRBDHDR**  Rank stats info  
- **30 (1E) UNSIGNED 2 SSGRLSET**  Size of each rank stats set excluding size of array info SSGRAR  
- **32 (20) CHARACTER 6 SSGRCUT**  Control unit type  
- **38 (26) CHARACTER 3 SSGRCUM**  Control unit model  
- **41 (29) CHARACTER 10 SSGRSEQ**  Control unit sequence number  
- **51 (33) CHARACTER 1 SSGRVER**  Version of rank statistics  
- **52 (34) UNSIGNED 2 SSGRRAR**  Num of array info sets available to be returned on the ESS starting from first rank ID SSGRRID. Can be used to determine buffer requirements  
- **54 (36) UNSIGNED 2 SSGRRARSZ**  Size of each array information set SSGRAR  
- **56 (38) CHARACTER * SSGRQDA**  Rank performance stats read  
- **56 (38) CHARACTER 56 SSGRSTA(*)**  Rank statistics entry  
- **56 (38) UNSIGNED 2 SSGRRKID**  Rank Identifier  

The returned data consists of rank sets with each rank set
## Mapping for the Subsystem Get Output Buffer

For segment pool performance statistics, output buffer is a queue, anchored by SSSTOAR, of segment pool perf statistics. Each queue element represents data read from a single storage subsystem box that describes sets of statistics. The queue is a single threaded queue. Each queue element needs to be freed by the caller of this interface. Output buffer built for storage subsystem boxes that support rank performance statistics and are returned only if requested by caller (SSGSEGMP) on a request for performance data (SSGPPD) for requests with scope of subsystem (SSGSS) or all subsystems (SSGALL).

### Description of Array Information Entry

<table>
<thead>
<tr>
<th>Offset</th>
<th>Type</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>STRUCTURE</td>
<td>*</td>
<td>Rank stats entry mapping includes one or more entries for array information</td>
</tr>
<tr>
<td>0</td>
<td>UNSIGNED</td>
<td>2</td>
<td>Rank identifier</td>
</tr>
<tr>
<td>2</td>
<td>UNSIGNED</td>
<td>2</td>
<td>Segment pool number</td>
</tr>
<tr>
<td>4</td>
<td>UNSIGNED</td>
<td>1</td>
<td>Count of arrays in rank</td>
</tr>
<tr>
<td>5</td>
<td>BITSTRING</td>
<td>1</td>
<td>Rank Type Qualifier</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Data Encrypted Rank. When '1', the data stored on the physical media (i.e., disk) is encrypted.</td>
</tr>
<tr>
<td>6</td>
<td>CHARACTER</td>
<td>2</td>
<td>Not used</td>
</tr>
</tbody>
</table>

In the statistics below bytes are accumulated in units of 128 KB, and time is accumulated in units of 16 milliseconds.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Type</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>UNSIGNED</td>
<td>4</td>
<td>Rank bytes read</td>
</tr>
<tr>
<td>12</td>
<td>UNSIGNED</td>
<td>4</td>
<td>Rank bytes written</td>
</tr>
<tr>
<td>16</td>
<td>UNSIGNED</td>
<td>4</td>
<td>Rank read operations</td>
</tr>
<tr>
<td>20</td>
<td>UNSIGNED</td>
<td>4</td>
<td>Rank write operations</td>
</tr>
<tr>
<td>24</td>
<td>UNSIGNED</td>
<td>4</td>
<td>Rank read response time</td>
</tr>
<tr>
<td>28</td>
<td>UNSIGNED</td>
<td>4</td>
<td>Rank write response time</td>
</tr>
<tr>
<td>32</td>
<td>CHARACTER</td>
<td>24</td>
<td>Array information mapping</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The format of the next 24 bytes is repeated the number of times as indicated by SSGRRCNT.</td>
</tr>
<tr>
<td>32</td>
<td>UNSIGNED</td>
<td>2</td>
<td>Rank array id</td>
</tr>
<tr>
<td>34</td>
<td>CHARACTER</td>
<td>16</td>
<td>Array type in ebcidic</td>
</tr>
<tr>
<td>50</td>
<td>UNSIGNED</td>
<td>1</td>
<td>Array type</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>01 = RAID-5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>02 = RAID-10</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>03 = RAID-6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>04-FF NOT USED</td>
</tr>
<tr>
<td>51</td>
<td>UNSIGNED</td>
<td>1</td>
<td>Array width</td>
</tr>
<tr>
<td>52</td>
<td>UNSIGNED</td>
<td>1</td>
<td>Array speed (1000 RPM)</td>
</tr>
<tr>
<td>53</td>
<td>BITSTRING</td>
<td>1</td>
<td>Array Device Class and Array Status</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>01... Raid Degraded. One or more array members need rebuilding</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>01... DDM Throttling. A Near-line DDM in the array is throttling performance due to temperature or workload.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>01... RPM Exception. A DDM with an slower RPM than the normal array DDMs is a member of the array as a result of a sparing action.</td>
</tr>
<tr>
<td>54</td>
<td>CHARACTER</td>
<td>2</td>
<td>Array capacity (GB)</td>
</tr>
</tbody>
</table>

### Additional Notes

- **OFFSET OFFSET**: Contains the offset in bytes from the beginning of the structure.
- **HEX**: The hexadecimal representation of the offset.
- **TYPE**: The type of data stored at the offset.
- **LENGTH**: The length of the data in bytes.
- **NAME (DIM)**: The name of the data and its dimensions.
Orientation to subsystem that stats pertain

---

<table>
<thead>
<tr>
<th>OFFSET</th>
<th>DECIMAL</th>
<th>TYPE</th>
<th>LENGTH</th>
<th>NAME (DIM)</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (0)</td>
<td>STRUCTURE</td>
<td>52</td>
<td>SSGSSSTT</td>
<td>seg pool statistics entry</td>
<td>Includes one or more entry</td>
</tr>
<tr>
<td>0 (0)</td>
<td>UNSIGNED</td>
<td>2</td>
<td>SGGSSPID</td>
<td>Segment pool type</td>
<td>00-03 = reserved</td>
</tr>
<tr>
<td>2 (2)</td>
<td>UNSIGNED</td>
<td>1</td>
<td>SGGSSLPT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 (3)</td>
<td>BITSTRING</td>
<td>1</td>
<td>SGGSSPTQ</td>
<td>Extent Pool Type Qualifier</td>
<td>Data Encrypted Extent Pool. When ‘1’, the data stored on the physical media (i.e disk) is encrypted.</td>
</tr>
<tr>
<td>4 (4)</td>
<td>UNSIGNED</td>
<td>4</td>
<td>SGGSSCAP</td>
<td>Real seg pool capacity (GB)</td>
<td>00-03 = reserved</td>
</tr>
</tbody>
</table>

---

Chapter 12. User Access to Subsystem Statistics, Status, and Counts Information
MAPPING FOR THE SPACE EFFICIENT VOLUME STATUS OUTPUT BUFFER

Output buffer is a queue, anchored by SSGSEDBF, of Space Efficient Volume (SEVOL) status. Each queue element represents status of the space efficient volume specified in the header. The queue is a single threaded queue. Each queue element needs to be freed by the caller of this interface.

Output buffer is built for SEVOL(s) when SEVOL status is requested (SSGSEV=ON).

Caller sets the scope of SEVOL(s) by turning one of flag bit that is the single device (SSGDEV), subsystem (SSG1SS), or all subsystems (SSGALL). For a single SEVOL status request, the SEVOL status can be obtain when the path of SEVOL is available. For a subsystem (SSG1SS) and all systems (SSGALL) scope, the SEVOL status is available when SEVOL is ONLINE. When the scope is the single device (SSGDEV=ON) and the Fix Block SEVOL is requested (SSGURFBD=ON), the status of FB SEVOL oriented by LSS (SSGFBLSS) and Device (SSGFBDEV) is reported. FB SEVOL status is not available with the scope of the subsystem (SSG1SS=ON) or the all subsystems (SSGALL=ON). FB SEVOL status is available when LSS number of both the I/O device (SSGAVOL or SSGDDN) and FB SEVOL are numbered in the same even or odd number.

Flag and variable combinations of SEVOL status request are shown as below in the railroad chart format.

>> _SSGSEV_ _SSGALL________ ________________> (A)
  | _SSG1SS________|
  | _SSGDEV_ ____________________ __> (B)
  | _SSGFBLSS__SSGFBDEV_|

(A)> _SSGAVOL__SSGUNIT__ __ __________________><
  | _SSGADDN___________|
  | _SSGAVOL__SSGOFFL__|

Flag and variable combinations of SEVOL status request are shown as below in the railroad chart format.

(B)> __ _SSGAVOL__SSGUNIT__ _|
  | _SSGADDN___________|
  | _SSGAVOL__SSGOFFL__|

MAPPING FOR THE SPACE EFFICIENT VOLUME STATUS OUTPUT BUFFER

<table>
<thead>
<tr>
<th>OFFSET</th>
<th>OFFSET</th>
<th>TYPE</th>
<th>LENGTH</th>
<th>NAME (DIM)</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (0)</td>
<td>STRUCTURE</td>
<td>84</td>
<td>SSGSEBUF</td>
<td>Queue element header</td>
<td></td>
</tr>
<tr>
<td>0 (0)</td>
<td>CHARACTER</td>
<td>56</td>
<td>SSGSEHDR</td>
<td>Length of this queue element, header plus data</td>
<td></td>
</tr>
<tr>
<td>0 (0)</td>
<td>UNSIGNED</td>
<td>4</td>
<td>SSGSEQAL</td>
<td>Offset to the start of the statistics sets</td>
<td></td>
</tr>
<tr>
<td>4 (4)</td>
<td>UNSIGNED</td>
<td>4</td>
<td>SSGSEQAO</td>
<td>Pointer to next queue element, when zero end of queue</td>
<td></td>
</tr>
<tr>
<td>8 (4)</td>
<td>ADDRESS</td>
<td>4</td>
<td>SSGSEFWP</td>
<td>Orientation to device that status pertain</td>
<td></td>
</tr>
</tbody>
</table>

Volume serial of device that stat where read from

Device number of device that Status where read from

Subsystem ID

not used

Total SE VOL Count, only valid for head of buffer queue

SE VOL header

Size of data

Control unit type

Control unit model

Control unit maker /
Chapter 12. User Access to Subsystem Statistics, Status, and Counts Information 433
Orientation to device that stats pertain

12  (C) CHARACTER 6  SSGEPVOL Volume serial of device that status where read from
18  (12) CHARACTER 2  SSGEPDEV Device number of device that stats where read from
20  (14) UNSIGNED 2  SSGEPIDC Ext Pool ID provided by caller
22  (16) CHARACTER 6  * Reserved
28  (1C) ADDRESS 4  SSGEPSIP Internal code use only. Branch Entry caller shall set zero.
32  (20) UNSIGNED 2  SSGEPSDO Offset to 1st Summary data (SSGEPSUM) array
34  (22) UNSIGNED 2  SSGEPDDO Offset to Detail data (SSGEPDAT)
36  (24) CHARACTER 32  SSGEPDHL Fixed length of epc data header
36  (24) CHARACTER 32  SSGEPDAD Ext Pool Config status
36  (24) CHARACTER 32  SSGEPLEN Length of data returned
38  (26) UNSIGNED 2  SSGEPIDV Ext Pool detailed data valid
40  (28) BITSTRING 2  SSGEPIDV Ext Pool Flags
1... .... SSGEPFB 0=CKD Ext Pool, 1=FB Ext Pool
1... .... SSGEPRC Ext Pool repository configured
1... .... SSGEOPV Over Provisioned volumes in Ext pool
...1 .... * not used
...1 .... * not used
...1 .... SSGEPWP Ext Pool at warning

The format of the next eight bytes is repeated the number times as indicated by SSGEPCNT.

42  (2A) CHARACTER 6  SSGEPCUT Control unit type
48  (30) CHARACTER 3  SSGEPCUM Control unit model
51  (33) CHARACTER 3  SSGEPCUC Control unit maker/company
54  (36) CHARACTER 2  SSGEPCUP Control unit plant
56  (38) CHARACTER 12 SSGEPSEQ Control unit sequence number

The following two fields are a remaining percentage of usable space that will initiate a notification to the host

68  (44) CHARACTER 8  SSGEPSUM Ext Pool summary data
68  (44) CHARACTER 2  SSGEPLSS Ext Pool summary data

*not used

...1 .... SSGEPWP Ext Pool at warning
The following fields contain detailed Extent Pool configuration information for Extent Pool identified by Ext Pool ID specified by caller. The address of SSGEPDAT is the sum of SSGEPAR and SSGEPDDO.

<table>
<thead>
<tr>
<th>DECIMAL HEX</th>
<th>OFFSET</th>
<th>TYPE</th>
<th>LENGTH</th>
<th>NAME (DIM)</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>CHARACTER</td>
<td>12316</td>
<td>SSGEPDAT</td>
<td>Ext Pool ID for detailed data</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>CHARACTER</td>
<td>2</td>
<td>SSGEPIDD</td>
<td>Ext Pool ID identified by Ext Pool ID specified by caller</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>BITSTRING</td>
<td>1</td>
<td>SSGEPDFG</td>
<td>Allocated Space in Repository is not available</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>CHARACTER</td>
<td>1</td>
<td>SSGEPARN</td>
<td>Repository at warning percent</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>CHARACTER</td>
<td>1</td>
<td>SSGEPRAW</td>
<td>Repository full</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>UNSIGNED</td>
<td>4</td>
<td>SSGEPSZ</td>
<td>Ext Pool size</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>UNSIGNED</td>
<td>4</td>
<td>SSGEALOC</td>
<td>Space currently allocated in Ext Pool</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>UNSIGNED</td>
<td>4</td>
<td>SSGEPSZ</td>
<td>Size of Ext Pool Repository, this set to 0 when SSGEPRC=0</td>
</tr>
<tr>
<td>16</td>
<td>16</td>
<td>UNSIGNED</td>
<td>4</td>
<td>SSGEPRAL</td>
<td>Space currently allocated in Ext Pool Repository, set to 0 when SSGEPRC=0</td>
</tr>
<tr>
<td>20</td>
<td>20</td>
<td>UNSIGNED</td>
<td>4</td>
<td>SSGEPGSZ</td>
<td>Amount of guaranteed space in Repository. Set to 0 when SSGEPRC=0</td>
</tr>
<tr>
<td>24</td>
<td>24</td>
<td>CHARACTER</td>
<td>4</td>
<td>SSGEPRC</td>
<td>not used</td>
</tr>
</tbody>
</table>

The following fields are expressed in number of cylinders for CKD device. The following fields are expressed in TENTH of binary Gigabytes for Fix Block device.

<table>
<thead>
<tr>
<th>DECIMAL HEX</th>
<th>OFFSET</th>
<th>TYPE</th>
<th>LENGTH</th>
<th>NAME (DIM)</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>4</td>
<td>UNSIGNED</td>
<td>4</td>
<td>SSGEPSZ</td>
<td>Ext Pool size</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>UNSIGNED</td>
<td>4</td>
<td>SSGEALOC</td>
<td>Space currently allocated in Ext Pool</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>UNSIGNED</td>
<td>4</td>
<td>SSGEPSZ</td>
<td>Size of Ext Pool Repository, this set to 0 when SSGEPRC=0</td>
</tr>
<tr>
<td>16</td>
<td>16</td>
<td>UNSIGNED</td>
<td>4</td>
<td>SSGEPRAL</td>
<td>Space currently allocated in Ext Pool Repository, set to 0 when SSGEPRC=0</td>
</tr>
<tr>
<td>20</td>
<td>20</td>
<td>UNSIGNED</td>
<td>4</td>
<td>SSGEPGSZ</td>
<td>Amount of guaranteed space in Repository. Set to 0 when SSGEPRC=0</td>
</tr>
<tr>
<td>24</td>
<td>24</td>
<td>CHARACTER</td>
<td>4</td>
<td>SSGEPRC</td>
<td>not used</td>
</tr>
</tbody>
</table>

The next three fields are 32k bitmaps

<table>
<thead>
<tr>
<th>DECIMAL HEX</th>
<th>OFFSET</th>
<th>TYPE</th>
<th>LENGTH</th>
<th>NAME (DIM)</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>28</td>
<td>BITSTRING</td>
<td>4096</td>
<td>SSGENORM</td>
<td>Normally provisioned volumes</td>
</tr>
<tr>
<td>4124</td>
<td>4124</td>
<td>BITSTRING</td>
<td>4096</td>
<td>SSGEPOPV</td>
<td>Over provisioned volumes</td>
</tr>
<tr>
<td>8220</td>
<td>8220</td>
<td>BITSTRING</td>
<td>4096</td>
<td>SSGEPESEV</td>
<td>Space Efficient volumes</td>
</tr>
</tbody>
</table>

MAPPING FOR THE DEVICE PERFORMANCE STATISTICS

Output buffer

Output buffer is anchored by SSGOADDR, of a device performance statistics output table. Output buffer needs to be freed by the caller of this interface. Output buffer is built for a device that supports device performance statistics and returned only if requested by caller for performance data (SSGRPD) for a request with scope of of device (SSGDEV). The parameter list extension must also be passed as indicated with SSGARGL2.
<table>
<thead>
<tr>
<th>Field</th>
<th>Size</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSGDPLN</td>
<td>2</td>
<td>SIGNED</td>
<td>Data Length</td>
</tr>
<tr>
<td>SSGDPSA</td>
<td>192</td>
<td>DEVICE PERFORMANCE</td>
<td>Device Performance Statistics Data Area</td>
</tr>
<tr>
<td>SSGDPFTF</td>
<td>1</td>
<td>BITSTRING</td>
<td>Bit 1: Cache Storage unavailable (Set to zero for 2107/1750)</td>
</tr>
<tr>
<td>SSGDPNAV</td>
<td>1...</td>
<td>BITSTRING</td>
<td>Bit 1: Cache Storage unavailable (Set to zero for 2107/1750)</td>
</tr>
<tr>
<td>SSGDPPDR</td>
<td>1..11</td>
<td>BITSTRING</td>
<td>Mask bits for Format of Perf Data Returned Value after masking (SSGDPPFTF &amp; SSGDPPDR)</td>
</tr>
<tr>
<td>SSGDPSD</td>
<td>.111</td>
<td>BITSTRING</td>
<td>The same definition and values of bit 4-7 of byte 0 from Sense Subsystem status.</td>
</tr>
<tr>
<td>SSGPDUUA</td>
<td>1</td>
<td>CHARACTER</td>
<td>Device Unit Address of the device to which the statistics pertain</td>
</tr>
<tr>
<td>SSGPDS</td>
<td>2</td>
<td>CHARACTER</td>
<td>Device Status. The same as Sense Subsystem Status in bytes 26-27</td>
</tr>
<tr>
<td>SSGDRNI</td>
<td>4</td>
<td>UNSIGNED</td>
<td>Search/Read Normal I/O Requests. The number of command chains which meet the following requirements.</td>
</tr>
<tr>
<td>SSGDRNH</td>
<td>4</td>
<td>UNSIGNED</td>
<td>Search/Read Normal I/O Request Hits. The number of command chains which meet the following requirements.</td>
</tr>
<tr>
<td>SSGPWNH</td>
<td>4</td>
<td>UNSIGNED</td>
<td>Write Normal I/O Requests. The number of command chains which meet the following requirements.</td>
</tr>
<tr>
<td>SSGPFWH</td>
<td>4</td>
<td>UNSIGNED</td>
<td>DASD Fast Write I/O Request Hits. The number of command chains which meet the following requirements.</td>
</tr>
</tbody>
</table>
- The chain contained at least one write command.
- The chain was completed without requiring access to any DDM.

36 (24) UNSIGNED 4 SSGDPRSI
Search/Read Sequential I/O Requests. The number of sequential mode command chains which meet the following requirements.
- The chain contained at least one search or read command but no write commands.

40 (28) UNSIGNED 4 SSGDPRSH
Search/Read Sequential I/O Request Hits. The number of sequential mode command chains which meet the following requirements.
- The chain contained at least one search or read command but no write commands.
- The chain was completed without requiring access to any DDM.

44 (2C) UNSIGNED 4 SSGDPWSI
Write Sequential I/O Requests. The number of sequential mode command chains which meet the following requirements.
- The chain contained at least one write command.

48 (30) UNSIGNED 4 SSGDPWSH
Fast Write Sequential I/O Request Hits. The number of sequential write operations that did not require movement of data to or from a storage device before completion of the I/O operation.

52 (34) UNSIGNED 4 SSGDPRCF
Search/Read Cache Fast Write I/O Requests. The number of command chains which meet the following requirements.
- The chain included a Define Extent command which specified the Cache Fast Write Data attribute.
- The chain contained at least one search or read command but no write commands.
- Cache Fast Write Data must be activated for the subsystem.

56 (38) UNSIGNED 4 SSGDPRCH
Search/Read Cache Fast Write I/O Request Hits. The number of command chains which meet the following requirements.
- The chain included a Define Extent command which specified the Cache Fast Write Data attribute.
- The chain contained at least one search or read command but no write commands.
- Cache Fast Write Data must be actuated for the subsystem.
- The chain was completed without requiring access to any DDM.

60 (3C) UNSIGNED 4 SSGDPDFW
Cache Fast Write I/O Requests. The number of command chains which meet the following requirements.
- The chain included a Define Extent command which
<table>
<thead>
<tr>
<th>Offset</th>
<th>Size</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>40</td>
<td>UNSIGNED 4</td>
<td>SSGPCFH Cache Fast Write I/O Request Hits. The number of command chains which meet the following requirements. - The chain contained at least one write command. - Cache Fast Write Data must be actuated for the subsystem. - The chain was completed without requiring access to any DDM.</td>
</tr>
<tr>
<td>68</td>
<td>44</td>
<td>UNSIGNED 4</td>
<td>SSGPCL Inhibit Cache Loading I/O Request that operate with DASD (Set to zeros for 2107/1750)</td>
</tr>
<tr>
<td>72</td>
<td>48</td>
<td>UNSIGNED 4</td>
<td>SSGPBCI Bypass Cache I/O Requests (Set to zeros for 2107/1750)</td>
</tr>
<tr>
<td>76</td>
<td>50</td>
<td>UNSIGNED 4</td>
<td>SSGPSDC Seq DASD to Cache Transfer Operations</td>
</tr>
<tr>
<td>80</td>
<td>54</td>
<td>UNSIGNED 4</td>
<td>SSGPDCD DASD to Cache Transfer Operation Count</td>
</tr>
<tr>
<td>84</td>
<td>58</td>
<td>UNSIGNED 4</td>
<td>SSGPFWD DASD Fast Write Operation Delayed Due to non-volatile storage Space constraints</td>
</tr>
<tr>
<td>92</td>
<td>60</td>
<td>UNSIGNED 4</td>
<td>SSGPFW Normal 'DASD Fast Write' Write Operation Counts. The number of command chains which meet the following requirements. - The chain is not part of a sequential operation. - The chain did not include a Define Extent command which specified Cache Fast Write Data. - The chain contained at least one write command. - The chain was completed without requiring access to any DDM.</td>
</tr>
<tr>
<td>96</td>
<td>64</td>
<td>UNSIGNED 4</td>
<td>SSGPSFW Sequential Access 'DASD Fast Write' Write Operation Counts</td>
</tr>
<tr>
<td>100</td>
<td>68</td>
<td>UNSIGNED 4</td>
<td>SSGPNRM Number of record cache Read misses</td>
</tr>
<tr>
<td>104</td>
<td>CHARACTER 1</td>
<td>SSGPDS2 Device Status - Group 2 The device status, SSGPDS2, is identical to the data returned by the Sense Subsystem Status in Byte 36. Note: The format of the data in SSGPDS2 is determined by the value in SSGPFR.</td>
<td></td>
</tr>
<tr>
<td>105</td>
<td>69</td>
<td>UNSIGNED 4</td>
<td>SSGPQWP Quick Write Promotes Flag byte SSGPQWP describes the validity of data returned and the increment values used for certain counters</td>
</tr>
<tr>
<td>109</td>
<td>BITSTRING 1</td>
<td>SSGPVD Mask for reserved bits</td>
<td></td>
</tr>
</tbody>
</table>

Note: The format of the data in SSGPDS2 is determined by the value in SSGPFR.
SSGDPSID are valid.

Mask bits for Units
Value after masking
(SSGDPSID & SSGDPUNI)

'00'b Unit of SSGDPRRT, SSGDPWAT, SSGDPA, SSGDPRAT,
SSGPBR, SSGDPRAH, and SSGDPTA is
16 milliseconds.
Unit of SSGDPPBB
SSGDPBEST is
128 KB.

'01'b Reserved
'10'b Reserved
'11'b Reserved

SSGDPSBYV
1 indicates SSGDPRRT, SSGDPPBB, SSGDPA, SSGDPRAT,
and SSGDPTA are valid
SSID

110 (6E) CHARACTER 2 SSGDPSID
112 (70) UNSIGNED 4 SSGDPITA
Irregular Track Accesses
116 (74) UNSIGNED 4 SSGDPTH
Irregular Track Access Hits
120 (78) UNSIGNED 4 SSGDPODC
Operation Delayed Due To Cache Space Constraints
124 (7C) UNSIGNED 4 SSGDPMCL
Milliseconds of lower interface I/O activity for the indicated device
(Set to zeros for 2107/1750)

128 (80) BITSTRING 1 SSGDPSVI
Non-volume information contain in the record
Value Definition
0 No Additional Information
1 RAID Rank Information
2 Segment Pool and Physical Storage Counter information
3-255 Not Defined

129 (81) CHARACTER 1 SSGDPRVB
Not used
130 (82) CHARACTER 2 SSGDPSPI
Segment Pool ID
132 (84) CHARACTER 1 SSGDPSTY
Non-volume information contained in the record
Segment Type
00-03 Reserved
04 FB 1GB
05-131 Reserved
132 CKD 1GB
133-255 Reserved

133 (85) BITSTRING 1 SSGDPSFL
Segment Flags
1... Dynamic Segment Allocation
.... Migrating/Migration Error State

134 (86) CHARACTER 2 * Not used
136 (88) UNSIGNED 4 SSGDPPRD
Physical Storage Read Operations
140 (8C) UNSIGNED 4 SSGDPMWD
Physical Storage Write Operations
144 (90) UNSIGNED 4 SSGDPPBD
Physical Storage Bytes Read. See Increment Value in SSGDPPVD
148 (94) UNSIGNED 4 SSGDPPBW
Physical Storage Bytes Written. See Increment Value in SSGDPPVD
152 (98) UNSIGNED 4 SSGDPRMR
Record Mode Read Operations
156 (9C) UNSIGNED 4 SSGDPMTR
Number of tracks read from the concurrent Copy or XRC Sidefile
160 (A0) UNSIGNED 4 SSGDPMCW
Number of contaminating writes for a Concurrent Copy or XRC volume as a result of:
- Update to a Concurrent Copy protected track.
- Update to an XRC monitored track.

164 (A4) UNSIGNED 4 SSGDPTTS
Number of tracks or portion of tracks that were transferred to

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>168 (A8)</td>
<td>UNSIGNED</td>
<td>4</td>
<td>SSGDPNSA NYS Space Allocations</td>
</tr>
<tr>
<td>172 (AC)</td>
<td>UNSIGNED</td>
<td>4</td>
<td>SSGDPRRT Physical Storage Read Response Time. See Increment Value in SSGDPDVD</td>
</tr>
<tr>
<td>176 (B0)</td>
<td>UNSIGNED</td>
<td>4</td>
<td>SSGDPWRT Physical Storage Write Response Time. See Increment Value in SSGDPDVD</td>
</tr>
<tr>
<td>180 (B4)</td>
<td>UNSIGNED</td>
<td>4</td>
<td>SSGDPBRD Bytes Read. See Increment Value in SSGDPDVD</td>
</tr>
<tr>
<td>184 (B8)</td>
<td>UNSIGNED</td>
<td>4</td>
<td>SSGDPBWR Bytes Written. See Increment Value in SSGDPDVD</td>
</tr>
<tr>
<td>188 (BC)</td>
<td>UNSIGNED</td>
<td>4</td>
<td>SSGDPRAT Read Accumulated Time. The accumulated response time for all read operations. See Increment Value in SSGDPDVD</td>
</tr>
<tr>
<td>192 (C0)</td>
<td>UNSIGNED</td>
<td>4</td>
<td>SSGDPWAT Write Accumulated Time. The accumulated response time for all write operations. See Increment Value in SSGDPDVD</td>
</tr>
<tr>
<td>196 (C4)</td>
<td>UNSIGNED</td>
<td>4</td>
<td>SSGDPZHR zHPF Read I/O Requests. This count is the number of Transport Mode read command chains. The Read I/O Request will also be counted in one of the following counters: - SSGDPRNI, Search/Read Normal I/O Requests - SSGDRPSI, Search/Read Sequential I/O Requests - SSGDPRCF, Search/Read Cache Fast Write I/O Requests</td>
</tr>
<tr>
<td>200 (C8)</td>
<td>UNSIGNED</td>
<td>4</td>
<td>SSGDPZHW zHPF Write I/O Requests. This count is the number of Transport Mode write command chains. The Write I/O Request will also be counted in one of the following counters: - SSGDPRNI, Write Normal I/O Requests - SSGDPWSI, Write Sequential I/O Requests - SSGDPCFW, Cache Fast Write I/O Requests</td>
</tr>
<tr>
<td>204 (CC)</td>
<td>CHARACTER</td>
<td>4</td>
<td>SSGDPRVX Not used - zeros</td>
</tr>
<tr>
<td>208 (DD)</td>
<td>UNSIGNED</td>
<td>4</td>
<td>SSGDPZHL zHPF List Pre-fetch I/O Requests. This count is the number of command chains which meet the following requirements: - The Transport Mode operation specified a non-zero Imbedded Locate Record Count.</td>
</tr>
<tr>
<td>212 (D4)</td>
<td>UNSIGNED</td>
<td>4</td>
<td>SSGDPZHH zHPF List Pre-fetch I/O Requests Hits. This count is the number of common chains which meet the following requirements: - The Transport Mode operation specified a non-zero Imbedded Locate Record Count. - The chain was completed without requiring access to any DDM.</td>
</tr>
<tr>
<td>216 (DB)</td>
<td>UNSIGNED</td>
<td>4</td>
<td>SSGDPGSF Global Mirror Collisions Sidefile Count. A GM collision occurs when, during the sending of data to the secondary to create a consistency group, a subsequent host update is attempted before the modified track has been transmitted to the secondary volume. The modified track will be moved</td>
</tr>
</tbody>
</table>
to the sidefile before allowing a new host write. This counter will be incremented by one when a track is added to the sidefile.

Global Mirror Collisions Send Synchronous Count. When a write collision occurs, the modified track data which belongs to the current consistency group may be sent to the remote control unit before allowing the write. The data may come from the sidefile if it is full or from cache if the collision sidefile is not being utilized.

RETURN CODES FROM IDCSS01

REASON CODES FROM ESS (BOX) PERFORMANCE STATISTICS
(SSSLPRET, SSGRSRET, SSGSGRET).

Chapter 12. User Access to Subsystem Statistics, Status, and Counts Information 441
Appendix A. Control Blocks

The following control blocks are described here:

- “Data Extent Block (DEB) Fields” for EXCP and EXCPVR.
- “Data Facilities Area (DFA) Fields” on page 449

Data Extent Block (DEB) Fields

The data extent block (DEB) fields shown here should only be used with EXCP and EXCPVR (see z/OS DFSMSdfp Diagnosis 53 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 basic section.
- DEBDASD (DSECT name for direct access section)
- DEBACSMD (DSECT name for access method sections)
- DEBSUBBNM (DSECT name for subroutine name section)
- DEBXTN (DSECT name for DEB extension)

Owning Component:
Data Management, subcomponent OPEN/CLOSE/EOV

Eye-Catcher ID:
None

Subpool and Key:
230 and key 5

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)

Serialization:
LOCAL lock serializes the placing of a DEB on the TCB DEB chain and in the DEB table. OPEN/CLOSE/EOV processing is serialized by local lock and DEBCHK.

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.

Table 97. Partial Listing of DEB Fields

<table>
<thead>
<tr>
<th>Offset</th>
<th>Type</th>
<th>Length</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0(0)</td>
<td>ADDRESS</td>
<td>4</td>
<td>DEBEOEA(0)</td>
<td>Address of end-of-extent appendage routine</td>
</tr>
<tr>
<td>0(0)</td>
<td>BITSTRING</td>
<td>1</td>
<td>DEBEOEAB</td>
<td>Flag byte</td>
</tr>
<tr>
<td>. . . xxx</td>
<td>DEBEOENP</td>
<td></td>
<td></td>
<td>X'0F'– number of 2K pages to be fixed for the end–of–extent appendage</td>
</tr>
<tr>
<td>1(1)</td>
<td>ADDRESS</td>
<td>3</td>
<td>DEBEOEAD</td>
<td>Address of end-of-extent appendage routine</td>
</tr>
<tr>
<td>4(4)</td>
<td>ADDRESS</td>
<td>4</td>
<td>DEBSIOA(0)</td>
<td>Address of start 1/O appendage routine</td>
</tr>
</tbody>
</table>
### Control Blocks

#### Table 97. Partial Listing of DEB Fields (continued)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Type</th>
<th>Length</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4(4)</td>
<td>BITSTRING</td>
<td>1</td>
<td>DEBSIOAB</td>
<td>Flag byte</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DEBGFX</td>
<td>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</td>
</tr>
<tr>
<td>.1.</td>
<td></td>
<td></td>
<td>DEBSIOX</td>
<td>X'40'– if zero, do not enter SIO appendage when ERP IS active. If one, enter SIO appendage even when ERP is active.</td>
</tr>
<tr>
<td>.1.1.</td>
<td></td>
<td></td>
<td>DEBIOVR</td>
<td>X'20'– If one, EXCPVR request is valid. If zero, EXCPVR request is invalid and will not be executed. Currently has no effect.</td>
</tr>
<tr>
<td>.1.1.</td>
<td></td>
<td></td>
<td>DEBFIX</td>
<td>X'10' Indication that DEB has been fixed</td>
</tr>
<tr>
<td>.1.1.x</td>
<td></td>
<td></td>
<td>DEBSIONP</td>
<td>X'0F'– number of 2K pages to be fixed for the SIO appendage</td>
</tr>
<tr>
<td>5(5)</td>
<td>ADDRESS</td>
<td>3</td>
<td>DEBSIOAD</td>
<td>Address of start I/O appendage routine</td>
</tr>
<tr>
<td>8(8)</td>
<td>ADDRESS</td>
<td>4</td>
<td>DEBPCIAD</td>
<td>Address of PCI appendage routine</td>
</tr>
<tr>
<td>8(8)</td>
<td>BITSTRING</td>
<td>1</td>
<td>DEBPCIAB</td>
<td>Flag byte</td>
</tr>
<tr>
<td>.1.x</td>
<td></td>
<td></td>
<td>DEBPCINP</td>
<td>X'0F'– number of 2K pages to be fixed for the PCI appendage</td>
</tr>
<tr>
<td>9(9)</td>
<td>ADDRESS</td>
<td>3</td>
<td>DEBPCIAD</td>
<td>Address of program controlled interruption (PCI) appendage routine</td>
</tr>
<tr>
<td>12(C)</td>
<td>ADDRESS</td>
<td>4</td>
<td>DEBCEA(0)</td>
<td>Address of channel end appendage routine</td>
</tr>
<tr>
<td>12(C)</td>
<td>BITSTRING</td>
<td>1</td>
<td>DEBCEAB</td>
<td>Flag byte</td>
</tr>
<tr>
<td>.1.x</td>
<td></td>
<td></td>
<td>DEBESMVR</td>
<td>X'80', x' validity check for EXCPVR caller</td>
</tr>
<tr>
<td>.1.x</td>
<td></td>
<td></td>
<td>DEBCENP</td>
<td>X'0F'– number of 2K pages to be fixed for the channel-end appendage</td>
</tr>
<tr>
<td>13(D)</td>
<td>ADDRESS</td>
<td>3</td>
<td>DEBCEAD</td>
<td>Address of channel end appendage routine</td>
</tr>
<tr>
<td>16(10)</td>
<td>ADDRESS</td>
<td>4</td>
<td>DEBXCEA(0)</td>
<td>Address of abnormal end appendage routine</td>
</tr>
<tr>
<td>16(10)</td>
<td>BITSTRING</td>
<td>1</td>
<td>DEBXCEAB</td>
<td>Flag byte</td>
</tr>
<tr>
<td>.1.x</td>
<td></td>
<td></td>
<td>DEBXCEAD</td>
<td>Address of abnormal end appendage routine</td>
</tr>
<tr>
<td>X&quot;</td>
<td></td>
<td></td>
<td>DEBAVTE</td>
<td>&quot;&quot; End of appendage vector table</td>
</tr>
</tbody>
</table>

**DEB PREFIX TABLE.** Addressable as negative offset from DEB basic section. Not necessarily contiguous with DEB appendage vector table.

-16(10)  | 16 | DEBPREFIX(0) | DEB prefix table |
-16(10)  | BITSTRING | 1 | DEBKAR | 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. If it exceeds 2040 bytes (X'FF' doublewords), the excess bits are ignored. |
-3(3)    | CHARACTER | 1 | DEBAMTY | Access method type |
| X'0'    | DEBAMTNY | '0' | Access method type not known |
| X'1'    | DEBAMVSM | '1' | VSAM access method type |
| X'2'    | DEBAMXCP | '2' | EXCP access method type |
| X'4'    |         | Reserved |
| 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'81'   | DEBAMSUB | '129' | Subsystem access method type |
| X'82'   | DEBAMVMT | '130' | VTAM® access method type |
| X'84'   |         | Reserved |
-2(2)    | UNSIGNED  | 2 | DEBTBLOF | For system use |
| X'24'   | DEBPREFE | "" | End of DEB prefix table |

**DEB basic section**

0(0)    | ADDRESS   | 4 | DEBTCBAD(0) | Address of TCB for this DEB |
0(0)    | BITSTRING | 1 | DEBNMSUB | Number of subroutines loaded by OPEN executor routines and identified in the subroutine name section (DEBSUBID) |
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 |
### Table 97. Partial Listing of DEB Fields (continued)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Type</th>
<th>Length</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4(4)</td>
<td>BITSTRING</td>
<td>1</td>
<td>DEBAMLNG</td>
<td>Number of bytes in the access method dependent section. For BDAM this field contains the length expressed in number of words.</td>
</tr>
<tr>
<td>5(5)</td>
<td>ADDRESS</td>
<td>3</td>
<td>DEBDEBB</td>
<td>Address of the next DEB in the same task</td>
</tr>
<tr>
<td>8(6)</td>
<td>ADDRESS</td>
<td>4</td>
<td>DEBIRBAD(0)</td>
<td>IRB storage address used for appendage asynchronous exits</td>
</tr>
<tr>
<td>8(8)</td>
<td>BITSTRING</td>
<td>1</td>
<td>DEBOFLGS</td>
<td>Data set status flags</td>
</tr>
<tr>
<td>xx</td>
<td></td>
<td></td>
<td>DEBDISP</td>
<td>X'C0'– data set disposition flags bit setting disposition</td>
</tr>
<tr>
<td>01</td>
<td></td>
<td></td>
<td>DEBDSDOLD</td>
<td>X'40' old data set</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td>DEBDSDMOD</td>
<td>X'80' mod data set</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td>DEBDNSNEW</td>
<td>X'C0' new data set</td>
</tr>
<tr>
<td>.1</td>
<td></td>
<td></td>
<td>DEBEOF</td>
<td>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)</td>
</tr>
<tr>
<td>.1</td>
<td></td>
<td></td>
<td>DEBRRLSE</td>
<td>X'10'– release unused external storage (DASD) emulator tape with second generation format. Tape might contain blocks shorter than 12 characters. (tape)</td>
</tr>
<tr>
<td>.1</td>
<td></td>
<td></td>
<td>DEBSPLIT</td>
<td>X'04'–7–track emulator tape with possible mixed parity records (tape)</td>
</tr>
<tr>
<td>.1</td>
<td></td>
<td></td>
<td>DEBLSLM</td>
<td>X'02'– nonstandard labels</td>
</tr>
<tr>
<td>.1</td>
<td></td>
<td></td>
<td>DEBERR</td>
<td>X'01'– use reduced error recovery procedure (tape) concatenated partitioned organization data sets processed using BPAM (DASD)</td>
</tr>
<tr>
<td>9(9)</td>
<td>ADDRESS</td>
<td>3</td>
<td>DEBIRBB</td>
<td>IRB storage address used for appendage asynchronous exits</td>
</tr>
<tr>
<td>12(C)</td>
<td>BITSTRING</td>
<td>1</td>
<td>DEBOPATB</td>
<td>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</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>DEBABEND</td>
<td>X'80'– set by abend indicating a sysabend or sysdump data set</td>
</tr>
<tr>
<td>.1</td>
<td></td>
<td></td>
<td>DEBZERO</td>
<td>X'40'– always zero</td>
</tr>
<tr>
<td>.xx</td>
<td></td>
<td></td>
<td>DEBPOSIT</td>
<td>X'30'– data set positioning flags bit setting positioning</td>
</tr>
<tr>
<td>.01</td>
<td></td>
<td></td>
<td>DEBERED</td>
<td>X'10' reread</td>
</tr>
<tr>
<td>.1</td>
<td></td>
<td></td>
<td>DEBLEAVE</td>
<td>X'30' leave</td>
</tr>
<tr>
<td>.xxx</td>
<td></td>
<td></td>
<td>DEBACCS</td>
<td>X'0E'– type of I/O accessing being done bit setting accessing</td>
</tr>
<tr>
<td>.0000</td>
<td></td>
<td></td>
<td>DEBINPT</td>
<td>X'0' INPUT</td>
</tr>
<tr>
<td>.111</td>
<td></td>
<td></td>
<td>DEBOUTPT</td>
<td>X'F' OUTPUT or EXTEND</td>
</tr>
<tr>
<td>.0011</td>
<td></td>
<td></td>
<td>DEBINOUT</td>
<td>X'3' INOUT</td>
</tr>
<tr>
<td>.011</td>
<td></td>
<td></td>
<td>DEBOUTIN</td>
<td>X'7' OUTIN or OUTINX</td>
</tr>
<tr>
<td>0001</td>
<td></td>
<td></td>
<td>DEBDRBCK</td>
<td>X'1' RDBACK</td>
</tr>
<tr>
<td>0100</td>
<td></td>
<td></td>
<td>DEBUAD</td>
<td>X'4' UPDAT</td>
</tr>
<tr>
<td>13(D)</td>
<td>BITSTRING</td>
<td>1</td>
<td>DEBQSCNT</td>
<td>PURGE (SVC 16) – Quiesce count. Number of devices executing user's channel programs, as shown by bits 5 and 6 of UCBFL1 fields.</td>
</tr>
<tr>
<td>14(E)</td>
<td>BITSTRING</td>
<td>1</td>
<td>DEBFLGS1</td>
<td>Flag field</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>DEBPWCKD</td>
<td>X'80'– password was supplied during OPEN, EOV will not request a password for each additional volume of a multivolume data set.</td>
</tr>
<tr>
<td>.1</td>
<td></td>
<td></td>
<td>DEBEOFDF</td>
<td>X'40'– set by EOV to inform close that an end of file has been encountered and, therefore, deferred user label processing is allowed.</td>
</tr>
<tr>
<td>.1</td>
<td></td>
<td></td>
<td>DEBEXCPA</td>
<td>X'10'– EXCP(VR) is authorized for this DEB</td>
</tr>
<tr>
<td>.1</td>
<td></td>
<td></td>
<td>DEBFICEV</td>
<td>X'04'–not valid with an EXCP DCB.</td>
</tr>
<tr>
<td>.1</td>
<td></td>
<td></td>
<td>DEBAPFIN</td>
<td>X'02'– if on, authorized programs can be loaded</td>
</tr>
<tr>
<td>.1</td>
<td></td>
<td></td>
<td>DEBXTNIN</td>
<td>X'01'– if one, DEB extension exists</td>
</tr>
<tr>
<td>15(F)</td>
<td>BITSTRING</td>
<td>1</td>
<td>DEBFILGS2</td>
<td>Flag field two</td>
</tr>
<tr>
<td>.1</td>
<td></td>
<td></td>
<td>DEBDSCMP</td>
<td>X'08' tape data set compaction mode</td>
</tr>
<tr>
<td>.1</td>
<td></td>
<td></td>
<td>DEBDSNCP</td>
<td>X'04' tape data set non-compaction mode</td>
</tr>
<tr>
<td>.1</td>
<td></td>
<td></td>
<td>DEB31UCB</td>
<td>UCB address fields are four bytes. Use DEBSUCBA, DEBUCBAD, DEBSDVMX or DEBDVMOD31 instead of DEBSUCBB, DEBUCBA, DEBSDVM or DEBDVMOD.</td>
</tr>
<tr>
<td>16/10</td>
<td>ADDRESS</td>
<td>4</td>
<td>DEBUSRPG(0)</td>
<td>Address of purged I/O restore list (PIRL)</td>
</tr>
<tr>
<td>16/10</td>
<td>BITSTRING</td>
<td>1</td>
<td>DEBNEXT</td>
<td>Number of extent descriptions starting at DEBBASND. One extent per unit for extended format or PDSE data sets</td>
</tr>
<tr>
<td>17/11</td>
<td>ADDRESS</td>
<td>3</td>
<td>DEBUSRPB</td>
<td>Address of purged I/O restore list (PIRL)</td>
</tr>
</tbody>
</table>
Table 97. Partial Listing of DEB Fields (continued)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Type</th>
<th>Length</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>20(14)</td>
<td>ADDRESS</td>
<td>4</td>
<td>DEBRRQ(0)</td>
<td>Pointer to related request queue</td>
</tr>
<tr>
<td>20(14)</td>
<td>BITSTRING</td>
<td>1</td>
<td>DEBPRIOR</td>
<td>Priority of the task owning DEB</td>
</tr>
<tr>
<td>24(18)</td>
<td>ADDRESS</td>
<td>4</td>
<td>DEBDCCBAD(0)</td>
<td>Address of DCB or ACB associated with this DEB</td>
</tr>
<tr>
<td>24(18)</td>
<td>BITSTRING</td>
<td>1</td>
<td>DEBPROTG(0)</td>
<td>Task protection key in high order 4 bits</td>
</tr>
<tr>
<td>24(18)</td>
<td>BITSTRING</td>
<td>1</td>
<td>DEBDEBID</td>
<td>A hex F in low–order 4 bits to identify this block as a DEB</td>
</tr>
<tr>
<td>25(19)</td>
<td>ADDRESS</td>
<td>3</td>
<td>DEBDCCBB</td>
<td>Address of DCB or ACB associated with this DEB</td>
</tr>
<tr>
<td>28(1C)</td>
<td>ADDRESS</td>
<td>4</td>
<td>DEBAPPAD(0)</td>
<td>Address of the I/O appendage vector table</td>
</tr>
<tr>
<td>29(1D)</td>
<td>ADDRESS</td>
<td>3</td>
<td>DEBAPPB</td>
<td>Address of the I/O appendage vector table</td>
</tr>
<tr>
<td>32(20)</td>
<td>DEBBASND</td>
<td></td>
<td></td>
<td>End of basic section</td>
</tr>
<tr>
<td>32(20)</td>
<td>ADDRESS</td>
<td>4</td>
<td>DEBSUCBA(0)</td>
<td>See the DEBUCBAD field</td>
</tr>
<tr>
<td>32(20)</td>
<td>BITSTRING</td>
<td>1</td>
<td>DEBSDVM</td>
<td>Device modifier. For magnetic tape, MODESET operation code. For unit record, not defined. Valid only if DEB31UCB is off.</td>
</tr>
<tr>
<td></td>
<td>1101 0011</td>
<td>1100 0011</td>
<td>DEBMTDN4</td>
<td>X'D3' 9–track MODESET CCW CODE DENSITY=6250BPI</td>
</tr>
<tr>
<td></td>
<td>1100 0011</td>
<td>1100 1011</td>
<td>DEBMTDN2</td>
<td>X'C3' 9–track MODESET CCW CODE DENSITY=1600BPI</td>
</tr>
<tr>
<td></td>
<td>1100 0011</td>
<td></td>
<td>DEBM7DN0</td>
<td>X'03' 7-track MODESET SKELETON DENSITY=200BPI</td>
</tr>
<tr>
<td></td>
<td>0100 0011</td>
<td></td>
<td>DEBM7DN1</td>
<td>X'43' 7-track MODESET SKELETON DENSITY=556BPI</td>
</tr>
<tr>
<td></td>
<td>1000 0011</td>
<td></td>
<td>DEBM7DN2</td>
<td>X'83' 7-track MODESET SKELETON DENSITY=800BPI 3480 tape operation code</td>
</tr>
<tr>
<td></td>
<td>1100 0011</td>
<td></td>
<td>DEBMSTWI</td>
<td>X'C3' 3480 set tape write immediate ccw code. Tape mode set function byte</td>
</tr>
<tr>
<td></td>
<td>1.</td>
<td>.1</td>
<td>DEBM3424</td>
<td>X'02' 3424 mode set flag</td>
</tr>
<tr>
<td></td>
<td>........</td>
<td>....</td>
<td>DEBMCOMP</td>
<td>&quot;DEBMCOMP&quot; COMPACTED RECORDING MODE</td>
</tr>
<tr>
<td></td>
<td>........</td>
<td>....</td>
<td>DEBM6250</td>
<td>&quot;DEBMTRF0+ DEBMTRF1+ DEBM3424&quot; SET 3424 DENSITY=6250BPI</td>
</tr>
<tr>
<td></td>
<td>........</td>
<td>....</td>
<td>DEBM1600</td>
<td>&quot;DEBMTRF1+ DEBM3424&quot; SET 3424 DENSITY=1600BPI</td>
</tr>
<tr>
<td>33(21)</td>
<td>ADDRESS</td>
<td>3</td>
<td>DEBSUCBB</td>
<td>Address of a UCB associated with a given data set</td>
</tr>
<tr>
<td>36(24)</td>
<td>DEBDEVED(0)</td>
<td></td>
<td></td>
<td>End of common tape and unit record fields if DEB31UCB is off.</td>
</tr>
<tr>
<td>36(24)</td>
<td>BITS</td>
<td>1</td>
<td>DEBSDVMX</td>
<td>Device modifier. For magnetic tape, modeset operation code or modeset function byte. For unit record reserved. Present only if DEB31UCB is on.</td>
</tr>
<tr>
<td>37(25)</td>
<td>CHARACTER</td>
<td>3</td>
<td>DEBDVEDX</td>
<td>End of section if DEB31UCB is on.</td>
</tr>
<tr>
<td>40(28)</td>
<td>ADDRESS</td>
<td>4</td>
<td>DEBRDCB(0)</td>
<td>Pointer to device control block</td>
</tr>
<tr>
<td>40(28)</td>
<td>BITSTRING</td>
<td>1</td>
<td>DEBRSV06</td>
<td>Pointer to device control block</td>
</tr>
<tr>
<td>40(28)</td>
<td>ADDRESS</td>
<td>4</td>
<td>DEBPDCB(0)</td>
<td>Address of a data set</td>
</tr>
</tbody>
</table>

Control Blocks

Unit Record, Magnetic Tape, Telecommunications Devices Section

Next four bytes present only if DEB31UCB is on

The following fields are present only for the 3525 with device–associated data set support

---

446  z/OS V2R1.0 DFSMSdfp Advanced Services
### Table 97. Partial Listing of DEB Fields (continued)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Type</th>
<th>Length</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>44(2C)</td>
<td>ADDRESS</td>
<td>4</td>
<td>DEBWDCB(0)</td>
<td>Address of DCB for the read associated data set</td>
</tr>
<tr>
<td>44(2C)</td>
<td>BITSTRING</td>
<td>1</td>
<td>DEBRSV08</td>
<td></td>
</tr>
<tr>
<td>45(2D)</td>
<td>ADDRESS</td>
<td>3</td>
<td>DEBWDCBA</td>
<td>Address of DCBBA</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DEBASDSE</td>
<td>Address of DASD section</td>
</tr>
</tbody>
</table>

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, in which case there is one section per device.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Type</th>
<th>Length</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0(0)</td>
<td>ADDRESS</td>
<td>4</td>
<td>DEBUCBAD</td>
<td>Address of a UCB associated with this extent. The following applies to DEBSUCBA, DEBSUCBB, DEBUCBAD and DEBUCBA: If the actual UCB is above the 16 MB line and the dynamic allocation nocapture option is not in effect, allocation normally captures the UCB to create a 24-bit address. When using EXCP and you specify the nocapture option of dynamic allocation but not the LOC=ANY option on the DCBE, then OPEN or EOV captures the UCB until a later EOV or close. In these cases the high order byte of this word contains the device modifier byte. If you specify nocapture on the dynamic allocation and the actual DASD or tape UCB address is above the line and the DCB is for BSAM, BPAM, QSAM or EXCP and the DCBE has specified LOC=ANY, then the system does not capture the UCB. In that case OPEN turns on the DEB31UCB bit to signify the 31-bit UCB address field is valid and that the device modifier byte is in DEBSDVMX or DEBDVMOD31. It may remain on for subsequent volumes even though they have actual 24-bit addresses in a four-byte field.</td>
</tr>
<tr>
<td>0(0)</td>
<td>BITSTRING</td>
<td>1</td>
<td>DEBDVMOD</td>
<td>Device modifier. File mask. Valid only if DEB31UCB is off.</td>
</tr>
<tr>
<td>1(1)</td>
<td>ADDRESS</td>
<td>3</td>
<td>DEBUCBA</td>
<td>Address of UCB for this extent. Valid only if DEB31UCB is off.</td>
</tr>
<tr>
<td>4(4)</td>
<td>BINARY</td>
<td>1</td>
<td>DEBDVMOD31</td>
<td>Reserved if DEB31UCB is off. File mask (device modifier) if DEB31UCB is on.</td>
</tr>
<tr>
<td>5(5)</td>
<td>BINARY</td>
<td>1</td>
<td>DEBNMTRKHI</td>
<td>High order byte of number of tracks in extent. Low order two bytes are in DEBNMTRK. DEBNMTRKHI combined with DEBNMTRK gives total number of tracks in this extent. DEBNMTRKHI is 0 for a basic format data set and may be non-zero for a large format data set.</td>
</tr>
<tr>
<td>6(6)</td>
<td>BINARY</td>
<td>2</td>
<td>DEBSTRCC</td>
<td>Low order 16 bits of cylinder number of start of extent</td>
</tr>
<tr>
<td>8(8)</td>
<td>BINARY</td>
<td>2</td>
<td>DEBSTRHH</td>
<td>High order 12 bits of cylinder number and four-bit track number of start of extent</td>
</tr>
<tr>
<td>10(A)</td>
<td>BINARY</td>
<td>2</td>
<td>DEBENDCC</td>
<td>Low order 16 bits of cylinder number of end of extent. Not set for a PDSE</td>
</tr>
<tr>
<td>12(C)</td>
<td>BINARY</td>
<td>2</td>
<td>DEBENDHH</td>
<td>High order 12 bits of cylinder number and four-bit track number of end of extent. For PDSE this field is reserved. For an extended format data set this field contains the track number of the format-1 dscb address.</td>
</tr>
<tr>
<td>14(E)</td>
<td>BINARY</td>
<td>2</td>
<td>DEBNMTRK</td>
<td>Number of tracks allocated to a given extent. For a pdse this field is set to one(X'0001'). For an extended format sequential data set the first byte contains the record number of the format-1 dscb address, and the second byte is zero.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DEBDASDE</td>
<td>“*” end of DASD device section</td>
</tr>
</tbody>
</table>

### EXCP Access Method, Bsam And Qsam Dependent Section Length Is In DEBAMLNG.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Type</th>
<th>Length</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0(0)</td>
<td>BINARY</td>
<td>2</td>
<td>DEBVOLSQ</td>
<td>Volume sequence number for multivolume sequential data sets</td>
</tr>
<tr>
<td>0(0)</td>
<td>BITSTRING</td>
<td>1</td>
<td>DEBVOLBT</td>
<td>First byte of debvolsq. Reserved for system use.</td>
</tr>
<tr>
<td>1(1)</td>
<td>SIGNED</td>
<td>1</td>
<td>DEBVSEQ</td>
<td>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.</td>
</tr>
<tr>
<td>2(2)</td>
<td>BINARY</td>
<td>2</td>
<td>DEBVOLNM</td>
<td>Total number of volumes in a multivolume sequential data set.</td>
</tr>
<tr>
<td>4(4)</td>
<td>BINARY</td>
<td>8</td>
<td>DEBSNRM</td>
<td>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.</td>
</tr>
<tr>
<td>4(4)</td>
<td>ADDRESS</td>
<td>4</td>
<td>DEBUTSAA</td>
<td>Address of the user totaling save area.</td>
</tr>
<tr>
<td>4(4)</td>
<td>BITSTRING</td>
<td>1</td>
<td>DEBRSV13</td>
<td>For system use.</td>
</tr>
<tr>
<td>5(5)</td>
<td>ADDRESS</td>
<td>3</td>
<td>DEBUTSAB</td>
<td>Address of the user totaling save area.</td>
</tr>
<tr>
<td>8(8)</td>
<td>BITSTRING</td>
<td>4</td>
<td>DEBRSV14</td>
<td>For system use (if user totaling was specified).</td>
</tr>
<tr>
<td>12(C)</td>
<td>SIGNED</td>
<td>2</td>
<td>DEBBLKSI</td>
<td>Maximum block size.</td>
</tr>
<tr>
<td>14(E)</td>
<td>SIGNED</td>
<td>2</td>
<td>DEBRLRECL</td>
<td>Logical record length.</td>
</tr>
</tbody>
</table>

### BPAM Dependent Section
### Table 97. Partial Listing of DEB Fields (continued)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Type</th>
<th>Length</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0(0)</td>
<td>BINARY</td>
<td>1</td>
<td>DEBEXTNM</td>
<td>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.</td>
</tr>
<tr>
<td>0(0)</td>
<td>CHARACTER</td>
<td>8</td>
<td>DEBDSNAM</td>
<td>For a partitioned data set opened for output for a member name, this field is the member name.</td>
</tr>
</tbody>
</table>

**BDAM Dependent Section For Fixed Length Records With The Option Of Relative Block Addressing**

<table>
<thead>
<tr>
<th>Offset</th>
<th>Type</th>
<th>Length</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0(0)</td>
<td>BINARY</td>
<td>4</td>
<td>DEBDBLK</td>
<td>One four byte field for each extent described in the device dependent section</td>
</tr>
<tr>
<td>0(0)</td>
<td>ADDRESS</td>
<td>1</td>
<td>DEBDBP</td>
<td>Number of blocks per track</td>
</tr>
<tr>
<td>1(1)</td>
<td>BINARY</td>
<td>3</td>
<td>DEBDBP</td>
<td>Number of blocks per extent</td>
</tr>
</tbody>
</table>

**Subroutine Name Section Note Follows The Access Method Dependent Section, Or The Device Dependent Section If There Is No Access Method Section**

<table>
<thead>
<tr>
<th>Offset</th>
<th>Type</th>
<th>Length</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0(0)</td>
<td>CHARACTER</td>
<td>2</td>
<td>DEBSUBID</td>
<td>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.</td>
</tr>
</tbody>
</table>

**DEB Extension Pointed To By DEBXTNP In The DEB Prefix**

<table>
<thead>
<tr>
<th>Offset</th>
<th>Type</th>
<th>Length</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0(0)</td>
<td>SIGNED</td>
<td>2</td>
<td>DEBXLNGH</td>
<td>Length of DEB extension</td>
</tr>
<tr>
<td>2(2)</td>
<td>BITSTRING</td>
<td>1</td>
<td>DEBXLFLG1</td>
<td>Flag byte—for system use</td>
</tr>
<tr>
<td>3(3)</td>
<td>BITSTRING</td>
<td>1</td>
<td>DEBXLFLG2</td>
<td>Flag byte</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DEBBYP</td>
<td>X'40'—when on EXCP scan routine will set iosbyp on and bypass building a prefix</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DEBCHCMP</td>
<td>X'20'—when on EXCP scan routine will set ioschcmp on and bypass building a prefix</td>
</tr>
<tr>
<td>4(4)</td>
<td>ADDRESS</td>
<td>4</td>
<td>DEBXDSAB</td>
<td>Pointer to dsab</td>
</tr>
<tr>
<td>8(8)</td>
<td>BITSTRING</td>
<td>4</td>
<td>DEBXDBPR</td>
<td>Pointer to DEB</td>
</tr>
<tr>
<td>12(C)</td>
<td>ADDRESS</td>
<td>4</td>
<td>DEBXDBPR</td>
<td>Pointer to DEB</td>
</tr>
<tr>
<td>16(10)</td>
<td>CHARACTER</td>
<td>24</td>
<td>DEBDSNED</td>
<td>For system use</td>
</tr>
<tr>
<td>40(28)</td>
<td>BITSTRING</td>
<td>8</td>
<td>DEBXDEF(0)</td>
<td>Define extent data area</td>
</tr>
<tr>
<td>40(28)</td>
<td>BITSTRING</td>
<td>1</td>
<td>DEBDEFG1</td>
<td>Flag byte</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DEBNSHED</td>
<td>X'80' no seek head permitted</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DEBXVDEF</td>
<td>X'40' DEB def ext data parms valid. Must be on for dx data to be used.</td>
</tr>
<tr>
<td>41(29)</td>
<td>BITSTRING</td>
<td>1</td>
<td>DEBGATTR</td>
<td>Global attributes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DEBECKD</td>
<td>X'CO' 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</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DEBGAEX1</td>
<td>X'80' extent definition 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DEBGAEX2</td>
<td>X'40' extent definition 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DEBSTRTP</td>
<td>X'20' data storage type-norm/temp b'1.....' normal data storage b'0.....' temporary data storage</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DEBGA345</td>
<td>X'1c' attributes bits 3,4,5 b'0000.....' normal cache access b'001.....' bypass cache load b'0010.....' inhibit cache load b'0011.....' sequential access b'100.....' sequential staging mode b'101.....' reserved b'111.....' reserved b'.....xx' reserved</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DEBGA1</td>
<td>X'10' attribute 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DEBGA2</td>
<td>X'08' attribute 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DEBGA3</td>
<td>X'04' attribute 3 b'0000.....' normal cache access b'001.....' bypass cache load b'0010.....' inhibit cache load b'0011.....' sequential access b'100.....' sequential staging mode b'101.....' record access mode b'111.....' reserved b'.....xx' reserved</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DEBNCACH</td>
<td>X'00' normal cache access</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DEBCACH</td>
<td>X'04' bypass cache load</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DEBCACH</td>
<td>X'08' inhibit cache load</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DEBCACH</td>
<td>X'0c' sequential access</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DEBNRD</td>
<td>X'02' non-retentive data access</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DEBINFHW</td>
<td>X'01' inhibit fast write b'.....10' use non-retentive data b'.....11' inhibit DASD fast write</td>
</tr>
<tr>
<td>42(2A)</td>
<td>BITSTRING</td>
<td>2</td>
<td>DEBBLKSZ</td>
<td>Blocksize in bytes if less than or equal to 32760</td>
</tr>
</tbody>
</table>
Table 97. Partial Listing of DEB Fields (continued)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Type</th>
<th>Length</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>44(2C)</td>
<td>CHARACTER</td>
<td>4</td>
<td>DEBEXTOK</td>
<td>Zero, extent locator or token</td>
</tr>
<tr>
<td>44(2C)</td>
<td>CHARACTER</td>
<td>2</td>
<td>DEBNRDID</td>
<td>Subsystem function id: non-retentive data id or concurrent copy / xrc session id</td>
</tr>
<tr>
<td>46(2E)</td>
<td>CHARACTER</td>
<td>2</td>
<td>DEBIOPID</td>
<td>For system use</td>
</tr>
<tr>
<td>48(30)</td>
<td>CHARACTER</td>
<td>4</td>
<td>DEBBLKID</td>
<td>Block id value used to calculate number of blocks in this tape volume for extended block count checking</td>
</tr>
</tbody>
</table>

Data Facilities Area (DFA) Fields

Except for DFADFVAD, all of the fields in the DFA are part of the intended programming interface.

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: 112 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.

Table 98. DFA Fields

<table>
<thead>
<tr>
<th>Offset</th>
<th>Type/Value</th>
<th>Length</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0(0)</td>
<td>UNSIGNED</td>
<td>2</td>
<td>DFALEN</td>
<td>LENGTH OF THIS TABLE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>VERSION, RELEASE, AND MODIFICATION LEVEL INFORMATION FOR DFP. THE FIRST THREE DIGITS OF DFAREL REPRESENTS THE LAST LEVEL OF DFP(X’332’). THE FOURTH DIGIT OF DFAREL BEING NON-ZERO INDICATES THE LEVEL OF THIS PRODUCT IS HIGHER THAN THE INDICATED LEVEL.</td>
</tr>
<tr>
<td>2(2)</td>
<td>BITSTRING</td>
<td>2</td>
<td>DFAREL</td>
<td>FOUR DIGITS = VERSION, RELEASE, MOD, X</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>4(4)</td>
<td>SIGNED</td>
<td>4</td>
<td>DFEATS(0)</td>
<td>ALL FEATURES BYTES–USED BY CS INSTR</td>
</tr>
<tr>
<td>4(4)</td>
<td>BITSTRING</td>
<td>1</td>
<td>DFEAT1</td>
<td>FEATURES BYTE 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>‘X’80’ MVS/XA (COPY OF CVTMVSE IN CVT)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>‘X’40’ MULTIPLE VSAM LSR POOLS SUPPORTED (1.1.0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>‘X20’ DASDM ERASE ON SCRATCH SUPPORTED (2.1.0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>‘X’10’ EXTENDED RECOVERY FACILITY (2.1.0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>‘X’08’ EXPORT BY CONTROL INTERVAL (2.1.0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>‘X’04’ ERASE ON SCRATCH FOR ICF (2.1.0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>‘X’02’ SYSTEM MANAGED STORAGE (3.1.0)</td>
</tr>
<tr>
<td>Offset</td>
<td>Type/Value</td>
<td>Length</td>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>------------</td>
<td>--------</td>
<td>----------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DFAPDSE</td>
<td>&quot;X'01'&quot; 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.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DFIIPDS</td>
<td>&quot;DFAPDSE&quot; PDSE IS OLD NAME FOR PDSE</td>
</tr>
<tr>
<td>5(5)</td>
<td>BITSTRING</td>
<td>1</td>
<td>DFAFEAT2</td>
<td>FEATURES BYTE 2</td>
</tr>
<tr>
<td>6(6)</td>
<td>BITSTRING</td>
<td>1</td>
<td>DFAFEAT3</td>
<td>FEATURES BYTE 3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DFAVOLS N</td>
<td>&quot;X'80'&quot; VOLSER EXTRACTED FROM SENSE INFO ACCEPTABLE BY THE INSTALLATION</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DFSAMEX</td>
<td>EXTENDED FORMAT SEQUENTIAL DATA SETS SUPPORTED</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DFASMSEX</td>
<td>ALIAS FOR DFSAMEX</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DFARKDEX</td>
<td>EXTENDED FORMAT KSDS SUPPORTED</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DFARLSJ3</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DFEARCAL</td>
<td>DATA SET RECALL CAPABILITY VIA® THE ARCHRCAL MACRO IS AVAILABLE.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DFADEEEXT</td>
<td>DESERV EXIT FUNCTION IS AVAILABLE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DFADLL</td>
<td>DFSMS DLL SUPPORT IS AVAILABLE</td>
</tr>
<tr>
<td>7(7)</td>
<td>BITSTRING</td>
<td>1</td>
<td>DFAFEAT4</td>
<td>FEATURES BYTE 4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DFAFDAT</td>
<td>RESERVED</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DFANSRV</td>
<td>DFP NIP SERVICES CAN BE INVOKED VIA IGDSRV MACRO</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DFADYNL</td>
<td>DYNAMIC LINKLIST IS SUPPORTED</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DFACIR2</td>
<td>THE CATALOG INFORMATION ROUTINE, IKJEHCIR, SUPPORTS A FORMAT 2 WORK AREA, I.E., FULL WORD LENGTH FIELDS.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DFADYLPA</td>
<td>DFSMS SUPPORT FOR DYNAMIC LPA IS AVAILABLE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DFAFORK</td>
<td>DFSMS LOADER FORK EXIT IS PRESENT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DFASNKB</td>
<td>SOFTWARE SUPPORT PROVIDING &quot;FAST&quot; BACKUP USING THE SNAPSHOT FEATURE OF THE RAMAC VIRTUAL ARRAY (RVA) IS INSTALLED.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DFEASNAP</td>
<td>THE API SUPPORT FOR THE SNAPSHOT FEATURE OF THE RAMAC VIRTUAL ARRAY (RVA) IS INSTALLED.</td>
</tr>
<tr>
<td>8(8)</td>
<td>CHARACTER</td>
<td>4</td>
<td>DFAACRON</td>
<td>ACRONYM FOR THIS CONTROL BLOCK</td>
</tr>
<tr>
<td>12(C)</td>
<td>BITSTRING</td>
<td>1</td>
<td>DFAFEAT5</td>
<td>FEATURES BYTE 5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DFAIPDS</td>
<td>UNMANAGED PDSE SUPPORT INSTALLED ON THIS SYSTEM.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DFBTSREQ</td>
<td>1 means BLOCKTOKENSIZE=REQUIRE in IGDSMSxx member of PARMLIB. Restrictions on opening large format data sets. 0 means BLOCKTOKENSIZE=REQUIRE.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DFABLDLS</td>
<td>BLDS START= and STOP= parameters are supported.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DFAUSEAV</td>
<td>System default USEEAV setting for an extended address volume (EAV) when SMS is not active. Initially set on, to allow the use of EAV, and changed to the IGDSMSxx PARMLIB specified or defaulted USEEAV value if SMS is active.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DFEASAMHPF</td>
<td>SAM_USE_HPFP, On = yes</td>
</tr>
</tbody>
</table>
## Table 98. DFA Fields (continued)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Type/Value</th>
<th>Length</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.......</td>
<td>....1</td>
<td>1</td>
<td>DFAALFOR</td>
<td>Determines whether installation ISO/ANSI version level is forced</td>
</tr>
<tr>
<td>13(D)</td>
<td>BITSTRING</td>
<td>1</td>
<td>DFAFEAT6</td>
<td>Features byte 6</td>
</tr>
<tr>
<td>1111...</td>
<td></td>
<td>1</td>
<td>DFACPSDB</td>
<td>COPYSDB value in DEVSUPxx in PARMLIB. System level default for the SDB option of IEBGENER and other copying programs. VAE constants below DFACPS**</td>
</tr>
<tr>
<td>0001...</td>
<td></td>
<td>1</td>
<td>DFACPSNO</td>
<td>COPYSDB = NO</td>
</tr>
<tr>
<td>0010...</td>
<td></td>
<td>1</td>
<td>DFACPSYE</td>
<td>COPYSDB = YES</td>
</tr>
<tr>
<td>0010...</td>
<td></td>
<td>1</td>
<td>DFACPSMSM</td>
<td>COPYSDB = SMALL (SAME AS COPYSDB = YES)</td>
</tr>
<tr>
<td>0011...</td>
<td></td>
<td>1</td>
<td>DFACPSIN</td>
<td>COPYSDB = INPUT</td>
</tr>
<tr>
<td>0100...</td>
<td></td>
<td>1</td>
<td>DFACPSLA</td>
<td>COPYSDB = LARGE</td>
</tr>
<tr>
<td>.......</td>
<td>1</td>
<td></td>
<td>DFADCMET</td>
<td>Data class media enforced for all tapes outside of libraries (ENFORCE_DC_MEDIA=ALMEDIATY)</td>
</tr>
<tr>
<td>.......</td>
<td>.1...</td>
<td>1</td>
<td>DFADCMIEJ</td>
<td>Data class media enforced for IBM 3592 outside of libraries (ENFORCE_DC_MEDIA=MEDIA5PLUS)</td>
</tr>
<tr>
<td>.......</td>
<td>..1.</td>
<td>1</td>
<td>DFAMTLAM</td>
<td>When no media preference is expressed, accept all media types for a manual tape library (MTL_NO_DC_WORM_OK)</td>
</tr>
<tr>
<td>.......</td>
<td>...1</td>
<td>1</td>
<td>FASTIFF</td>
<td>STOW supports the IFF operand</td>
</tr>
<tr>
<td>14(E)</td>
<td>UNSIGNED</td>
<td>2</td>
<td>DFABPV</td>
<td>System default break point value (BPV) for an extended address volume (EAV) when the BPV is not specified in the SMS IGDSMxx PARMLIB or when SMS is not active. Initially set to 10 and changed to the IGDSMxx SPECIFIED BPV value if SMS is active.</td>
</tr>
</tbody>
</table>

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. A PRODUCT BYTE (DFAPROD) OF X'00' INDICATES DFP AS A PRODUCT. BYTES 1-3 OF DFARELS WILL ALSO BE X'00' IN THIS CASE. THE USER MAY CHOOSE TO CHECK DFAREL FOR THE RELEASE LEVEL OF THE DFP PRODUCT IN THIS CASE. IF DFAPROD IS NOT EQUAL TO X'00', DFAREL SHOULD NOT BE CHECKED AS IT WILL BE FROZEN AT THE LAST LEVEL OF DFP PRODUCT SHIPPED. A PRODUCT BYTE (DFAPROD) OF X'01' INDICATES DFSMS AS A PRODUCT. BYTES 1-3 OF DFARELS WILL INDICATE THE VERSION, RELEASE AND MODIFICATION LEVELS OF THE DFSMS PRODUCT. A PRODUCT BYTE (DFAPROD) OF X'02' INDICATES OS/390. THIS VALUE INDICATES THIS LEVEL OF DFSMS IS OS/390 EXCLUSIVE. SINCE DFSMS MIGHT NOT BE REFRESHED WITH EACH OS/390 RELEASE, THE VERSION, RELEASE, AND MODIFICATION FIELDS INDICATE THE LEVEL OF OS/390 IN WHICH THIS LEVEL OF DFSMS WAS FIRST SHIPPED. THE VERSION, RELEASE, AND MODIFICATION FIELDS ARE BINARY VALUES. (FOR EXAMPLE TEN WOULD BE X'0A').

| 16(10) | BITSTRING | 4      | DFARELS              | 4 BYTES = PRODUCT, VERSION, REL, MOD                                        |
| 16(10) | BITSTRING | 1      | DFAPROD              | PRODUCT BYTE                                                                |
| 0000 0000 |            |       | DFADFDP              | DFP PRODUCT CODE.                                                           |
| 0000 0001 |            |       | DFADFSSMS            | PRODUCT CODE FOR DFSMS.                                                     |
| 0000 0002 |            |       | DFADSS90             | PRODUCT CODE FOR OS/390 VERSION 2.                                           |
| 0000 0003 |            |       | DFADZOS              | PRODUCT CODE FOR z/OS.                                                      |
| 17(11) | BITSTRING | 1      | DFAYER               | VERSION BYTE                                                                |
| 18(12) | BITSTRING | 1      | DFARLSE              | RELEASE BYTE                                                                |
| 19(13) | BITSTRING | 1      | DFAMOD               | MODIFICATION BYTE                                                          |
| 20(14) | SIGNED    | 2      | DFAMSMDDE            | MAXIMUM LENGTH OF THE SMDE IN THIS RELEASE WITH AN 8-BYTE ALIAS NAME         |
| 22(16) | UNSIGNED  | 1      | DFAVERBO             | FLAGS                                                                       |
| ....... | ....       |       | *                    | Reserved.                                                                   |
| ....... | .1...      | 1      | DFATVS               | IF ON, TRANSACTIONAL VSAM IS INSTALLED.                                    |
| ....... | ..1...     | 1      | DFACFXHS             | FlashCopy® across Hyperswap enabled                                         |
| 23(17) | UNSIGNED  | 1      | DFASEFVR             | Data set format version for new sequential extended format data sets. This is set by the PS_EXT_VERSION keyword in the IGDSMxx member of SYSL.PARMLIB. |
| 24(18) | ADDRESS   | 4      | DFACSXVT             | CALLABLE SYSTEM SERVICES VECTOR TABLE ADDRESS                                |
| 28(1C) | ADDRESS   | 4      | DFADCXSO             | DATA CONVERSION SERVICES - OPEN.                                             |
| 32(20) | ADDRESS   | 4      | DFADCXSID            | DATA CONVERSION SERVICES - CONVERT.                                         |
| 36(24) | ADDRESS   | 4      | DFADCXSC             | DATA CONVERSION SERVICES - CLOSE.                                           |
| 40(28) | ADDRESS   | 4      | DFAEELNM             | Address of DFSMS element name. Name is mapped by DSECT DFAEELNM. See Table 99 on page 452 Valid only on z/OS 1.3 and later. |
| 44(2C) | ADDRESS   | 4      | DFADFVAD             | DATA FACILITIES VECTOR TABLE ADDR                                           |

END OF DFA AS IT WAS WHEN IT WAS FIRST SHIPPED IN MVS/XA DFP VERSION 2 RELEASE 1 MODIFICATION LEVEL 0. PRIOR TO REFERENCING ANY FIELD BEYOND THIS COMMENT, THE USER MUST ENSURE THAT DFARELS IS GREATER THAN OR EQUAL TO X'00200A00' OR DFALEN IS BIG ENOUGH (SEE DFALEN).

| 48(30) | INTEGER   | 8      | DFABLKSZ             | LIMIT ON SYSTEM DETERMINED BLOCK SIZE. DEFAULT IS 32760. OBTAINED FROM DEVSUPxx PARMLIB MEMBER. |

---

**Control Blocks**

Appendix A. Control Blocks 451
Table 98. DFA Fields (continued)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Type/Value</th>
<th>Length</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>56(38)</td>
<td>BITSTRING</td>
<td>1</td>
<td>DFAFEAT7</td>
<td>FEATURES BYTE 7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DFATADSN</td>
<td>TAPEAUTHDSN=YES</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DFATADSI</td>
<td>TAPEAUTHFI=YES</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DFATARC8</td>
<td>TAPEAUTHRC8=WARN</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DFATARC4</td>
<td>TAPEAUTHRC4=FAIL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DFAXTBAM</td>
<td>THE NON_VSAM_XTIOT OPTION OF THE DEVSUPxx MEMBER OF PARMLIB HAS BEEN SET TO 'YES'.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DFATPMVA</td>
<td>TAPEMULTIVOLUMEERROR=ALLOW</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DFATPMVF</td>
<td>TAPEMULTIVOLUMEERROR=FAIL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>57(39)</td>
<td>BITSTRING</td>
<td>1</td>
<td>DFAFEAT7</td>
<td>FEATURES BYTE 8. The following 7 flags indicate VSAM and non-VSAM data set support for the Extended Addressing Space (EAS) on an EAV</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DFAVSAMFOREAS</td>
<td>VSAM enabled for EAS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DFASEQFOREAS</td>
<td>Basic, large format sequential (QSAM, BSAM, BDAM access) enabled for EAS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DFAPDSEFOREAS</td>
<td>PDSE enabled for EAS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DFAPDSEFOREAS</td>
<td>PDS enabled for EAS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DFADIRFOREAS</td>
<td>Direct (BDAM access) enabled for EAS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DFAEFSEQFOREAS</td>
<td>Extended format sequential enabled for EAS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DFANDNEFFOREAS</td>
<td>Undef DSORGs enabled for EAS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DFEXPMSG</td>
<td>EXPIRATION_MESSAGE=NEVER</td>
</tr>
<tr>
<td>58(3A)</td>
<td>Unsigned</td>
<td>6</td>
<td>DFAADDRSZ</td>
<td>Storage size limit allowed in DDR swap (number of megabytes)</td>
</tr>
<tr>
<td>58(3A)</td>
<td>Unsigned</td>
<td>2</td>
<td>DFAJ3AA</td>
<td>JES3_ALLOC_ASSIST ENABLED</td>
</tr>
<tr>
<td>58(3A)</td>
<td>Unsigned</td>
<td>2</td>
<td>DFAJ3AASN</td>
<td>MEMBER SELECTION USER EXIT SUPPORT</td>
</tr>
<tr>
<td>60(3C)</td>
<td>Bit String</td>
<td>1</td>
<td>DFAFEAT9</td>
<td>FEATURES BYTE 9</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DFAMEMUX</td>
<td>MEMBER SELECTION USER EXIT SUPPORT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DFAPDSEG</td>
<td>PDSE Generation support is installed</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DFAZEDCCMP</td>
<td>zEDC Compression support is installed</td>
</tr>
<tr>
<td>61(3D)</td>
<td></td>
<td>3</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>64(40)</td>
<td>Unsigned</td>
<td>8</td>
<td>DFA47S</td>
<td>PRSDx47 counts (Use for QUERYFC)</td>
</tr>
<tr>
<td>68(44)</td>
<td>Signed</td>
<td>4</td>
<td>DFA47MAX</td>
<td>Maximum PRSD47 so far</td>
</tr>
<tr>
<td>72(48)</td>
<td>Bit string</td>
<td>4</td>
<td>DFAFEATC</td>
<td>HPF/FCX Feature Code bytes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DFAMAXGN</td>
<td>Maximum Generations supported for this system</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DFACMPTYPE</td>
<td>Default compression type.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DFAMAXGN</td>
<td>Maximum Generations supported for this system</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DFACMPTYPE</td>
<td>Default compression type.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DFAMAXGN</td>
<td>Maximum Generations supported for this system</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DFAMAXGN</td>
<td>Maximum Generations supported for this system</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DFAMAXGN</td>
<td>Maximum Generations supported for this system</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DFAMAXGN</td>
<td>Maximum Generations supported for this system</td>
</tr>
<tr>
<td>81(5D)</td>
<td></td>
<td>31</td>
<td>Reserved</td>
<td></td>
</tr>
</tbody>
</table>

Table 99. DFA Element Name

<table>
<thead>
<tr>
<th>Offset</th>
<th>Type/Value</th>
<th>Length</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0(0)</td>
<td>DSECT</td>
<td>2</td>
<td>DFAELNM</td>
<td>DSECT name</td>
</tr>
</tbody>
</table>
| 0(0)   | UNSIGNED   | 2      | DFAELNML| Length of significant characters in next field. Currently ten.
<table>
<thead>
<tr>
<th>Offset</th>
<th>Type/Value</th>
<th>Length</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2(2)</td>
<td>CHARACTER</td>
<td>22</td>
<td>DFAEXTXT</td>
<td>DFSMS element name in EBCDIC. May contain lower case characters. Currently is &quot;z/OS DFSMS&quot;.</td>
</tr>
</tbody>
</table>
Control Blocks
Appendix B. Maintaining the System Image Library

This information 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 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:

<table>
<thead>
<tr>
<th>Printer</th>
<th>Images</th>
</tr>
</thead>
<tbody>
<tr>
<td>1403 or 3203</td>
<td>AN, HN, PCAN, PCHN, PN, QN, QNC, RN, SN, TN, XN, YN</td>
</tr>
<tr>
<td>3211</td>
<td>A11, G11, H11, P11, T11</td>
</tr>
</tbody>
</table>

For detailed, printer-specific information, or to determine which print bands/trains are available, see the publications listed in the following table:

<table>
<thead>
<tr>
<th>Publication Title</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM 2821 Control Unit Component Description</td>
<td>Information on creating a user-designed chain/train for the 1403 Printer.</td>
</tr>
<tr>
<td>IBM 3203 Printer Component Description and Operator's Guide</td>
<td>Information on creating a user-designed train for the 3203 Printer.</td>
</tr>
<tr>
<td>IBM 3211 Printer, 3216 Interchangeable Train Cartridge, and 3811 Printer Control Unit Component Description and Operator’s Guide</td>
<td>Information on creating a user-designed train for the 3211 Printer.</td>
</tr>
<tr>
<td>IBM 3262 Model 5 Printer Product Description</td>
<td>Information on band IDs for the 3262 Model 5 Printer.</td>
</tr>
<tr>
<td>IBM 3800 Printing Subsystem Programmer’s Guide</td>
<td>Information on planning for, conversion to, and use of the IBM 3800 Model 1 Printing Subsystem.</td>
</tr>
<tr>
<td>IBM 3800 Printing Subsystem Model 3 Programmer’s Guide: Compatibility</td>
<td>Information on planning for, conversion to, and use of the IBM 3800 Model 3, 6, and 8 Printing Subsystem.</td>
</tr>
<tr>
<td>IBM 4245 Printer Model 1 Component Description and Operator’s Guide</td>
<td>Information on band IDs for the 4245 Printer.</td>
</tr>
<tr>
<td>IBM 4248 Printer Model 1 Description</td>
<td>Information on band IDs for the 4248 Printer.</td>
</tr>
<tr>
<td>IBM 6262 Printer Print Band Manual</td>
<td>Information on band IDs for the 6262 Printer.</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>Publication Title</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>z/OS DFSMS Macro Instructions for Data Sets</td>
<td>Describes the SETPRT macro. You can use it to specify the images (modules) that you want.</td>
</tr>
<tr>
<td>z/OS DFSMSdfp Utilities</td>
<td>Describes the IEBIMAGE utility program.</td>
</tr>
<tr>
<td>z/OS MVS JCL Reference</td>
<td>Describes the CHARS, MODIFY, UCS, and FCB parameters of the DD statement, that are processed at OPEN.</td>
</tr>
<tr>
<td>z/OS JES2 Initialization and Tuning Guide</td>
<td>Contains JES2 reference information.</td>
</tr>
<tr>
<td>z/OS JES3 Initialization and Tuning Guide</td>
<td>Contains JES3 reference information.</td>
</tr>
</tbody>
</table>

Table 100 provides information about the printer-specific contents of SYS1.IMAGELIB.

Table 100. SYS1.IMAGELIB Contents

<table>
<thead>
<tr>
<th>Printer Type</th>
<th>UCS Image</th>
<th>UCS Image Table</th>
<th>FCB Image</th>
<th>Control Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>1403</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3203 1</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3211</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3262 Model 5</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4245 3</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4248 3</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6262 Model 14</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3800</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

Notes to Table 100
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 Table 101 on page 463. The contents of the 4248 image table are shown in Table 102 on page 464.

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 z/OS DFSMSdfp 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 z/O uses other types of modules from other libraries. See 3900 Product Description for more information.

8. Figure 54 on page 463 defines and describes the structure of a UCS image table entry.

UCS Images in SYS1.IMAGELIB

This information 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:

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>UCS1</td>
<td>1403 printer</td>
</tr>
<tr>
<td>UCS2</td>
<td>3211 printer (or 3211-compatible printer)</td>
</tr>
<tr>
<td>UCS3</td>
<td>3203 printer</td>
</tr>
</tbody>
</table>

   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 z/OS MVS Program Management: User’s Guide and Reference)

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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
</table>
   | X'00' | Indicates that the image is not to be used as a default.
Maintaining IMAGELIB

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'40'</td>
<td>Indicates that the output is to be folded.</td>
</tr>
<tr>
<td>X'80'</td>
<td>Indicates a default image.</td>
</tr>
<tr>
<td>X'C0'</td>
<td>Indicates default image and folding.</td>
</tr>
</tbody>
</table>

For non-JES2, specify:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'00'</td>
<td>Indicates that the image is not to be used as a default.</td>
</tr>
<tr>
<td>X'80'</td>
<td>Indicates a default image.</td>
</tr>
</tbody>
</table>

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 470 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.

<table>
<thead>
<tr>
<th>Printer</th>
<th>Image Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>1403</td>
<td>240 bytes</td>
</tr>
<tr>
<td>3203</td>
<td>304 bytes (240 characters followed by 64 bytes of associative bits)</td>
</tr>
<tr>
<td>3211</td>
<td>512 bytes (432 characters followed by 15 bytes of X'00', 64 bytes of associative bits, and 1 reserved byte of X'00')</td>
</tr>
</tbody>
</table>

You must code associative bits to prevent data checks when adding a UCS image to SYS1.IMAGELIB. See the appropriate printer publication for more information on coding associative bits.

See the following information:

- “Examples of UCS Image Coding”
- “UCS Image Alias Names” on page 462
- “UCS Image Tables in SYS1.IMAGELIB” on page 462
- “Alias Names in UCS Image Tables” on page 462
- “Adding or Modifying a UCS Image Table Entry” on page 466
- “Verifying the UCS Image” on page 470

Examples of UCS Image Coding

Figure 51 on page 459 shows an example of the JCL to add a 1403 UCS image, YN, to SYS1.IMAGELIB. Notes, which apply to all examples, follow Figure 53 on page 461.
Maintaining IMAGELIB

//ADDYN JOB MSGLEVEL=1
//STEP EXEC PROC=ASMHCL,PARM.ASM='NODECK,LOAD',
// PARM.LKED='LIST,OL,REFR,RENT,XREF'
//ASM.SYSIN DD *
UCSIYN 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(39) (39 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'1234567890STABCDEFGHIJKLMNOPQRSTUVWXYZ*,.#-$'
  DC C'1234567890STABCDEFGHIJKLMNOPQRUVWXYZ*,.
  DC C'1234567890STABCDEFGHIJKLMNOPQRSTUVWXYZ*,.#-$'
  DC C'1234567890STABCDEFGHIJKLMNOPQRUVWXYZ*,.
  DC C'1234567890STABCDEFGHIJKLMNOPQRSTUVWXYZ*,.'
END
/*
//LKED.SYSLMOD DD DSNAME=SYS1.IMAGELIB(UCSIYN),DISP=OLD,
// SPACE= (OVERRIDE SECONDARY ALLOCATION)

Figure 51. Code to Add a 1403 UCS Image to SYS1.IMAGELIB

See Figure 53 on page 461 for the notes to this figure.

Figure 52 on page 460 shows an example of the JCL to add a 3203 UCS image, YN, to SYS1.IMAGELIB. Notes to this figure follow Figure 53 on page 461.
Maintaining IMAGELIB

Figure 52. Code to Add a 3203 UCS Image to SYS1.IMAGELIB

Notes: See Figure 53 on page 461 for the notes to this figure.

Figure 53 on page 461 shows an example of the JCL to add a 3211 UCS image, A11, to SYS1.IMAGELIB.

// 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'1234567890STABCDEFGHIJKLMNOPQRSTUVWXYZ*.,'  
DC C'1234567890STABCDEFGHIJKLMNOPQRSTUVWXYZ*.,'  
DC C'1234567890STABCDEFGHIJKLMNOPQRSTUVWXYZ*.,'  
DC C'1234567890STABCDEFGHIJKLMNOPQRSTUVWXYZ*.,'  
DC C'1234567890STABCDEFGHIJKLMNOPQRSTUVWXYZ*.,'  
DC C'1234567890STABCDEFGHIJKLMNOPQRSTUVWXYZ*.,'  
* THE FOLLOWING FOUR DC INSTRUCTIONS DEFINE THE ASSOCIATIVE BITS,
* UCSB BYTE POSITIONS 241-304
DC X'C01010101010101010000000000000010'
DC X'1010101010101000040000000000000010'
DC X'1010101010100000000000000000000010'
DC X'101010100000000000000000000000000000010'
END
/*
// LKED.SYSLMOD DD DSNAME=SYS1.IMAGELIB(UCS3YN),DISP=OLD,
// SPACE= (OVERRIDE SECONDARY ALLOCATION)
In the sample code in the Figure 51 on page 459, Figure 52 on page 460, and Figure 53 be aware of the following:

- The RENT linkage editor attribute is required.

- 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 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.

- Executing the ASMHCL procedure does not generate executable code. The assembler/linkage editor merely places the UCS image into SYS1.IMAGELIB.

- 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:

<table>
<thead>
<tr>
<th>Image</th>
<th>Alias</th>
</tr>
</thead>
<tbody>
<tr>
<td>UCS1AN</td>
<td>UCS1A11</td>
</tr>
<tr>
<td>UCS1HN</td>
<td>UCS1H11</td>
</tr>
<tr>
<td>UCS1PN</td>
<td>UCS1P11</td>
</tr>
<tr>
<td>UCS1TN</td>
<td>UCS1T11</td>
</tr>
<tr>
<td>UCS2A11</td>
<td>UCS2AN</td>
</tr>
<tr>
<td>UCS2H11</td>
<td>UCS2HN</td>
</tr>
<tr>
<td>UCS2P11</td>
<td>UCS2PN, UCS2RN, UCS2QN</td>
</tr>
<tr>
<td>UCS2T11</td>
<td>UCS2TN</td>
</tr>
</tbody>
</table>

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 z/OS MVS Program Management: User's Guide and Reference.) 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, 4248, and 6262 Model 14 printers. SYS1.IMAGELIB does not contain UCS images for these printers, but, instead, contains image tables. 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. 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 54 on page 463 for the format of image table entries, and “Adding or Modifying a UCS Image Table Entry” on page 466 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.

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 466. A typical UCS image table entry is shown in Figure 54.

Figure 54. UCS Image Table Entry Format

Notes to Figure 54
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 Table 101.

Table 101. UCS5 Image Table Contents

<table>
<thead>
<tr>
<th>Name</th>
<th>Alias</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AN21</td>
<td>AN21</td>
<td>YES</td>
<td>Default UCS image</td>
</tr>
<tr>
<td>AN21</td>
<td>AN</td>
<td>NO</td>
<td>1403/3203 AN image</td>
</tr>
<tr>
<td>AN21</td>
<td>A11</td>
<td>NO</td>
<td>3211 A11 image</td>
</tr>
<tr>
<td>AN21</td>
<td>40E1</td>
<td>NO</td>
<td>4248 40E1 image</td>
</tr>
<tr>
<td>HN21</td>
<td>HN21</td>
<td>NO</td>
<td>Nondefault UCS image</td>
</tr>
</tbody>
</table>
Maintaining IMAGELIB

Table 101. UCS5 Image Table Contents (continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>Alias</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HN21</td>
<td>HN</td>
<td>NO</td>
<td>1403/3203 HN image</td>
</tr>
<tr>
<td>HN21</td>
<td>H11</td>
<td>NO</td>
<td>3211 H11 image</td>
</tr>
<tr>
<td>HN21</td>
<td>4101</td>
<td>NO</td>
<td>4248 4101 image</td>
</tr>
<tr>
<td>PL21</td>
<td>PL21</td>
<td>NO</td>
<td>Nondefault UCS image</td>
</tr>
<tr>
<td>PL21</td>
<td>PN</td>
<td>NO</td>
<td>1403/3203 PN image</td>
</tr>
<tr>
<td>PL21</td>
<td>P11</td>
<td>NO</td>
<td>3211 P11 image</td>
</tr>
<tr>
<td>PL21</td>
<td>4121</td>
<td>NO</td>
<td>4248 4121 image</td>
</tr>
<tr>
<td>SN21</td>
<td>SN21</td>
<td>NO</td>
<td>Nondefault UCS image</td>
</tr>
<tr>
<td>SN21</td>
<td>4201</td>
<td>NO</td>
<td>4248 4201 image</td>
</tr>
<tr>
<td>TN21</td>
<td>TN21</td>
<td>NO</td>
<td>Nondefault UCS image</td>
</tr>
<tr>
<td>TN21</td>
<td>TN</td>
<td>NO</td>
<td>1403/3203 TN image</td>
</tr>
<tr>
<td>TN21</td>
<td>T11</td>
<td>NO</td>
<td>3211 T11 image</td>
</tr>
<tr>
<td>TN21</td>
<td>4181</td>
<td>NO</td>
<td>4248 4181 image</td>
</tr>
<tr>
<td>GN21</td>
<td>GN21</td>
<td>NO</td>
<td>Nondefault UCS image</td>
</tr>
<tr>
<td>GN21</td>
<td>G11</td>
<td>NO</td>
<td>3211 G11 image</td>
</tr>
<tr>
<td>GN21</td>
<td>41C1</td>
<td>NO</td>
<td>4248 41C1 image</td>
</tr>
<tr>
<td>RN21</td>
<td>RN21</td>
<td>NO</td>
<td>Nondefault UCS image</td>
</tr>
<tr>
<td>RN21</td>
<td>RN</td>
<td>NO</td>
<td>1403/3203 RN image</td>
</tr>
<tr>
<td>KA21</td>
<td>KA21</td>
<td>NO</td>
<td>Nondefault UCS image</td>
</tr>
<tr>
<td>KA21</td>
<td>4041</td>
<td>NO</td>
<td>4248 4041 image</td>
</tr>
<tr>
<td>KA22</td>
<td>CA22</td>
<td>NO</td>
<td>Nondefault UCS image</td>
</tr>
<tr>
<td>FC21</td>
<td>FC21</td>
<td>NO</td>
<td>Nondefault UCS image</td>
</tr>
<tr>
<td>FC21</td>
<td>4161</td>
<td>NO</td>
<td>4248 4161 image</td>
</tr>
</tbody>
</table>

The contents of the UCS image table UCS6 (IGGUCS6 macro), for the 4248 printer, are shown in Table 102.

Table 102. UCS6 Image Table Contents

<table>
<thead>
<tr>
<th>Name</th>
<th>Alias</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>40E1</td>
<td>40E1</td>
<td>YES</td>
<td>Default UCS image</td>
</tr>
<tr>
<td>40E1</td>
<td>AN21</td>
<td>NO</td>
<td>4245 AN21 image</td>
</tr>
<tr>
<td>40E1</td>
<td>AN</td>
<td>NO</td>
<td>1403/3203 AN image</td>
</tr>
<tr>
<td>40E1</td>
<td>A11</td>
<td>NO</td>
<td>3211 A11 image</td>
</tr>
<tr>
<td>4101</td>
<td>4101</td>
<td>NO</td>
<td>Nondefault UCS image</td>
</tr>
<tr>
<td>4101</td>
<td>HN21</td>
<td>NO</td>
<td>4245 HN21 image</td>
</tr>
<tr>
<td>4101</td>
<td>HN</td>
<td>NO</td>
<td>1403/3203 HN image</td>
</tr>
<tr>
<td>4101</td>
<td>H11</td>
<td>NO</td>
<td>3211 H11 image</td>
</tr>
<tr>
<td>41C1</td>
<td>41C1</td>
<td>NO</td>
<td>Nondefault UCS image</td>
</tr>
<tr>
<td>41C1</td>
<td>GN21</td>
<td>NO</td>
<td>4245 GN21 image</td>
</tr>
<tr>
<td>41C1</td>
<td>G11</td>
<td>NO</td>
<td>3211 G11 image</td>
</tr>
<tr>
<td>4121</td>
<td>4121</td>
<td>NO</td>
<td>Nondefault UCS image</td>
</tr>
</tbody>
</table>
Tip: 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 Table 103 and “Adding or Modifying a UCS Image Table Entry” on page 466.

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 466 describes how to add entries to the UCS image table. For a list of the bands available for the 3262 Model 5, see Table 103. For a list of the bands available for the 6262 Model 14, see IBM 6262 Printer Print Band Manual and IBM 6262 Printer Model 014 User’s Guide.

For Table 103 be aware of the following:

- You can define any 1- to 4-character name as aliases of the UCS6 Band names.
- xx designates switch settings that are ignored. Switch number 3 must be on if you are using a special order (RPQ) band.
- Note that bb here represents a space.

Table 103. 3262 Model 5 Print Bands

<table>
<thead>
<tr>
<th>Character Set Name</th>
<th>UCS6 Band Name</th>
<th>3262 Model 5 Band Image Select Switch Settings (positions 1-8)</th>
</tr>
</thead>
<tbody>
<tr>
<td>U.S./International</td>
<td></td>
<td></td>
</tr>
<tr>
<td>48 char EBCDIC</td>
<td>00bb</td>
<td>xx00 0000</td>
</tr>
<tr>
<td>63 char EBCDIC</td>
<td>01bb</td>
<td>xx00 0001</td>
</tr>
<tr>
<td>64 char EBCDIC</td>
<td>02bb</td>
<td>xx00 0010</td>
</tr>
<tr>
<td>96 char EBCDIC</td>
<td>03bb</td>
<td>xx00 0011</td>
</tr>
<tr>
<td>48 char AON OCR</td>
<td>04bb</td>
<td>xx00 0100</td>
</tr>
<tr>
<td>48 char BON OCR</td>
<td>05bb</td>
<td>xx00 0101</td>
</tr>
</tbody>
</table>
### 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 55 on page 469](#) and [Figure 56 on page 470](#).

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.

   **Requirement:** The RENT linkage editor attribute is required.

The format of the IGGUCSIT macro is:
**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 54 on page 463](#). 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.

**Exception:** 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 470.

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 470.

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 55 on page 469, 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.
When adding a new band ID to the 4245 UCS image table as shown in Figure 55, be aware of the following:

- The RENT linkage editor attribute is required.
- Executing the ASMHCL procedure does not generate executable code. The assembler/linkage editor places the updated UCS image table into SYS1.IMAGELIB.
- 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 56 on page 470, 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.
For Figure 56, be aware of the following:

- 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.
- The RENT linkage editor attributes is required.
- Executing the ASMHCL procedure does not generate executable code. The assembler/linkage editor places the updated UCS image table into SYS1.IMAGELIB.
- 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 z/OS MVS JCL Reference and z/OS DFSMS 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. z/OS DFSMSdfp 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 57 on page 472 and Figure 58 on page 473 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.
<table>
<thead>
<tr>
<th>Line</th>
<th>Channel</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>X'01'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td>X'02'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
<tr>
<td>X'03'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td>X'04'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
</tr>
<tr>
<td>X'05'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6</td>
</tr>
<tr>
<td>X'06'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7</td>
</tr>
<tr>
<td>X'07'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>8</td>
</tr>
<tr>
<td>X'08'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>9</td>
</tr>
<tr>
<td>X'09'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>10</td>
</tr>
<tr>
<td>X'0A'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>11</td>
</tr>
<tr>
<td>X'0B'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>12</td>
</tr>
<tr>
<td>X'0C'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>13</td>
</tr>
<tr>
<td>X'0D'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>14</td>
</tr>
<tr>
<td>X'0E'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>15</td>
</tr>
<tr>
<td>X'0F'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>16</td>
</tr>
<tr>
<td>X'10'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>17</td>
</tr>
<tr>
<td>X'11'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>18</td>
</tr>
<tr>
<td>X'12'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>19</td>
</tr>
<tr>
<td>X'13'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>20</td>
</tr>
<tr>
<td>X'14'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>21</td>
</tr>
<tr>
<td>X'15'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>22</td>
</tr>
<tr>
<td>X'16'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>23</td>
</tr>
<tr>
<td>X'17'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>24</td>
</tr>
<tr>
<td>X'18'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>25</td>
</tr>
<tr>
<td>X'19'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>26</td>
</tr>
<tr>
<td>X'1A'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>27</td>
</tr>
<tr>
<td>X'1B'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>28</td>
</tr>
<tr>
<td>X'1C'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>29</td>
</tr>
<tr>
<td>X'1D'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>30</td>
</tr>
<tr>
<td>X'1E'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>31</td>
</tr>
<tr>
<td>X'1F'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>32</td>
</tr>
<tr>
<td>X'20'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>33</td>
</tr>
<tr>
<td>X'21'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>34</td>
</tr>
<tr>
<td>X'22'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>35</td>
</tr>
<tr>
<td>X'23'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>36</td>
</tr>
<tr>
<td>X'24'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>37</td>
</tr>
<tr>
<td>X'25'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>38</td>
</tr>
<tr>
<td>X'26'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>39</td>
</tr>
<tr>
<td>X'27'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>40</td>
</tr>
<tr>
<td>X'28'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>41</td>
</tr>
<tr>
<td>X'29'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>42</td>
</tr>
<tr>
<td>X'2A'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>43</td>
</tr>
<tr>
<td>X'2B'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>44</td>
</tr>
<tr>
<td>X'2C'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>45</td>
</tr>
<tr>
<td>X'2D'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>46</td>
</tr>
<tr>
<td>X'2E'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>47</td>
</tr>
<tr>
<td>X'2F'</td>
<td></td>
</tr>
</tbody>
</table>

*Figure 57. Format of the Standard STD1 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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'80'</td>
<td>Indicates a default image</td>
</tr>
<tr>
<td>X'00'</td>
<td>Indicates a nondefault image</td>
</tr>
</tbody>
</table>

Figure 58. Format of the Standard STD2 FCB Image

See the following information:

- "Adding an FCB Image to the Image Library"
- "Modifying an FCB Image" on page 475

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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'80'</td>
<td>Indicates a default image</td>
</tr>
<tr>
<td>X'00'</td>
<td>Indicates a nondefault image</td>
</tr>
</tbody>
</table>
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.

**Requirement:** 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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'1n'</td>
<td>Sets 8 LPI</td>
</tr>
<tr>
<td>X'0n'</td>
<td>Sets 6 LPI</td>
</tr>
</tbody>
</table>

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 59 on page 475 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).
For **Figure 59**, be aware of the following:

- The **RENT** linkage editor attribute is required.
- Executing the **ASMHCL** procedure does not generate executable code. The assembler/linkage editor is used to place the FCB image into SYS1.IMAGELIB.
- 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 [z/OS DFSMS Using Data Sets].

The formats of the BLDL and SETPRT macros are given in [z/OS DFSMS Macro
Instructions for Data Sets], the format of the LOAD macro is given in [z/OS MVS
Programming: Assembler Services Guide] and [z/OS MVS Programming: Assembler
Services Reference ABE-HSP].

The format of the IMGLIB macro is:

```
IMGLIB label
OPEN
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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (X'00')</td>
<td>Operation successful.</td>
</tr>
<tr>
<td>4 (X'04')</td>
<td>Either the volume containing SYS1.IMAGELIB is not mounted or a required catalog volume is not mounted.</td>
</tr>
<tr>
<td>8 (X'08')</td>
<td>Either SYS1.IMAGELIB does not exist on the volume to which the catalog points, or SYS1.IMAGELIB is not cataloged.</td>
</tr>
<tr>
<td>12 (X'0C')</td>
<td>An error occurred in reading the catalog or VTOC.</td>
</tr>
</tbody>
</table>

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:

- Specification of the INDEX parameter on the /*OUTPUT card.
- 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 C. Using the extended address volume (EAV) migration assistance tracker

The EAV migration assistance tracker can help you find programs that you might need to change if you want to support extended address volumes (EAV). The EAV migration assistance tracker is an extension of the console ID tracking facility. It helps you:

- Identify select systems services by job and program name, where the invoking programs might require analysis for changes to use new services. The program calls are identified as informational instances for possible migration actions. They are not considered errors, because the services return valid information.
- Identify possible instances of improper use of returned information in programs, like parsing 28-bit cylinder numbers in output as 16-bit cylinder numbers. These instances are identified as warnings.
- Identify instances of programs that will either fail or run with an informational message if they run on an EAV. These are identified as programs in error. The migration assistance tracker flags programs with the following functions, when the target volume of the operation is non-EAV, and the function invoked did not specify the EADSCB=OK keyword:
  - OBTAIN
  - CVAFDIR
  - CVAFSEQ
  - CVAFDSM
  - CVAFFILT
  - CVAFVSM - Note that the CVAFVSM interface is an internal system function that is not documented externally for general use.
  - OPEN of VTOC
  - DCB OPEN of an EAS eligible data set

This allows the system programmer to identify programs in error by job and program name, without failing the programs. It also allows you to exclude programs that are not ready for evaluation.

Programs identified in this phase of migration assistance tracking will continue to fail if the system service is issued for an EAV if you do not specify the EADSCB=OK keyword for them.

The EAV migration assistance tracker can be manipulated with the following commands:

- The SETGTZ operator command, which is used to activate and deactivate the tracking facility.
- The DISPLAY GTZ[,STATUS] operator command, which is used to display the current status of the tracking facility.
- The DISPLAY GTZ,TRACKDATA operator command, which is used to display any recorded instances of violations.
- The GTZPRMxx dynamic parmlib member, which is used to list violations that have already been identified in order to prevent them from being recorded again.

See Console ID tracking facility in z/OS MVS Diagnosis: Tools and Service Aids for more information.
EAV migration assistance tracker

The following describe the instances identified and recorded by the EAV migration assistance tracker:

- “Information conventions for the EAV migration assistance tracker”
- “DFSMS instances tracked by the EAV migration assistance tracker” on page 481

Information conventions for the EAV migration assistance tracker

The information returned by the EAV migration assistance tracker describes the occurrence of an instance in text. Like the console ID tracking facility, the EAV migration assistance tracker returns tracking information and a tracking value. The tracking information can be from 1 to 28 characters in length and the system can set any EBCDIC value. The tracking value is four bytes of binary data associated with this instance. For DFSMS, these values include data to associate an instance to a specific DFSMS function and to define the reason for the instance being recorded. This standard allows for maximum flexibility in defining exclusion records that apply to DFSMS records.

The conventions for the tracking information and tracking value for a DFSMS instance follow.

Tracking information

The tracking information for a DFSMS instance might look as follows:

'SMS-I:3 LSPACE MSG='

It can be broken down into several parts:

1. The first portion of the tracking information will be set to ‘SMS-’ to identify this as a DFSMS instance.
2. Appended to this is an error category, of “E” for error, “W” for warning, or “I” for informational, followed by a colon.
3. Appended to the colon is a numeric value that will identify the reason for the recorded instance. These values are
   - 1 - EAV migration: EADSCB=OK keyword was not specified on an invoking program where the target volume was non-EAV. The invoking program fails if the target volume is an EAV.
   - The invoking program would fail if the target volume was an EAV. The following section of the tracking information indicates the error that would have occurred. This instance is recorded in the tracker as an error message.
   - 2 - EAV migration. Formatted output display may contain 28-bit cylinder numbers. Program usage of these track addresses may need to be changed. Use macro TRKADDR for the comparison and manipulation of 28-bit cylinder numbers. This instance is recorded in the tracker as a warning message.
   - 3 - EAV migration. The new function is available on the invoking program. The identified program may want to exploit the available new function. This instance is recorded in the tracker as an informational message.
4. The remaining tracking information must be an EBCDIC value that describes the function executing when the tracker recorded the instance.

In the example above, the 'SMS-I:3 LSPACE MSG=' tracking information describes a DFSMS instance as an informational instance where new function is available that the invoking program might want to exploit. The function running when the tracker recorded this instance was LSPACE MSG=.
EAV migration assistance tracker

Tracking value

For DFSMS instances, the EAV migration assistance tracker sets the tracking value as follows:

- The system sets the low order byte of the track value to the same numeric value that identifies the reason for the instance.
- The remaining high order 3-bytes are left for the function recording the tracked instance. These 3-bytes are optional. For example, these values could be set to return and reason codes or parameter list flags.

Note that the tracking value must be set to a non-zero value in order for the SETCON TRACKING=ONWITHABEND to be applicable when an instance is recorded in the tracker. If the track value is 0 or 128 no ABEND will be issued when you specify tracking ONWITHABEND.

DFSMS instances tracked by the EAV migration assistance tracker

LSPACE (SVC 78)

An LSPACE request with the DATA=, MSG=, or EXPMSG= keywords was issued. Additional data from track-managed space is available with the EXPDATA= and XEXPMSG= keywords. When this instance occurs for any volume type, it will be recorded in the tracker as an informational message.

These five keywords are mutually exclusive.

LSPACE processing (IGC0007H) will set the following tracking information:

TRPL_TRACK_INFO =

'SMS-I:3 LSPACE reqtype '
where,
reqtype = DATA= or MSG= or EXPMSG=

TRPL_TRACK_DATA =

Byte 0-1
Set to zero, not used.

Byte 2
Set to the LSPACE parameter flag byte
BIT 2 ON INDICATES THAT LSPACE WITH THE DATA= KEYWORD WAS SPECIFIED
BIT 3 ON INDICATES THAT LSPACE WITH THE MSG= KEYWORD WAS SPECIFIED
BIT 4 ON INDICATES THAT LSPACE WITH THE EXPMSG= KEYWORD WAS SPECIFIED

Byte 3
Set to 03.
DFSMS Tracking category 3: EAV Migration. Informational Message. New function is available. Additional data from track-managed space is available with the EXPDATA= and XEXPMSG= keywords.

TRPL_VIOLATORS_ADDR =

SET TO THE RESUME PSW OF THE RB THAT ISSUED SVC 78. TRACER CODE WILL DETERMINE JOB AND PROGRAM NAMES FROM THIS ADDRESS.

SAMPLE OUTPUT =

15.00.00 SYSTEM1 d opdata,tracking
15.00.00 SYSTEM1 CNZI0011 15.00.00 TRACKING DISPLAY 831
STATUS=ON NUM=3 MAX=1000 MEM=7T EXCL=0 REJECT=0
-----TRACKING INFORMATION----- VALUE-- JOBNAME PROGNAME=OPT ASID NUM
SMS-I:3 LSPACE MSG= 5003 ALLOCAS IEFW21SD 4CE5C 11 1
----TRACKING INFORMATION---- VALUE-- JOBNAME PROGNAME=OPT ASID NUM
SMS-I:3 LSPACE MSG= 2003 VTDS01S1 VTDS01S2 118 28 2
SMS-I:3 LSPACE MSG= 8803 VTDS01S1 VTDS01S2 118 28 2
SMS-I:3 LSPACE MSG= 9003 MASTER IEE70110 52F6 01 46

Appendix C. Using the extended address volume (EAV) migration assistance tracker 481
EAV migration assistance tracker

SAMPLE EXCLUSION LIST =

* Tracking Information Mask | Mask | Mask | Comments (ignored) *
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SMS-I:3 LSPACE*</td>
<td>MASTER</td>
<td>IEE70110</td>
<td>I SMF CALLS TO LSPACE</td>
</tr>
<tr>
<td>SMS-I:3 LSPACE*</td>
<td>ALLOCAS</td>
<td>IFW21SD</td>
<td>VARY DEVICE OFFLINE</td>
</tr>
<tr>
<td>SMS-I:3 LSPACE*</td>
<td>*</td>
<td>VTDSOIS2</td>
<td>VTDSOIS2 PROG CALLS</td>
</tr>
<tr>
<td>SMS-I:3 LSPACE*</td>
<td>VTDSOIS1</td>
<td>*</td>
<td>VTDSOIS1 JOB CALLS</td>
</tr>
</tbody>
</table>

DEVTYPE (SVC 24)

A DEVTYPE request with DEVTAB or UCBLIST without INFOLIST, returns the number of cylinders on the volume. This is in a two-byte field at offset 8, which is too small if the volume has more than 65,520 cylinders. Consider using INFO=DASD which returns the number of cylinders in a four-byte field. When INFO=DASD is specified additional fields are now provided. They include (see mapping macro IHADVA):

DVAIXVLD BIT DVAICYLMG, DVAEADSCB, DVAVIRSZ valid

DVAICYLMG BIT Cylinder-managed space exists on
| this volume and begins at DVAIXCYL |
| * in multicylinder units of DVAMCU. |
| * DVAEADSCB is also set with this |
| * flag on. Valid when DVAIXVLD is set. |

DVAEADSCB BIT Extended attribute DSCBs, Format 8
| * and 9 DSCBs, are allowed on this |
| volume. Valid when DVAIXVLD is set. |

DVAMCU 8-BIT UNSIGNED INTEGER
| Minimum allocation size in |
| * cylinders for cylinder-managed |
| * space. Each extent in this space |
| * must be a multiple of this value. |
| * space. Also referred to as the |
| * multicylinder unit (MCU). This is |
| * the smallest unit of disk space in |
| * cylinders that can be allocated |
| * in cylinder-managed space. |
| * Valid when DVAICYLMG is set. |
| * This field is zero on releases |
| * before z/OS 1.10 or if the status |
| * is not yet known. In these two |
| * cases DVAIXVLD is not set. |

DVAICYL 16-BIT UNSIGNED INTEGER
| First cylinder address divided by |
| * 4095 where space is managed in |
| * multicylinder units. Cyl-managed |
| * space begins at this address. |
| * Valid when DVAICYLMG is set. This |
| * field is zero on releases before |
| * z/OS 1.10 or if the status is not |
| * yet known. In these two cases |
| * DVAIXVLD is not set. |

DVAVIRSZ 16-BIT UNSIGNED INTEGER
| Block size of the index data set. |
| * Valid when DVAIXVLD is set on. |
| * When valid and zero the volume |
| * has no working VTOC index. This |
| * field is zero on releases before |
| * z/OS 1.10 or if the status is not |
| * yet known. In these cases |
| * DVAIXVLD is not set. |

When this instance occurs for any volume type, it will be recorded in the tracker as an informational message.
DEVTYPE processing (IGC0002D) will set the following tracking information:

```
TRPL_TRACK_INFO =
   'SMS-I:3 DEVTYPE'
TRPL_TRACK_DATA =
   Byte 0-2
      Set to zero, not used.
   Byte 3
      Set to 03.
      DFSMS Tracking category 3: EAV Migration. Informational Message. New function is available. Additional data from DEVTYPE INFO=DASD invocation is available. See mapping macro IHADVA.
TRPL_VIOLATORS_ADDR =
   SET TO THE RESUME PSW OF THE RB THAT ISSUED SVC 24. TRACKER CODE WILL DETERMINE JOB AND PROGRAM NAMES FROM THIS ADDRESS.
```

SAMPLE OUTPUT =

```
15.00.00 SYSTEM1 d opdata,tracking
15.00.00 SYSTEM1 CNZ1001I 15.00.00 TRACKING DISPLAY 631
STATUS=ON NUM=1 MAX=1000 MEM=7T EXCL=0 REJECT=0
----TRACKING INFORMATION---- -VALUE-- JOBNAME PROGNAME+OFF-- ASID NUM
SMS-I:3 DEVTYPE 03 DEVTJOB DEVTPROG 4CE5C 11 1
-----
```

SAMPLE EXCLUSION LIST =

```
* Jobname Pgmname *
* Tracking Information Mask Mask Mask Comments (ignored) *
*----------------------------+--------+--------+----------------------*
|SMS*DEVTYPE* |* |* | ALL DEVTYPE |
```

**IDCAMS LISTDATA PINNED**

An IDCAMS LISTDATA PINNED request was processed. The track addresses for the PINNED tracks may contain 28-bit cylinder numbers.

When this instance occurs for any volume type, it will be recorded in the tracker as a warning message.

IDCAMS LISTDATA PINNED processing (IDCSS05) will set the following tracking information:

```
TRPL_TRACK_INFO =
   'SMS-W:2 IDCAMS LISTDATA PINN'
TRPL_TRACK_DATA =
   Byte 0-2
      Set to zero, not used.
   Byte 3
      Set to 02.
      DFSMS Tracking category 2: EAV Migration. Warning Message. An IDCAMS LISTDATA PINNED request was processed. The track addresses for the PINNED tracks may contain 28-bit cylinder numbers.
TRPL_VIOLATORS_ADDR =
   SET TO THE RESUME PSW OF THE RB WITH A VALID POINTER (RBCDE1) TO THE 'IDCAMS' CNAME. USE THE PREVIOUS RB (RBLINKB) IF THIS RB IS NOT THE FIRST RB. TRACKER CODE WILL DETERMINE JOB AND PROGRAM NAMES FROM THIS ADDRESS.
```

SAMPLE OUTPUT =

```
07.34.08 SYSTEM1 d opdata,tracking
07.34.08 SYSTEM1 CNZ1001I 07.34.08 TRACKING DISPLAY 673
STATUS=ON,ABEND NUM=1 MAX=1000 MEM=7T EXCL=5 REJECT=0
----TRACKING INFORMATION---- -VALUE-- JOBNAME PROGNAME+OFF-- ASID NUM
```
IEHLIST LISTVTOC

An IEHLIST LISTVTOC request was processed. Extent descriptors may contain cylinder addresses 65520 or larger. Free space descriptors may contain track addresses 982800 or larger and/or full cylinders 65520 or larger. The generated report will display the information in different columns as compared to reports generated on releases prior to z/OS V1.10.

When this instance occurs for any volume type, it will be recorded in the tracker as a warning message.

IEHLIST LISTVTOC processing will set the following tracking information:

TRPL_TRACK_INFO =
'SMS-W:2 IEHLIST LISTVTOC '

TRPL_TRACK_DATA =

Byte 0-2
Set to zero, not used.

Byte 3
Set to 02.

DFSMs Tracking category 2: EAV Migration. Warning Message.

An IEHLIST LISTVTOC request was processed. Extent descriptors may contain cylinder addresses 65520 or larger. Free space descriptors may contain track addresses 982800 or larger and/or full cylinders 65520 or larger. The generated report will display the information in different columns as compared to reports generated on releases prior to z/OS V1.10.

TRPL_VIOLATORS_ADDR =

SET TO THE RESUME PSW OF THE PRB WITH A VALID POINTER (RBCDE1) TO THE 'IEHLIST' CDNAME.
USE THE PREVIOUS R8 (RBLINKB) IF THIS PRB IS NOT THE FIRST RB. TRACKER CODE WILL DETERMINE JOB AND PROGRAM NAMES FROM THIS ADDRESS.

SAMPLE OUTPUT =

| SMS-W:2 IEHLIST LISTVTOC | 02 LST004 IEHLIST 2304 29 1 |
| SMS-W:2 IEHLIST LISTVTOC | 02 LST004 IEHLIST 34A6 29 1 |

484  z/OS V2R1.0 DFSMSdfp Advanced Services
EAV migration assistance tracker

IDCAMS DCOLLECT

An IDCAMS DCOLLECT request for 'V' (Volume Record Field) and 'VL' (SMS Volume Definition Field) records was processed. Additional data for track-managed space was recorded.

When this instance occurs for any volume type, it will be recorded in the tracker as an informational message.

IDCAMS DCOLLECT processing will set the following tracking information:

TRPL_TRACK_INFO =
'SMS-1:3 IDCAMS DCOLLECT ' 

TRPL_TRACK_DATA =

Byte 0-2
Set to zero, not used.

Byte 3
Set to 03.

DFSMS Tracking category 3: EAV Migration. Informational Message. An IDCAMS DCOLLECT request for 'V' (Volume Record Field) and 'VL' (SMS Volume Definition Field) records was processed. Additional data for track-managed space was recorded.

TRPL_VIOLATORS_ADDR =

SET TO THE RESUME PSW OF THE PRB WITH A VALID POINTER (RBCDE1) TO THE 'IDCAMS' CDNAME. USE THE PREVIOUS RB (RBLINKB) IF THIS PRB IS NOT THE FIRST RB. TRACKER CODE WILL DETERMINE JOB AND PROGRAM NAMES FROM THIS ADDRESS.

SAMPLE OUTPUT =

07.34.08 SYSTEM1 d opdata,tracking
07.34.08 SYSTEM1 CNZ1001I 07.34.08 TRACKING DISPLAY 673
STATUS=ON,ABEND NUM=1 MAX=1000 MEM=7T EXCL=5 REJECT=0

----TRACKING INFORMATION----

| SMS-I:3 IDCAMS DCOLLECT 03 DCOLLECT IDCAMS xxx 28 4 |

SAMPLE EXCLUSION LIST =

* Tracking Information Mask Mask Comments (ignored) *

| SMS+DCOLLECT* | * | IDCAMS | IDCAMS DCOLLECT PGM |

IDCAMS LISTCAT

An IDCAMS LISTCAT request was processed that printed extent descriptors for one or more EAS eligible data set (VSAM in z/OS V1R10). The returned extent descriptors may contain 28-bit cylinder numbers.

When this instance occurs for any volume type, it will be recorded in the tracker as a warning message.

IDCAMS LISTCAT processing will set the following tracking information:

TRPL_TRACK_INFO =
'SMS-N:2 IDCAMS LISTCAT ' 

TRPL_TRACK_DATA =

Byte 0-2
Set to zero, not used.
EAV migration assistance tracker

Byte 3

Set to 02.

DFSMS Tracking category 2: EAV Migration. Warning Message.
An IDCAMS LISTCAT request was processed that printed extent descriptors for one or more EAS eligible data set (VSAM in z/OS V1R10).
The returned extent descriptors may contain 28-bit cylinder numbers.
This instance will be recorded for both EAS and non-EAS capable volumes.
Please note that this listcat output format may change as a result of service and new function support. IBM recommends applications processing LISTCAT output be updated to obtain results directly from the Catalog Search Interface (CSI). For more information on CSI, see z/OS DFSMS Managing Catalogs and HLASM Programmer's Guide.

TRPL_VIOLATORS_ADDR =

SET TO THE RESUME PSW OF THE PRB WITH A VALID POINTER (RBSCD1) TO THE 'IDCAMS' CDNAME. USE THE PREVIOUS RB (RBLINKB) IF THIS PRB IS NOT THE FIRST RB. TRACKER CODE WILL DETERMINE JOB AND PROGRAM NAMES FROM THIS ADDRESS.

SAMPLE OUTPUT =

07.34.08 SYSTEM1 d opdata,tracking
07.34.08 SYSTEM1 CNZ1001I 07.34.08 TRACKING DISPLAY 673
SMS=2 IDCAMS LISTCAT 02 LISTCAT IDCAMS xxx 28 4

SAMPLE EXCLUSION LIST =

<table>
<thead>
<tr>
<th>Jobname</th>
<th>Pgmname</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tracking Information mask</td>
<td>Mask</td>
</tr>
</tbody>
</table>

| SMS+IDCAMS LISTCAT | * | IDCAMS | IDCAMS LISTCAT PGM |

OBTAIN (SVC 27)

OBTAIN was issued with the search or seek option to a non-EAV volume. The caller did not specify with EADSCB=OK that it supports the extended attribute DSCBs and the target data set is EAS eligible.

When this instance occurs for a non-EAV volume type, it will be recorded in the tracker as an error message.

OBTAIN processing will set the following tracking information:

TRPL_TRACK_INFO =

'SMS=E:1 DADSM OBTAIN'

TRPL_TRACK_DATA =

Byte 0

Set to zero, not used.

Byte 1-2

Operation code.

X'C100' SEARCH for DSNAME.

X'C080' SEEK for track address.

Byte 3

Set to 01.

DFSMS Tracking category 1: EAV Migration. Error Message. DADSM OBTAIN was issued with the search or seek option to a non-EAV volume. The caller did not specify with EADSCB=OK that it supports the extended attribute DSCBs and the target data set is 'EAS eligible'.
EAV migration assistance tracker

TRPL_VIOLATORS_ADDR =

    SET TO THE RESUME PSW OF THE RB THAT ISSUED SVC 27. TRACKER CODE WILL
    DETERMINE JOB AND PROGRAM NAMES FROM THIS ADDRESS.

SAMPLE OUTPUT =

07.34.08 SYSTEM1 d opdata,tracking
07.34.08 SYSTEM1 CNZ1001I 07.34.08 TRACKING DISPLAY 673
STATUS=ON,ABEND NUM=1 MAX=1000 MEM=7T EXCL=5 REJECT=0
----TRACKING INFORMATION---- VALUE-- JOBNAME PROGNAME+OFF-- ASID NUM
SMS-E:1 DADSM OBTAIN C08001 OBITJBN OBTPGM xxx 20 4

SAMPLE EXCLUSION LIST =

* Jobname Pgmname *
* Tracking Information Mask Mask Mask Comments (ignored) *
*----------------------------+--------+--------+----------------------*
|SMS*OBTAIN* |* |* | DADSM OBTAIN |

CVAFDIR

CVAFDIR was issued with the search or seek option to a volume that does not
support extended attribute DSCBs. The caller did not specify with EADSCB=OK
that it supports the extended attribute DSCBs and the target data set is 'EAS
eligible'. CVAF return code 4 and CVSTAT of 'X'52' would have been set if issued
to a volume that supports extended attribute DSCBs.

When this instance occurs for a volume that does not support extended attribute
DSCBs, it will be recorded in the tracker as an error message.

CVAFDIR processing will set the following tracking information:

TRPL_TRACK_INFO =

    'SMS-E:1 CVAFDIR STAT082 '

TRPL_TRACK_DATA =

    Byte 0
    Set to zero, not used.

    Byte 1
    CVAF Return Code = 4

    Byte 2
    CVAF Status Code = STAT082 (X'52')

    Byte 3
    Set to 01.

DFSMS Tracking category 1: EAV Migration, Error Message. CVAFDIR was
issued with the search or seek option to a volume that does not
support extended attribute DSCBs. The caller did not specify with
EADSCB=OK that it supports the extended attribute DSCBs and the
target data set is 'EAS eligible'. CVAF return code 4 and CVSTAT of
X'52' would have been set if issued to a device that supports
extended attribute DSCBs.

TRPL_VIOLATORS_ADDR =

    FOR BRANCH ENTRY CALLERS, SET TO THE CALLER OF CVAF RETURN ADDRESS.
    TRACKER CODE WILL DETERMINE JOB AND PROGRAM NAMES FROM THIS ADDRESS.

    FOR SVC CALLS, SET TO THE RESUME PSW OF THE RB THAT ISSUED SVC 139.
    TRACKER CODE WILL DETERMINE JOB AND PROGRAM NAMES FROM THIS ADDRESS.

SAMPLE OUTPUT =

Appendix C. Using the extended address volume (EAV) migration assistance tracker 487
EAV migration assistance tracker

CVAFSEQ

CVAFSEQ was issued for physical sequential or index order to a volume that does not support extended attribute DSCBs. The caller did not specify with EADSCB=OK that it supports the extended attribute DSCBs and the target data set is 'EAS eligible'. CVAF return code 4 and CVSTAT of X'52' would have been set if issued to a volume that supports extended attribute DSCBs.

When this instance occurs for a volume that does not support extended attribute DSCBs, it will be recorded in the tracker as an error message.

CVAFSEQ processing will set the following tracking information:

TRPL_TRACK_INFO =
'SMS-E:1 CVAFSEQ STAT082'

TRPL_TRACK_DATA =
Byte 0
Set to zero, not used.

Byte 1
CVAF Return Code = 4

Byte 2
CVAF Status Code = STAT082 (X'52')

Byte 3
Set to 01.

DFSMS Tracking category 1: EAV Migration. Error Message. CVAFSEQ was issued for physical sequential or index order to a volume that does not support extended attribute DSCBs. The caller did not specify with EADSCB=OK that it supports the extended attribute DSCBs and the target data set is 'EAS eligible'. CVAF return code 4 and CVSTAT of X'52' would have been set if issued to a volume that supports extended attribute DSCBs.

TRPL_VIOLATORS_ADDR =

FOR BRANCH ENTRY CALLERS, SET TO THE CALLER OF CVAF RETURN ADDRESS. TRACKER CODE WILL DETERMINE JOB AND PROGRAM NAMES FROM THIS ADDRESS.

FOR SVC CALLS, SET TO THE RESUME PSW OF THE RB THAT ISSUED SVC 139. TRACKER CODE WILL DETERMINE JOB AND PROGRAM NAMES FROM THIS ADDRESS.

SAMPLE OUTPUT =

07.34.08 SYSTEMI d opdata,tracking
07.34.08 SYSTEMI CNZ1001I 07.34.08 TRACKING DISPLAY 673
STATUS=ON,ABEND NUM=1 MAX=1000 MEM=7T EXCL=5 REJECT=0
------TRACKING INFORMATION------ -VALUE-- JOBNAME PROGNAME+OFF-- ASID NUM
SMS-E:1 CVAFSEQ STAT082 045201 CVAFJBN CVAFPGM xxx 2B 4

SAMPLE EXCLUSION LIST =

<table>
<thead>
<tr>
<th>SMS=CVAFDIR STAT082</th>
<th>*</th>
<th>*</th>
<th>CVAFDIR</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMS=CVAFDIR STAT082</td>
<td>*</td>
<td>*</td>
<td>CVAFDIR</td>
</tr>
</tbody>
</table>
EAV migration assistance tracker

CVAFDSM

CVAFDSM was issued to retrieve unallocated space on a volume that does not support extended attribute DSCBs. The caller did not specify with EADSCB=OK that it supports the extended attribute DSCBs. CVAF return code 4 and CVSTAT of X'52' would have been set if issued to a volume that supports extended attribute DSCBs.

When this instance occurs for a volume that does not support extended attribute DSCBs, it will be recorded in the tracker as an error message.

CVAFDSM processing will set the following tracking information:

\[
\text{TRPL\_TRACK\_INFO} = \\
'SMS-E:1 CVAFDSM STAT082' \\
\text{TRPL\_TRACK\_DATA} = \\
\text{Byte 0} \\
\text{Set to zero, not used.} \\
\text{Byte 1} \\
\text{CVAF Return Code} = 4 \\
\text{Byte 2} \\
\text{CVAF Status Code} = \text{STAT082 (X'52')} \\
\text{Byte 3} \\
\text{Set to 01.} \\
\]

DFSM tracking category 1: EAV Migration. Error Message. CVAFDSM was issued to retrieve unallocated space on a volume (CVAFDSM ACCESS=MAPDATA, MAP=VOLUME, RTA4BYTE=YES) that does not support extended attribute DSCBs. The caller did not specify with EADSCB=OK that it supports the extended attribute DSCBs. CVAF return code 4 and CVSTAT of X'52' would have been set if issued to a volume that supports extended attribute DSCBs.

CVAFVSM interface is an internal system function that is not documented externally for general use.

\[
\text{TRPL\_VIOLATORS\_ADDR} = \\
\text{FOR BRANCH ENTRY CALLERS, SET TO THE CALLER OF CVAF RETURN ADDRESS.} \\
\text{TRACKER CODE WILL DETERMINE JOB AND PROGRAM NAMES FROM THIS ADDRESS.} \\
\text{FOR SVC CALLS, SET TO THE RESUME PSW OF THE RB THAT ISSUED SVC 139.} \\
\text{TRACKER CODE WILL DETERMINE JOB AND PROGRAM NAMES FROM THIS ADDRESS.} \\
\]

SAMPLE OUTPUT =

\[
07.34.08 \text{ SYSTEM1 d opdata,tracking} \\
07.34.08 \text{ SYSTEM1 CNZ1001I 07.34.08 TRACKING DISPLAY 673} \\
\text{STATUS=ON, ABEND NUM=1 MAX=1000 MEM=7T EXCL=5 REJECT=0} \\
\text{----TRACKING INFORMATION---- -VALUE-- JOBNAME PGMNAME OFF-- ASID NUM} \\
'SMS-E:1 CVAFDSM STAT082 045201 CVAFJBN CVAFPGM xxx 20 4' \\
\]

SAMPLE EXCLUSION LIST =

\[
* \text{ Tracking Information Mask} \text{ Mask} \text{ Mask} \text{ Comments (ignored)} * \\
\text{----------------------------+--------+--------+----------------------+} \\
|SMS+CVAFDSM STAT082* |* |* | CVAFDSM |
\]
CVAFFILT

CVAFFILT was issued to obtain DSCB information for fully or partially qualified data set names on a volume that does not support extended attribute DSCBs. The caller did not specify with EADSCB=OK that it supports the extended attribute DSCBs and the qualified data set is 'EAS eligible'. CVAF return code 4 and CVSTAT of X'56' along with data set name status in the FCL (FCLDSNST) of X'06' would have been set if the request was issued to a volume that supports extended attribute DSCBs.

When this instance occurs for a volume that does not support extended attribute DSCBs, it will be recorded in the tracker as an error message.

CVAFFILT processing will set the following tracking information:

TRPL_TRACK_INFO =

'SMS-E:1 CVAFFILT STAT086'

TRPL_TRACK_DATA =

Byte 0

CVAF Return Code = 4

Byte 1

CVAF Status Code = STAT086 (X'56')

Byte 2

FCL data set status code = X'06'

Byte 3

Set to 01.

DFSMS Tracking category 1: EAV Migration. Error Message. CVAFFILT was issued to obtain DSCB information for fully or partially qualified data set names on a volume that does not support extended attribute DSCBs. The caller did not specify with EADSCB=OK that it supports the extended attribute DSCBs and the qualified data set is 'EAS eligible'. CVAF return code 4 and CVSTAT of X'56' along with data set name status in the FCL (FCLDSNST) of X'06' would have been set if the request was issued to a volume that supports extended attribute DSCBs.

TRPL_VIOLATORS_ADDR =

FOR BRANCH ENTRY CALLERS, SET TO THE CALLER OF CVAF RETURN ADDRESS. TRACKER CODE WILL DETERMINE JOB AND PROGRAM NAMES FROM THIS ADDRESS.

FOR SVC CALLS, SET TO THE RESUME PSW OF THE RB THAT ISSUED SVC 139. TRACKER CODE WILL DETERMINE JOB AND PROGRAM NAMES FROM THIS ADDRESS.

SAMPLE OUTPUT =

07.34.08 SYSTEM1 d opdata,tracking
07.34.08 SYSTEM1 CNZ1001I 07.34.08 TRACKING DISPLAY 673
STATUS=ON,ABEND NUM=1 MAX=1000 MEM=7T EXCL=5 REJECT=0
----TRACKING INFORMATION---- -VALUE-- JOBNAME PROGNAME+OFF-- ASID NUM
SMS-E:1 CVAFFILT STAT086 04560601 CVAFJBN CVAFPGM xxx 28 4

SAMPLE EXCLUSION LIST =

* Jobname Pgmname *
CVAFVSM

CVAFVSM was issued to allocate space for a volume that is not an EAV. The caller did not specify with EADSCB=OK that it supports an EAV. CVAF return code 4 and CVSTAT of X’52’ would have been set if issued to an EAV.

When this instance occurs for a non EAV, it will be recorded in the tracker as an error message.

Note that the CVAFVSM interface is an internal system function that is not documented externally for general use.

CVAFVSM processing will set the following tracking information:

TRPL_TRACK_INFO =
'SMS-E:1 CVAFVSM STAT082 '

TRPL_TRACK_DATA =
Byte 0
Set to zero, not used.

Byte 1
CVAF Return Code = 4

Byte 2
CVAF Status Code = STAT082 (X’52’)

Byte 3
Set to 01.

DFSM Tracking category 1: EAV Migration. Error message. CVAFVSM was issued to allocate space for a volume that is not an EAV. The caller did not specify with EADSCB=OK that it supports an EAV. CVAF return code 4 and CVSTAT of X’52’ would have been set if issued to an EAV. CVAFVSM interface is an internal system function that is not documented externally for general use.

When this instance occurs for a non EAV, it will be recorded in the tracker as an error message.

TRPL_VIOLATORS_ADDR =
FOR BRANCH ENTRY CALLERS, SET TO THE CALLER OF CVAF RETURN ADDRESS.
TRACKER CODE WILL DETERMINE JOB AND PROGRAM NAMES FROM THIS ADDRESS.

FOR SVC CALLS, SET TO THE RESUME PSW OF THE RB THAT ISSUED SVC 139.
TRACKER CODE WILL DETERMINE JOB AND PROGRAM NAMES FROM THIS

SAMPLE OUTPUT =

07.34.08 SYSTEM1 d opdata,tracking
07.34.08 SYSTEM1 CNZ1001I 07.34.08 TRACKING DISPLAY 673
STATUS=ON,ABEND NUM=1 MAX=1000 MEM=7T EXCL=5 REJECT=0
----TRACKING INFORMATION---- VALUE JOBNAME PROGNAME+OFF ASID NUM
SMS-E:1 CVAFVSM STAT082 045201 CVAFJBN CVAFPGM xxx 28 4

SAMPLE EXCLUSION LIST =
* Jobname Pgmname *
**DCB Open of a VTOC**

A DCB Open of a VTOC was issued to a volume that does not support extended attribute DSCBs. The caller did not specify EADSCB=OK on the DCBE macro indicating that it supports the extended attribute DSCBs in the VTOC. Open would have issued an ABEND, MSGIEC142I 113-48 if an attempt was made to open the VTOC of a volume that supported extended attribute DSCBs.

When this instance occurs for a volume that does not support extended attribute DSCBs, it will be recorded in the tracker as an error message.

OPEN processing will set the following tracking information:

**TRPL_TRACK_INFO =**

'SMS-E:1 DCB OPEN VTOC 113-48'

**TRPL_TRACK_DATA =**

| Byte 0-2 | Set to zero, not used. |
| Byte 3  | Set to 01. |

DFSM Tracking category 1: EAV Migration. Error Message. A DCB Open of a VTOC was issued to a volume that does not support extended attribute DSCBs. The caller did not specify EADSCB=OK on the DCBE macro indicating that it supports the extended attribute DSCBs in the VTOC. Open would have issued an ABEND, MSGIEC142I 113-48 if an attempt was made to open the VTOC of a volume that supported extended attribute DSCBs.

**TRPL_VIOLATORS_ADDR =**

Set to the resume PSW of the RB that issued the OPEN SVC. Tracker code will determine job and program names from this address.

**SAMPLE OUTPUT =**

```
07.34.08 SYSTEM1 d opdata,tracking
07.34.08 SYSTEM1 CNZ1001I 07.34.08 TRACKING DISPLAY 673
STATUS=OK,ABEND NUM=1 MAX=1000 MEM=7T EXCL=5 REJECT=0
----TRACKING INFORMATION---- VALUE-- JOBNAME PROGNAME+OFF-- ASID NUM
SMS-E:1 DCB OPEN VTOC 113-48 01 OPENJBN OPENPGM xxx 28 4
```

**SAMPLE EXCLUSION LIST =**

```
* Tracking Information Mask Jobname Pgmname |
*SMS+DCB OPEN VTOC* |
```

**DCB Open of EAS eligible data set**

A DCB Open (MACRF = E for EXCP) of an EAS eligible data set was issued to a volume that does not support extended attribute DSCBs. The caller did not specify with EADSCB=OK on the DCBE macro to indicate that it supports the extended attribute DSCBs for an EAS eligible data set. Open would have issued an ABEND, MSGIEC142I 113-44 if an attempt was made to open the EAS eligible data set on a volume that supported extended attribute DSCBs.
When this instance occurs for a volume that does not support extended attribute DSCBs, it will be recorded in the tracker as an error message.

OPEN processing will set the following tracking information:

```
TRPL_TRACK_INFO =

'SMS-E:1 DCB OPEN EAS 113-44'
```

```
TRPL_TRACK_DATA =

Byte 0-2
Set to zero, not used.

Byte 3
Set to 01.
```

DFSMS Tracking category 1: EAV Migration. Error Message. A DCB Open (MACRF = E for EXCP) of an EAS eligible data set was issued to a volume that does not support extended attribute DSCBs. The caller did not specify with EADSCB=OK on the DCBE macro to indicate that it supports the extended attribute DSCBs for an EAS eligible data set. Open would have issued an ABEND, MSGIEC142I 113-44 if an attempt was made to open the EAS eligible data set on a volume that supported extended attribute DSCBs.

```
TRPL_VIOLATORS_ADDR =

SET TO THE RESUME PSW OF THE RB THAT ISSUED THE OPEN SVC. TRACKER CODE WILL DETERMINE JOB AND PROGRAM NAMES FROM THIS ADDRESS.
```

SAMPLE OUTPUT =

```
07.34.08 SYSTEM1 d opdata,tracking
07.34.08 SYSTEM1 CNZ1001I 07.34.08 TRACKING DISPLAY 673
STATUS=ON,ABEND NUM=1 MAX=1000 MEM=7T EXCL=5 REJECT=0
----TRACKING INFORMATION---- | VALUE-- JOBNAME | PROGRAM | OFF-- ASID NUM |
SMS-E:1 DCB OPEN EAS 113-44 01 OPENJBN OPENPGM xxx 28 4
```

SAMPLE EXCLUSION LIST =

```
* Jobname Pgmname *
* Tracking Information Mask Mask Mask Comments (ignored) *
*-------------------------------+--------+--------+----------------------*
|SMS*DCB OPEN VSAM* |* |* | MSGIEC142I 113-44 |
```

Other Sample exclusion list

```
SAMPLE EXCLUSION LIST =

* Jobname Pgmname *
* Tracking Information Mask Mask Mask Comments (ignored) *
*-------------------------------+--------+--------+----------------------*
|SMS+DCB OPEN VSAM+ |* |* | MSGIEC142I 113-44 |
```

Other Sample exclusion list

```
SAMPLE EXCLUSION LIST =

* Jobname Pgmname *
* Tracking Information Mask Mask Mask Comments (ignored) *
*-------------------------------+--------+--------+----------------------*
|SMS* |* |* | SMS all instances |
|SMS+E* |* |* | SMS Errors |
|SMS+W* |* |* | SMS Warnings |
|SMS+I* |* |* | SMS Informational |
|SMS+1* |* |* | EAV EADSCB=OK missing |
|SMS+2* |* |* | EAV 28-bitcyls output |
|SMS+3* |* |* | EAV new function |
```

Recommend exclusion list

To be obtained from the following web site in the CNIDTRxx parmlib member:

Recommended EXCLUSION LIST FOR DFSMS =

* Download latest 1.6 Console ID tracker data from
  * http://www-03.ibm.com/servers/eserver/zseries/zos/downloads/
## EAV migration assistance tracker

### Summary of DFSMS instances

<table>
<thead>
<tr>
<th>Time</th>
<th>Action</th>
<th>Instance</th>
<th>Status</th>
<th>Jobname</th>
<th>Progname</th>
<th>Asid</th>
<th>Num</th>
</tr>
</thead>
<tbody>
<tr>
<td>07.34.08</td>
<td>SYSTEM1</td>
<td>opdata,tracking</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>07.34.08</td>
<td>SYSTEM1</td>
<td>CNZ1001I</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>STATUS=ON,ABEND NUM=1 MAX+1000 MEM=7T EXCL=5 REJECT=0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>----TRACKING INFORMATION----</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>VALUE-- JOBNAME</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SMS-E:1</td>
<td>CVAFDIR STAT082</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SMS-E:1</td>
<td>CVAFDSM STAT082</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SMS-E:1</td>
<td>CVAFFILT STAT086</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SMS-E:1</td>
<td>CVAFSEQ STAT082</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SMS-E:1</td>
<td>CVAFVSM STAT082</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SMS-E:1</td>
<td>DADSM OBTAIN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SMS-E:1</td>
<td>DCB OPEN EAS 113-44</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SMS-E:1</td>
<td>DCB OPEN VTDC 113-48</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SMS-I:3</td>
<td>DEVTYPE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SMS-I:3</td>
<td>LSPACE MSG=</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SMS-I:3</td>
<td>LSPACE DATA=</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SMS-I:3</td>
<td>LSPACE EXPMGS=</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SMS-I:3</td>
<td>IDCAMS DCOLLECT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SMS-I:3</td>
<td>IDCAMS LISTCAT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SMS-I:3</td>
<td>IDCAMS LISTDATA PINN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SMS-I:3</td>
<td>IEHLIST LISTVTOC</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

z/OS V2R1.0 DFSMSdfp Advanced Services
Appendix D. Accessibility

Accessible publications for this product are offered through the z/OS Information Center, which is available at www.ibm.com/systems/z/os/zos/bkserv/.

If you experience difficulty with the accessibility of any z/OS information, please send a detailed message to mhvrcks@us.ibm.com or to the following mailing address:

IBM Corporation
Attention: MHVRCFS Reader Comments
Department H6MA, Building 707
2455 South Road
Poughkeepsie, NY 12601-5400
USA

Accessibility features

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use software products successfully. The major accessibility features in z/OS enable users to:

- Use assistive technologies such as screen readers and screen magnifier software
- Operate specific or equivalent features using only the keyboard
- Customize display attributes such as color, contrast, and font size.

Using assistive technologies

Assistive technology products, such as screen readers, function with the user interfaces found in z/OS. Consult the assistive technology documentation for specific information when using such products to access z/OS interfaces.

Keyboard navigation of the user interface

Users can access z/OS user interfaces using TSO/E or ISPF. Refer to z/OS TSO/E Primer, z/OS TSO/E User’s Guide, and z/OS ISPF User’s Guide Vol I for information about accessing TSO/E and ISPF interfaces. These guides describe how to use TSO/E and ISPF, including the use of keyboard shortcuts or function keys (PF keys). Each guide includes the default settings for the PF keys and explains how to modify their functions.

Dotted decimal syntax diagrams

Syntax diagrams are provided in dotted decimal format for users accessing the z/OS Information Center using a screen reader. In dotted decimal format, each syntax element is written on a separate line. If two or more syntax elements are always present together (or always absent together), they can appear on the same line, because they can be considered as a single compound syntax element.

Each line starts with a dotted decimal number; for example, 3 or 3.1 or 3.1.1. To hear these numbers correctly, make sure that your screen reader is set to read out punctuation. All the syntax elements that have the same dotted decimal number (for example, all the syntax elements that have the number 3.1) are mutually
exclusive alternatives. If you hear the lines 3.1 USERID and 3.1 SYSTEMID, you know that your syntax can include either USERID or SYSTEMID, but not both.

The dotted decimal numbering level denotes the level of nesting. For example, if a syntax element with dotted decimal number 3 is followed by a series of syntax elements with dotted decimal number 3.1, all the syntax elements numbered 3.1 are subordinate to the syntax element numbered 3.

Certain words and symbols are used next to the dotted decimal numbers to add information about the syntax elements. Occasionally, these words and symbols might occur at the beginning of the element itself. For ease of identification, if the word or symbol is a part of the syntax element, it is preceded by the backslash (\) character. The * symbol can be used next to a dotted decimal number to indicate that the syntax element repeats. For example, syntax element *FILE with dotted decimal number 3 is given the format 3 \* FILE. Format 3* FILE indicates that syntax element FILE repeats. Format 3* \* FILE indicates that syntax element * FILE repeats.

Characters such as commas, which are used to separate a string of syntax elements, are shown in the syntax just before the items they separate. These characters can appear on the same line as each item, or on a separate line with the same dotted decimal number as the relevant items. The line can also show another symbol giving information about the syntax elements. For example, the lines 5.1*, 5.1 LASTRUN, and 5.1 DELETE mean that if you use more than one of the LASTRUN and DELETE syntax elements, the elements must be separated by a comma. If no separator is given, assume that you use a blank to separate each syntax element.

If a syntax element is preceded by the % symbol, this indicates a reference that is defined elsewhere. The string following the % symbol is the name of a syntax fragment rather than a literal. For example, the line 2.1 %OP1 means that you should refer to separate syntax fragment OP1.

The following words and symbols are used next to the dotted decimal numbers:
- ? means an optional syntax element. A dotted decimal number followed by the ? symbol indicates that all the syntax elements with a corresponding dotted decimal number, and any subordinate syntax elements, are optional. If there is only one syntax element with a dotted decimal number, the ? symbol is displayed on the same line as the syntax element, (for example 5? NOTIFY). If there is more than one syntax element with a dotted decimal number, the ? symbol is displayed on a line by itself, followed by the syntax elements that are optional. For example, if you hear the lines 5 ?, 5 NOTIFY, and 5 UPDATE, you know that syntax elements NOTIFY and UPDATE are optional; that is, you can choose one or none of them. The ? symbol is equivalent to a bypass line in a railroad diagram.
- ! means a default syntax element. A dotted decimal number followed by the ! symbol and a syntax element indicates that the syntax element is the default option for all syntax elements that share the same dotted decimal number. Only one of the syntax elements that share the same dotted decimal number can specify a ! symbol. For example, if you hear the lines 2? FILE, 2.1! (KEEP), and 2.1 (DELETE), you know that (KEEP) is the default option for the FILE keyword. In this example, if you include the FILE keyword but do not specify an option, default option KEEP will be applied. A default option also applies to the next higher dotted decimal number. In this example, if the FILE keyword is omitted, default FILE(KEEP) is used. However, if you hear the lines 2? FILE, 2.1, 2.1.1!
(KEEP), and 2.1.1 (DELETE), the default option KEEP only applies to the next higher dotted decimal number, 2.1 (which does not have an associated keyword), and does not apply to 2? FILE. Nothing is used if the keyword FILE is omitted.

- * means a syntax element that can be repeated 0 or more times. A dotted decimal number followed by the * symbol indicates that this syntax element can be used zero or more times; that is, it is optional and can be repeated. For example, if you hear the line 5.1* data area, you know that you can include one data area, more than one data area, or no data area. If you hear the lines 3*, 3 HOST, and 3 STATE, you know that you can include HOST, STATE, both together, or nothing.

Note:
1. If a dotted decimal number has an asterisk (*) next to it and there is only one item with that dotted decimal number, you can repeat that same item more than once.
2. If a dotted decimal number has an asterisk next to it and several items have that dotted decimal number, you can use more than one item from the list, but you cannot use the items more than once each. In the previous example, you could write HOST STATE, but you could not write HOST HOST.
3. The * symbol is equivalent to a loop-back line in a railroad syntax diagram.

- + means a syntax element that must be included one or more times. A dotted decimal number followed by the + symbol indicates that this syntax element must be included one or more times; that is, it must be included at least once and can be repeated. For example, if you hear the line 6.1+ data area, you must include at least one data area. If you hear the lines 2+, 2 HOST, and 2 STATE, you know that you must include HOST, STATE, or both. Similar to the * symbol, the + symbol can only repeat a particular item if it is the only item with that dotted decimal number. The + symbol, like the * symbol, is equivalent to a loop-back line in a railroad syntax diagram.
Notices

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

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

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

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

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

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

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

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.
IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

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:

Site Counsel
IBM Corporation
2455 South Road
Poughkeepsie, NY 12601-5400
USA

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

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

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

All statements regarding IBM’s future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

COPYRIGHT LICENSE:

This information might contain sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Policy for unsupported hardware

Various z/OS elements, such as DFSMS, HCD, JES2, JES3, and MVS, contain code that supports specific hardware servers or devices. In some cases, this device-related element support remains in the product even after the hardware devices pass their announced End of Service date. z/OS may continue to service element code; however, it will not provide service related to unsupported hardware devices. Software problems related to these devices will not be accepted
for service, and current service activity will cease if a problem is determined to be associated with out-of-support devices. In such cases, fixes will not be issued.

Minimum supported hardware

The minimum supported hardware for z/OS releases identified in z/OS announcements can subsequently change when service for particular servers or devices is withdrawn. Likewise, the levels of other software products supported on a particular release of z/OS are subject to the service support lifecycle of those products. Therefore, z/OS and its product publications (for example, panels, samples, messages, and product documentation) can include references to hardware and software that is no longer supported.

- For information about software support lifecycle, see: IBM Lifecycle Support for z/OS (http://www.ibm.com/software/support/systemsz/lifecycle/)
- For information about currently-supported IBM hardware, contact your IBM representative.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available at Copyright and Trademark information (http://www.ibm.com/legal/copytrade.shtml)
Index

Numerics
1403 printer
UCS images 457
28-bit cylinder address
perform calculations on 317
perform conversions on 317
31-bit IDAW (indirect addressing word) 177
3203 Model 5 printer
JES support 456
UCS images 457
3211 printer
indexing feature 474, 476
JES support 476
UCS images 457
3262 Model 5 printer
alias names 462
default FCB image 471
print bands 465
UCS image table 462, 465
3480/3490 Magnetic Tape Subsystem
messages, displaying 325
MSGDISP macro 325
tape recording technique 206
3800 Printing Subsystem
standard FCB image 471
4245 printer
alias names 462
default FCB image 471
UCS image table 462
UCS image table, contents 463
4248 printer
alias names 462
default FCB image 471
image table, contents 464
UCS image table 462
6262 Model 14 printer
alias names 462
default FCB image 471
UCS image table 462, 465
64-bit IDAW (indirect addressing word) 177

AMCAP
DEVTYPE macro 271
AMODE value
requirement to match UCB address 77
APF (authorized program facility)
EXCP 173, 188
EXCPVR 186
VTOC operations 143
CVAFDIR BRANCH 77
CVAFDIR DEB 75
CVAFDIR IOAREA 75
CVAFDSM BRANCH 97
CVAFDSM DEB 95
CVAFFILT BRANCH 101
CVAFFILT DEB 100
CVAFFILT IOAREA 101
CVAFFSEQ BRANCH 123
CVAFFSEQ DEB 120
CVAFFSEQ IOAREA 123
API 341
appendages
ABE (abnormal end) 229
CHE (channel end) 231
description 222
entry points 223, 224
EOE (end-of-extent) 228
installing 224
list, authorized 224, 229
naming convention 224
PCl (program controlled interruption) 227
PGFX (page fix) 226
programming restrictions 223
returns 223, 224
SIO (start I/O) 225, 227
SYS1.PARMLIB listing 224
work areas available 224
assistive technologies 495
authorization
RACE, DASDVOL 143, 144, 146
BPX1PCT (continued)
DisplayBufferLimits 412
DisplayFSStats, using
using 414
DisplayGlobalStats, using 413
ExtendFS, using 414
BPXPRMxx member 410
buffer
data transfer rate with EXCP 202
installing 202
lists
creating 60
entries 61
format 62
function 60
header format 60
reading DSCBs 102
releasing 65, 78
operating system overhead 202
performance considerations with
BUFNO 202
releasing 65, 78
BWO (backup-while-open) facility
indicator bits 347
C
callable service, IECTRKAD 359
callable services
data set attribute retrieval 345
data set backup-while-open support 347
description 341
DFSMSdfp share attribute retrieval 353
invoking 342
nonreentrant program 341
return and reason codes 354
system level determination 342
CAMLST macro
CAT(BX) operand 163
EADSCB operand 36
format 36
NUMBERDSCB operand 36
RECAT operand 166
RENAME operand 150, 155
SCRATCH operand 145, 150
SEARCH operand 36, 38
SEEK operand 41
UNCAT operand 164
cartridge tape
high-speed positioning 301
catalog
entry
non-VSAM data set 163
reading 157
information retrieval
alias 161
data set name 158
GDS name 159
maintenance 158
catalog (continued)
management
information retrieval 157
macro functions 157
non-VSAM data sets
example 164
macro specifications 163
return codes 167
CATALOG macro
CAT(BX) operand 163
RECAT operand 166
UNCAT operand 164
CCW (channel command word) 187
CDRA 341, 362
cfltm macro
using 56
chaining check 229
ChangeBufferLimits command 413
channel
control check 229
data check 229
channel error
channel program ending status 192
channel program
channel error 192
invalid ending status 192
channel programs
appendages 222
command chaining 173, 175
completion codes 184, 189
control information 185
data set characteristics 184
EXCP
initiation and execution 185, 189
related 190
translating virtual addresses to
central storage addresses 232
problem program
communication 184
executing 170
processing requirements 184
real storage 186
seek address 214
translation
V=R address space 226
V=V address space 187
XDAP 240, 244
CHE appendage
CENDA operand 200
description 231
CLOSE macro
EXCP 195
XDAP 241
command retry 188
communication vector table (CVT) 232, 237
completion codes xi
EXCP I/O 220
XDAP I/O 244
confights command 413
control block
data facilities area (DFA) 449
DCBE (data control block
extension) 207
DEB-EXCP(VR) 443
DFA 449
EXCP 207
control block (continued)
DCB 184, 196, 207
DCBE 185
DEB 185, 222, 443
ECB 184, 220
IEDB 184, 209
IOB 184, 210, 211
IOBE 184, 216
fields used with EXCP 207
fields-EXCP(VR) 443
purged I/O restore list 307
XDAP 244
control password 252
conversion
actual track address to relative track
address
register usage 234
return codes 235
relative track address to actual track
address
procedure 232
register usage 232
return codes 234
sector value, RPS devices 236, 245
virtual channel program to real
channel program 186
VTOC 143
count information
programming interface for 421
CPIO command 409
CSW (channel status word) 149, 188
CVAF (common VTOC access facility)
addressing mode 56
buffer 78
freeing 79
filter service
control blocks required 68
invoking 100
reading sets of DSCBs 67
macros
coding 73
uses and format 73
VT0C access 56, 73
parameter list xi
recommendation 121
return codes
PARTREL 47
RENAME 154
SCRATCH 149
serialization 56
status codes 149, 154
tracing calls 141
volume identification 57
CVAFDSM macro
coding 74
examples 81
parameters 75, 81
return codes 81
using 63
CVAFDIR macro
coding 74
examples 81
parameters 75, 81
return codes 81
using 63
CVAFFILT macro
coding block address resolution 101
DSCBs, reading 67
CVAFFILT macro (continued)
entry 104
example 83, 106
filter criteria list
entry format 70
header format 69
format 101
forms 104
invocation sequences 72, 73
parameters 102, 105
partially-qualified names,
examples 106
restriction 101
RESUME capability 68
return codes 105
RLSE function 73
tip 103
using 56, 100
CVAFSEQ macro
dscb or dsn access 65
examples 129
format 120
parameters 121, 125
return codes 125
using 120
CVAFSTST macro
format 139
return codes 139
using 56, 138
CVFCTN field of CVPL 60
CVPL (CVAF parameter list)
creation 58
format 58
function 58, 60
initializing 100
mapping 100
CVSTAT
codes 47, 140
recommendation 105
D
DADSM (direct access device space
management) 43
allocate routine 16
return codes 54
CAMLST macro
EADSCB operand 39
format 39
NUMBERDSCB operand 39
reading by absolute device
address 39
SEEK operand 39
definition 1
non-VSAM data sets, deleting 145
OBTAIN routine
EADSCB operand 39
NUMBERDSCB operand 39
reading by absolute device
address 39
reading by data set name 36
return codes 38, 41
RENAME macro
return codes 154
status codes 155
return codes 32
DADSM (direct access device space management) (continued)
SCRA.TCH macro
return codes 149
status codes 150
DASD (direct access storage device)
block size 202
buffer allocation 202
data set, creating 48
data transfer rate 202
password protected data sets 250
reading and writing to, with XDAP macro 239, 244
releasing space 16
track capacity calculations 307
volume
fragmentation information 20
initializing 144
restoring from tape 144
space allocation 19
space information 20, 36
VTOC status 20
DASD rotational positioning sensing 188
DASDVOL
RACF, class 143, 144, 146
data
areas, fixing with EXCPVR 186
class
name 345
control block xi
data control block extension (DCBE) used with EXCP 207
Data Extent Block (see DEB) 443
data facilities area (DFA)
control block 449
data set
allocation
absolute 49
movable 49
attribute retrieval 345
backup 347
control block xi
defining 2
deleting
macro instructions 145, 150
shared cylinders 146, 147
stored across devices 146
VIO processed 146
device, assigning 203
DSCB formats 2
expiration date 146
non-VSAM recataloging 166
organization 201
password protection 247, 250
renaming
data set that might be in use 152
password protection 152
VTOC 150
WRITE protection mode indicator 152
repositioning on tape 199
security
access types 248
concatenation 251
space allocation, releasing 42
data set (continued)
space, releasing 20
user label extent 49
DCB (data control block)
address 184
buffer parameters 201
device dependent parameters (EXCP) 203, 207
device-dependent format 201
EXCP 184, 207
fields 196
format after OPEN (EXCP) 196
initializing 207
macro instruction
XDAP 240
OPEN installation exit 202
parameters
device dependent 203
EXCP 196
foundation block 199
restoring 195
DCBD mapping macro for EXCP 207
DCBE (data control block extension) 184, 207
DCBFAD field 205
DCBOFLGS field 194, 196
DCBTRBAL field 205
DDR (dynamic device reconfiguration) repositioning tape data sets 199
DEB (data extent block)
EXCP 185
fields 222
layout (EXCP and EXCPVR) 443
obtaining 57
validating with DEBCHK 259, 263
DEBCHK macro
functions 263
list form 263
register contents 262
return codes 263
specification 260, 263
DELETE
as functional replacement 363
DELETE command 409
DEQ macro, tape volume demount facility 299
DESERV Exit
DELETE 363
GET 363
global exit 365
interaction between task and global exits 365
PUT 363
RENAME 363
SVC Screening 363
SVCPUDPTE 363
task level exit 364
UPDATE 363
determining DFSMSdfp release level 342
determining release level 321
DEVD
DCB operands 206
parameters 203
device
characteristics table entry (DCTE) 311
characteristics, I/O 263
dependent parameters 203
end error recovery procedures 189
DEVTYPE macro
AMCAP 271
examples 278
execute form 267
INFO form 270
INFOLIST type 265
list form 265, 269
output 276, 277
return, reason codes 278
RPS devices 266
specification 264, 265
UCBLIST type 265
DFA
fields 449
mapping of 321
DFSMS level determination 321
DFSMSdfp callable services
data set
attribute retrieval 345
backup-while-open support 347
description 341
DFSMSdfp share attribute retrieval 353
invoking 342
reason codes 354
return codes 354
system attribute call 342
system level determination 342
DFSMSdfp level determination 321, 342
DFSMSdfp share attribute retrieval 353
DFSMSsds
dump utility (restore) 410
DisplayBufferLimits command 412
DisplayFSStats command 414
DisplayGlobalStats command 413
DSCB (data set control block) 4, 14
absolute track address 36
access
direct 63
indexed 66
physical sequential 66
sequential 65
buffer generation 81
chains 72, 100
format sequence 2
format-1 5
format-7 14
format-8 5
format-9 15
mapping 81
nonindexed VTOC 14
qualified data set name 100
reading
buffer list 102
by absolute device address 39
by address 74
by data set name 36, 74
directly by absolute device address 63
directly by DSCB location 64
sequentially 65, 128
sets, CVAF filter service 67
Index 505
F

FCB (forms control buffer) image

standard image STD1 471
standard image STD2 471

EXCP (execute channel program)

control blocks (continued)

DEB 185, 222
ECB 184, 220
error recovery 212
JOB 184, 210, 211
DCBE 207
device-end error recovery procedures 189
EOE appendage 228
I/O error handling 190
indirect addressing word requirements 177
initiation and execution 185, 189
interface parameters 200
macro instruction 196
macros
CLOSE 195
EOV 194
format 185
OPEN 172, 173
multivolume data set requirement 172
overhead 202
PCI appendage 227
PDSE 169
problem programs 170
programming 169
considerations 188
real channel programs 186
storage 186
return codes 149
SIO appendage 225
status information 220
translation by system 226
EXCPVVR
DEB layout 443
executing your own channel programs 169
EXCPV VR 177
EXCPV VR macro
format 186
using 186
expiration date, overriding 146, 147
extended format data set cannot be used as PASSWORD data set 249
ExtendFS command 414
extent area
ICVEDT02 mapping macro 62
exents
allocating 49
available 20
control information 185
end of, appendage 200
free space 14
tracking 4
VSAM data space 11

DSCB (data set control block) (continued)

retrieving 102
VSAM data space extents 11
VTOC contents 11
VTOC types 2
writing
by address 74
by data set name 74
directly by data set name 63
directly by DSCB location 64
DSN order, accessing DSNs, DSCBs 66
DSSIZE value 51
DSTYPE data set attribute 345
DUMP command 409
dynamic device reconfiguration 199

E

EADSCB operand

on CAMLST SEARCH macro 36
on CAMLST SEEK macro 39
on OBTAIN macro 36, 39
EAV
migration assistance tracker 479
ECB (event control block)
completion code 154
completion codes 149
EXCP 184, 220
XDAP 244
end-of-data-set condition 241
routine 201
EOE (end-of-extent) appendage
description 228
EOE A operand 200
EOV (end-of-volume)
erasing data 145
EXCP 194
macro 194
end-of-data-set routine, user 194
EXCP 194
XDAP 241
magnetic tape 194
erasing sensitive data 145
error
CVAF VTOC index 139
I/O recovery procedures (EXCP) 190
routines, ignoring 201
VTOC or index processing 140
recovery from system or user errors 140
event control block xi
examples
DEVTYPE macro 278
exception
printers 467
exception conditions 201
EXCP (execute channel program)
ABE appendage 229
appendages 222
CHE appendage 231
command chaining 173, 175
completion processing 189
data set control blocks 207
DCB 184, 196, 207
DCBE 185

G

GDS (generation data set)
retrieving by name 159
generation
name
absolute 159
relative 159
number 159
GET
DESERV 363
global exit (DESERV) 365
GTF trace of CVAF processing 141

H

HFS
planning 405
HFS data set 2, 7, 346
description 405
file system activity 408
managing 407
SMS-managed 406
HFS file system
deleting 409
HFS files
backing up 409
migrating 409
recovering 410
transporting 408
high-speed cartridge tape
positioning 301

I

I/O
device characteristics 263
efficiency, improving 186
purged restore list 307
I/O (continued)
requests
  purging 304
  reprocessing 307
  restoring 304
ICVEDT02 mapping macro 62
IDAL (indirect addressing list) 177
IDAW (indirect addressing word) 177
IDCAMS
  programming interfaces for 421
IEAAPP00 224
IEBUPDTE program
  SYS1.PARMLIB appendage
  listing 224
IEC301A message 146, 152
IEC502E message 146
IEC606I message 139
IEC614I message 149, 154
IECDPPL macro 305
IECPDSCB macro 49
IECSDSL1 macro 4, 81
IECTRKAD callable service 359
IEDB (input/output error data block)
  EXCP 184
  status information 191
  VIO considerations 192
IEEE POSIX standard 2
IEHLIST program 141
IEWPMAR macro 389, 403
IEZDEB
  fields-EXCP(VR) 443
IGGUCSIT macro 466
IGW023A message 407
IGW023I message 407
IGWABWO call statement
  format 347
  return and reason codes 354
  using 350
IGWARLS call statement
  format 355
  return codes 357
IGWASMS call statement
  format 346
  return and reason codes 354
IGWASYS call statement
  format 342
  return and reason codes 354
IGWDES macro 367, 403
IGWLSHR call statement
  format 353
  return and reason codes 354
IGWSMDE macro 389, 403
IHAAR macro 295
IHAARL macro 294
IHADFA
  control block 449
  fields 449
  macro 321
IHADVA macro 263
IHAFDS macro 372
image
  alias name 457
  identifier 473
  length 458
  library
  adding FCB image 473
  IMAGELIB macro 475
image (continued)
library (continued)
  maintaining 455, 476
  modifying an FCB image 475
  printer information 466
table
  entries, adding or modifying 466
  object module 466
  verification 458, 470
image tables
  tip 465
IMGLIB macro 475
IMMEDIATE option 410
indexed VTOC xi
  buffer disposition 78
  contents 11
  conversion 143
  DSCB
    access 66
    reading 74
    writing 74
    initializing 143
    listings 141
    modifying 74
    password protection 144
    protecting 143
    records 74
    structure 19
    system
      error 140
      support 138
      unauthorized user 140
      volume
        restoring 144
        updating 144
      volume update 144
      indexing feature for 3211 474, 476
      indirect
        addressing word (IDAW) 177
INFO form
  DEVTYPE macro 270
INFOLIST parameter
  DEVTYPE macro 265
intercept condition 229
interception handling procedures 190
IOAREA 76
JOB (input/output block)
  EXCP 184
  fields used with EXCP 210, 211
  PURGE macro 304
  sense bytes 149, 154
 XDAP 244
  XDAP fields 244
JOBE (input/output block common extension)
  EXCP 184
  requesting extended error information 191
JOBEIOT bit 219
IPI volume 249
J
JES (job entry subsystem)
  processing printed output 462
JFCB (job file control block) xi
  macros used with
    IHAARL 294, 297
    OPEN 302
    RDIJFCB 287, 293
  modifying
    functions 284, 304
    precautions 285
    type 07 exit list entry 290
K
keyboard
  navigation 495
  PF keys 495
  shortcut keys 495
KEYLEN operand 206
L
labels, standard 194
  large block interface
    DEVTYPE macro 271
  large format data set
    213-14 ABEND 207
    BLOCKTOKENIZE parameter 207
    cannot be used as PASSWORD data set 249
level
  DFSMSdfp or DFSMS 321, 342
library
  callable services 341
  FCB images 475
  maintenance 475
  printer control information 455
  SYS1.IMAGELIB data set 455
list form, DEVTYPE macro 269
LOCATE macro
  alias name 161
  data set name 158
  generation name 159
  retrieving catalog information 158
  return codes 162
locate-record CCW (EXCP) 187
LSPACE macro 31
  data return area 35
  description 36
  format 20
  data return area 34
  expanded message return area 34
  message return area 32
  parameter list 30
  message return area 35

M
macro
  data management
    CATALOG 163
    LOCATE 157, 162
    MSGDISP 325, 339
macro, mapping
  ICVEDE02 62
macros system
  DEVTYPE macro 267
macros, data management
CAMLST
  RENAME operand  150, 155
SCRATCH operand  145, 150
SEARCH operand  36, 38
SEEK operand  39, 41
CLOSE
  EXCP 195
  XDAP 241
CVAF VTOC access  56, 73
CVAFDIR  63, 74
CVAFDSM  94
CVAFFILT  67, 100
CVAFSEQ  65, 120
CVAFTST  138
DCB  207
  EXCP 196
  XDAP 240
DEBCHK  260, 263
DEVTYPE  263, 280
EOV
  EXCP 194
  XDAP 241
EXCP  185, 196, 207
EXCPVR  186
LSPACE  20, 36
OBTAIN  36, 41
OPEN
  EXCP 172, 173
    modified JFCB 302
  XDAP 240
PARTREL  41, 47
PROTECT  252, 258
PURGE  304
RDJFCB  284, 293
REALLOC  48, 54
RENAME  150, 155
RESTORE  307
SCRATCH  145
TRKCALC  307, 317
using XDAP 240, 241
VTOC access
  DADSM 19, 54
  XDAP 239, 244
management class name 345
mapping macros
IECSDSL1  81
IHADFA  321
messages
CVAF VTOC index error  139
displaying  325
displaying on 3480/3490 options 336
parameters  325
ready  330
resetting display  334
verify volume  328
volume demount  331
volume mounting  326
IECSDSL2  299
IGW023A  407
IGW023B  407
LSPACE macro 32
MSGDISP macro 325
migration assistance tracker
EAV  479
MINAU parameter  51
MKDIR command  406
MODE
  restriction 46
MOUNT command  407, 410
MSG
  restriction 24
MSGDISP macro
  parameters
demount message  331
depect options  336
ready message  330
reset message display  334
verify volume  328
volume mounting  326
return codes  338
multivolume data set
  processing with EXCP 172

N
name
  VTOC index  17
navigation
    keyboard  495
nocapture option 195, 446
non-VSAM
data set
cataloging  163
recataloging  166
uncataloging  164
nonindexed VTOC
  available space 14
  structure 19
NOPWREAD protection-mode
  indicator 252
Notice  499
NOWRITE protection-mode indicator 252
NUMBERDSCB operand
  on CAMLST SEARCH macro 36
  on CAMLST SEEK macro 39
  on OBTAIN macro 36, 39

O
OBTAIN macro 41
  reading by data set name  36
  restriction 20
OPEN macro
  DEQ at demount facility, tape volumes  299
  EXCP
    dummy data set restriction 172
    procedures performed 172
    modified JFCB 302
    TYPE=j
      example 292
      invoking 299
      specification 303
  XDAP  240
  OPENJ (OPEN, TYPE=J) macro 302
OSF XPG/4.2 standard  2
out-of-extent error  229
overhead, system  202
overwriting data  145
P
page
  fix
    appendage 226
    list processing 226
  formatting  474
  paging exceptions 226
  partial DSCB  49, 50
  partially-qualified data set name
    restriction 158
  partitioned data set
    space
      releasing 42, 47
PARTREL macro
  description 42, 47
DSECT form  46
list form  46
return codes  47
password xi
  control 252
counter maintenance  251
data set concatenation  251
deleting protected data set 251
parameter
  ADD record 254
  DELETE record 256
  LIST record 257
  REPLACE record 255
  protection
data sets 247, 258
  JFCB modifications 290
  mode indicator 252
tape data sets 250
  volume switching 251
  VTOC indexes 144
  record 250
renaming protected data set 251
  secondary 252
PASSWORD data set
  creating 250
  maintenance 252
  requirements 249
  restrictions on 249
PAX command  409
PCI (program controlled interruption)
  appendage 200, 227
  CCW modification 188
  description 227
  PCIA operand 200
PDISCALO flag byte  51
PDDBIQTY (number of directory blocks)  49
PDPIQTY (primary space request in tracks) 49
PDSE (partitioned data set extended)
  attribute retrieval 345
  sharing protocol 353
  space
    releasing 42, 47
    releasing unused 20
  TRKCALC macro  307
psfctl
callable service (BPX1PCT)  411
PGFX appendage 226
physical file system 411
POSIX 2
posting completion code in ECB
EXCP I/O 220
printer
bands
3262 Model 5 465
alias names 462
national band IDs 465
special order (RPQ) 465
storing 462
color images 456
character set images 455
control information 455
default image
print position indexing
feature 473
specifying defaults 473
standard FCB images 471
image xi
image library 456
page layout 474
UCS image table 457
printers
exception 467
program check 229
programming interfaces
for count information 421
for subsystem statistics 421
for subsystem status 421
provided by IDCAMS 421
PROTECT macro xi
format 253
functions 252
parameter list
ADD function 254
DELETE function 256
LIST function 257
REPLACE function 255
PASSWORD data set 247, 257
protection-mode indicator 252
requirement 254
return codes 257
protection
check 229
mode indicator 252
VTOC index 143
PURGE macro
parameter list 304
return codes 306
specification 304
purge routines 304
purged I/O restore list 307
PUT
as functional replacement 363
PWREAD protection-mode indicator 252
PWWRITE protection-mode indicator 252
R
RACF (resource access control facility)
DASDVOL class 143, 144, 146
renaming a data set 150
return codes 149
scratching a data set 145
VTOCs and VTOC indexes 143
RACF (resource access naming facility)
return codes 154
RDJFCB macro
DCB exit list entry
type '13' 293
type '07' 290
description 287
description 287
example 288
format
allocation retrieval area 295
allocation retrieval list 294
invoking DEQ at mount 299
retrieving allocation information
DCB exit list entry 294
description 287
example 297
return codes 289
security 290
specification 287
use by authorized programs 291
REALLOC macro
description 48, 54
DSECT form 53
execute form 50
list form 53
return codes 54
reason codes
DEVTYPE macro 278
recommendation
CVAF 121
CVSTAT 105
UCB parameter 76, 122, 139
record
current location 204
REFORMAT REVFtOC command
rebuilding VTOC index with 17
register
contents 222
conversion routines usage
actual to relative 235
relative to actual 232
related requests 190, 213
relative
generation number 159
to actual track address conversion
routine 232
release, determining level of 321
release, determining level of
DFSMsdp 342
RENAME
as functional replacement 363
RENAME macro
description 153
example 153
return codes 154
SMS-managed volumes 151
specification 150
status codes 155
RENAME Parameter List 154
renaming a data set
data set security 152
multivolume considerations 151
renaming a data set that might be in
use 152
SMS considerations 151
specifying volumes affected 151
unrenamable data sets and UNIX
files 151
RENT
requirement 466
requesting different levels of ERP
processing 192
requesting extended error
information 191
requirement
PROTECT macro 254
RENT 466
SMS-managed non-VSAM data
set 147
TEST=YES 334, 335
UCB 22
VTOC 78, 79, 97, 124
Resource Access Control Facility xi
RESTORE macro 307
restore routines 304
restriction
CVAFFLIT 101
MODE 46
MSG 24
OBTAIN 20
partially-qualified data set name 158
UCATDX 165
UCB 46
VERIFY 76
retrieving data set attributes 345
retrieving DFSMsdp share
attributes 353
return codes
CATALOG macro 167
CVAF VTOC index error
message 140
CVAFDIR macro 81
CVAFDSM macro 100
CVAFFILT macro 105
CVAFSEQ macro 125
CVAFSTST macro 139
DADSM allocation 54
DEBCHR macro 263
DEVTYPE macro 278
IGWABWO call statement 354
IGWARLS call statement 357
IGWASMS call statement 354
IGWASYS call statement 354
IGWLISHR call statement 354
LOCATE macro 162
LSPACE macro 31
MSGDISP macro 338
OBTAIN macro
reading from VTOC by absolute
device address 41
reading from VTOC by data set
name 38
PARTREL macro 47
RDJFCB macro 289
REALLOC macro 54
RENAME macro 154
SCRATCH macro 149
track address convert routine 234
track address convert routines 235
TRKCALC macro 316
root file system 406
rotational position sensing 266
RPS (rotational position sensing)
device sector number 236, 244
parameter 266
S

S99ACUCB bit 171
S99DSABA bit 171
S99TI0EX bit 171
SCRATCH command 409
SCRATCH macro
description 145
example 147
return codes 149
status codes 150
scratch parameter list 148
scratching a data set 145, 150
secondary passwords 252
sector address (RPS device) 236, 244
XDAP macro 243
conversion routine 236, 245
number (RPS device) 236, 244
sending comments to IBM xvii
sensitive data, erasing 145
sequential data set space, releasing 42
serializing CVAF requests 56
share attributes, retrieval of DFSMSdfp 353
shortcut keys 495
SIO (start I/O) appendage entry points 226
EXCP (execute channel program) 225
SIOA operand 200
SMF (System Management Facilities) volume information 25
SMS (Storage Management Subsystem) class names 345
data set, deleting 145
indexed VTOC 17
release level 342
status 342
version 342
VTOC deleting data sets 145
SMS-managed non-VSAM data set requirement 147
space allocation releasing unused 42
space map 19
SSGARCL 421
start I/O appendage 226
status codes RENAME macro 155
SCRATCH macro 150
storage class name 345
STORCLASS parameter 406
STOW 363
subsystem statistics programming interface for 421
subsystem status programming interface for 421
SUFFIX, block 273
summary of changes as updated March 2014 xix
Summary of changes xix
SVC Screening 363
SVCUPDATE 363
replacing SVC routine 363
switching volumes, password protection 251
syntax diagram
how to read xiii
SYSI.CSSLIB data set 341
SYSI.IMAGELIB data set adding
FCB image 473, 475
UCS image 457
alias name 462
maintaining 455, 476
modifying FCB image 475, 476
UCS image tables 462
system attribute call 342
macro instructions 259
system level determination 342
system residence volume 249
T

tape
block count field 214
end-of-volume 194
recording density 206
technique 206
reduced error recovery 200
reflective spot 194
repositioning data sets 199
tapemark 194
volumes
DEQ at demount facility 299
password protected data sets 250
tape, cartridge
high-speed positioning 301
TAR command 409
task level exit (DESERV) 364, 365
TEST=YES requirement 334, 335
tip CVAFFILT 103
image tables 465
track balance 310, 311
calculating capacity 307, 317
deleting a record 312
fixed-length records 311
TRKCALC output 315
trademarks 501
TRKADDR macro
ABSTOREL parameter 317
calculating relative track number with 317
COMPARE parameter 318
comparing two track addresses with 318
converting normalized track address into absolute track address with 321
converting relative track number to 28-bit cylinder address with 320
description 317
EXTRACTCYL parameter 318
extracting 28-bit cylinder number with 318
TRKADDR macro (continued)
extracting 4-bit track number with 319
EXTRACTTRK parameter 319
incrementing track address with 319
NEXTTRACK parameter 319
normalize cylinder number with 320
NORMALIZE parameter 320
NORMTOABS parameter 321
RELOABS parameter 320
SETCYL parameter 320
setting cylinder number with 320
TRKCALC macro
description 307, 317
DSECT form 314
examples 316
execute form 313
for extended sequential d.s. 273
inconsistency with PDSE 307
list form 314
output 315
parameter list
creating 314
empty, in-line 314
initialization 313
remote 313
storage definition 313
symbolic expansion 314
return codes 316
standard form 307
TRITCH operand 206
TYPE operand
DEBCHK macro 260
OPEN macro 303
TYPE=j (OPEN macro) 302
U

UCATDX restriction 165
UCB (unit control block) index 214
mapping macro 273
requirement 22
restriction 46
track address 311
UCB parameter recommendation 76, 122, 139
requirement to match AMODE value 77
UCBLIST parameter
DEVTYPE macro 265
UCBRPS bit 236
UCBVRDEB bit 227
UCS (universal character set) image
adding, JCL 459
alias name 462
creating 457
SYSI.IMAGELIB addition 457
SYSI.IMAGELIB, examples 458, 462
verifying 470, 471
image table adding data 468
aliases 466
entry format 462
UCS (universal character set) (continued)
  image table (continued)
    image names 466
    modifying entries 466
    object module 466
    UCS contents 463
    UCS6 contents 464
    system image library
    maintenance 455
VERIFY parameters 471
UE (unit exception) 231
UNCATLG command 409
unit check 229
control block xi
exception 229
universal character set xi
UNIX file system 2
UNMOUNT command 407, 409, 410
UPDATE
  as functional replacement 363
user interface
  ISPF 495
  TSO/E 495
V
V=V address space 187
validating the DEB (DEBCHEK) 259, 263
VCB (volume control block) 158
VERIFY
  restriction 76
  VIER (VTOC index entry record)
    characteristics 18
    description 17
    index search 74
    reading 74
    retaining in virtual storage 78
VIO data sets
  EXCP 192
VIR (VTOC index record) 17
  buffer, releasing 75
  reading 74
  releasing 74
virtual
  IDAW (indirect addressing word) 127
VIXM (VTOC index map)
  bit maps
    allocated DSCBs, VIRs 19
    description 19
  maps of allocated space
    for VIRs 19
volume
  copying (VTOC) 144
  indexed VTOC 144
  label 1
  list 157, 158
    description 145
    renaming 150
  list entry
    scratch status code 149, 154
    secondary status code 149, 154
  nonindexed VTOC 144
  restoring from tape 144
  space allocation 19
  swapping with DDR 199
volume (continued)
  switching
    during EOV 194
    multivolume data sets 251, 292
    password protection 251
  table of contents xi
  tape, protecting 250
  volume serial in CAMLST
    names 157
  volume, IPL 249
VFSM (VTOC pack space map)
  allocated cylinders and tracks 19
  description 19
VSAM (virtual storage access method)
  data space, defining 2
VTOC (volume table of contents)
  access 63
  CVAF macros 73
  DADSM macros 19, 54
  DSCB directly, with CVAFDIR 63
  DSCB sequentially, with
    CVAFSEQ 65
  DSN sequentially, with
    CVAFSEQ 65
  access macros
    CVAFDIR 63, 74
    CVAFDSM 94
    CVAFFILT 67, 100
    CVAFSEQ 65, 120
    CVAFIST 138
    LSPACE 20
    PARTREL 41
    REALLOC 48
    SCRATCH 145
  APF authorization
    contents 11
  conversion 143
  data set renaming 150
  deleting
    non-VSAM data sets 145
    temporary VSAM data sets 145
  description 1
  DSCB
    defining data sets 2
    formats 2
    map xi
    nonindexed 2
    reading 74
    writing 74
  index
    contents 17
    creating 143
    description 17
    entry record (VIER) 18
    error messages 140
    listing 141
    maintaining 145
    map (VIXM) 19
    name 17
    password protection 144
    RACF protection 143
    rebuilding 17
    record 17
    relationship 17
    renaming 17
    structure 17
testing 138
VTOC (volume table of contents) (continued)
  index (continued)
    volumes 144
    initializing 143
    locating 1
    map of DSCBs (VMDS) 17
    modifying 74, 145
    nonindexed 14
    pack space map (VPSM) 19
    protection 143
    reading
      by data set name 36, 41
      DSN order 120
      physical sequential order 120
    records 74
    requirement 78, 79, 97, 124
    structure 19
    system support 138
    updating 57
    using 1
    volume space allocation 19
  VTOC index
    rebuilding 17
W
WLR (wrong-length record) 231
wrong-length indication 229
X
XDAP (execute direct access program)
  channel program 244
  control blocks
    DCB 240
    ECB 244
    IOB 244
  macros
    CLOSE 241
    DCB 240
    EOV 241
    OPEN 240
    specification 241, 244
    requirements 239, 241
Z
z/OS UNIX files
  device characteristics
    information 273
  RENAME macro 153
  zFS data set 2