z/OS

DFSMS: Using Data Sets

SC26-7410-02

z/OS

DFSMS: Using Data Sets

SC26-7410-02

Note

Before using this information and the product it supports, be sure to read the general information under “Notices” on page 613.

Second Edition, March 2002

This edition applies to Version 1 Release 3 of z/OS™ (5694-A01) and to all subsequent releases and modifications until otherwise indicated in new editions.

This edition replaces SC26-7410-01.

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

IBM welcomes your comments. A form for readers’ comments may be provided at the back of this publication, or you may address your comments to the following address:

International Business Machines Corporation
RCF Processing, Department M86/050
5600 Cottle Road
San Jose, CA 95193-0001
United States of America

IBMLINK from US: STARPUBS at SJEVM5
IBMLINK from Canada: STARPUBS at TORIBM
IBM Mail Exchange: USIB3VVD at IBMMAIL
Internet: starpubs@us.ibm.com

If you would like a reply, be sure to include your name, address, telephone number, or FAX number.

Make sure to include the following in your comment or note:

- Title and order number of this book
- Page number or topic related to your comment

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

© Copyright International Business Machines Corporation 1987, 2002. All rights reserved.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Figures . . . xvii

Tables . . . xxi

About This Book . . . xxiii
References to Customer Information Control System (CICS®) . . . xxiii
Organization of This Book . . . xxiv
Required Product Knowledge . . . xxiv
Referenced Publications . . . xxv
Accessing z/OS DFSMS Books on the Internet . . . xxvi
Using LookAt to look up message explanations . . . xxvi
Accessing Licensed Books on the Web . . . xxvii
How to Send Your Comments . . . xxviii

Summary of Changes . . . xxix
Summary of Changes for SC26-7410-02 z/OS Version 1 Release 3 . . . xxix
New Information . . . xxix
Changed Information . . . xxix

Part 1. All Data Sets . . . 1

Chapter 1. Introducing Data Sets . . . 3
System-Managed Data Sets . . . 4
Access Methods . . . 4
Basic Direct Access Method (BDAM) . . . 4
Basic Partitioned Access Method (BPAM) . . . 4
Basic Sequential Access Method (BSAM) . . . 5
Data-in-Virtual (DIV) . . . 5
Indexed Sequential Access Method (ISAM) . . . 5
Object Access Method (OAM) . . . 5
Queued Sequential Access Method (QSAM) . . . 6
Virtual Storage Access Method (VSAM) . . . 6
Access to Files in HFS Data Sets . . . 7
Selection of an Access Method . . . 7
Magnetic Tape and DASD Labels . . . 7
Direct Access Storage Device (DASD) Volumes . . . 8
Track Format . . . 8
Track Overflow . . . 9
VSAM Record Addressing . . . 9
Actual and Relative Addressing with Non-VSAM Access Methods . . . 10
Magnetic Tape Volumes . . . 11
Label Character Coding . . . 11
Data Conversion . . . 11
Identification of Unlabeled Tapes . . . 12
Tape Marks . . . 12
Data Management Macros . . . 12
Data Set Processing . . . 13
Allocating Data Sets . . . 13
Processing Data Sets through Programs . . . 14
Using Access Methods . . . 14
Using Addressing Modes . . . 15
Using Hiperspace™ and Hiperbatch™ . . . 15
Processing VSAM Data Sets . . . 15

© Copyright IBM Corp. 1987, 2002

Processing PDSs and PDSEs 16
Processing Sequential Data Sets and Members of PDSEs and PDSs 17
Processing USS Files with an Access Method 18
Data Set Processing without an Access Method 21
EXCP, EXCPVR, and XDAP Macros 21
USS File Processing 21
Distributed Data Management (DDM) Attributes 22
Virtual I/O for Temporary Data Sets 22
Data Set Names 23
Catalogs and VTOCs 24

Chapter 2. Using the Storage Management Subsystem 27
Using Automatic Class Selection Routines 29
Allocating Data Sets 30
Allocating Data Sets with JCL 31
Allocating System-Managed Data Sets with the Access Method Services ALLOCATE Command 32
Allocating Data Sets with the TSO ALLOCATE Command 34
Allocating Data Sets with Dynamic Allocation 35

Chapter 3. Allocating Space on Direct Access Volumes 37
Specification of Space Requirements 37
Blocks 37
Average Record Length. 38
Tracks or Cylinders 38
Absolute Track 38
Additional Space-Allocation Options 39
Maximum Data Set Size 39
Maximum Size on One Volume 39
Maximum Number of Volumes 39
Maximum VSAM Data Set Size 39
Primary and Secondary Space Allocation without the Guaranteed Space Attribute 40
Multivolume VSAM Data Sets 40
Multivolume Non-VSAM Data Sets. 40
Extended-Format Data Sets 40
Allocation of Data Sets with the Guaranteed Space Attribute 41
Guaranteed Space with DISP=NEW or MOD 41
Guaranteed Space with DISP=OLD or SHR 41
Guaranteed Space with Extended-Format Data Sets 42
Guaranteed Space Example 42
Allocation of Data Sets with the Space Constraint Relief Attributes 42
Extension to Another DASD Volume 43
Examples of Dynamic Volume Count When Defining a Data Set. 43
Examples of Dynamic Volume Count When Allocating an Existing Data Set 44
Multiple Volume Considerations for Sequential Data Sets 45
Additional Information on Space Allocation. 45

Chapter 4. Backing Up and Recovering Data Sets 47
Using REPRO for Backup and Recovery 48
Using EXPORT and IMPORT for Backup and Recovery of VSAM Data Sets 50
Structure of an Exported Data Set 50
EXPORT and IMPORT Commands 50
Writing a Program for Backup and Recovery 51
Using Concurrent Copy for Backup and Recovery 51
Updating a Data Set After Recovery 51

Synchronizing Catalog and VSAM Data Set Information During Recovery . . . 52
Handling an Abnormal Termination. 52
Using VERIFY to Process Improperly Closed Data Sets 53

Chapter 5. Protecting Data Sets 55
z/OS Security Server (RACF) 55
RACF Protection for VSAM Data Sets 55
Generic and Discrete Profiles for VSAM Data Sets. 56
RACF Protection for Non-VSAM Data Sets 56
Data Set Password Protection 57
Passwords for VSAM Data Sets 57
Passwords for Non-VSAM Data Sets 61
User-Security-Verification Routine 62
Erasure of Residual Data 62
Erasing DASD Data 62
Erasing Tape Data 63
Authorized Program Facility and Access Method Services 64
Access Method Services Cryptographic Option 65
Data Enciphering and Deciphering. 66
Key Management 68
Requirements for REPRO ENCIPHER and DECIPHER 70

Chapter 6. Using Hierarchical File System Data Sets 71
Creating Hierarchical File System Data Sets 72
Creating Additional Directories 72

Part 2. VSAM Data Sets 73

Chapter 7. Organizing VSAM Data Sets 75
VSAM Data Formats 75
Data Set Size 75
Control Intervals 76
Control Information Fields 76
Compressed Control Information Field 78
Control Areas 78
Spanned Records 78
VSAM Data Striping 80
Layering Concept for Data Striping 82
Other Considerations for Data Striping 82
Selection of VSAM Data Set Types 84
Entry-Sequenced Data Sets 85
Simulated VSAM Access to USS files 86
Key-Sequenced Data Sets 88
Linear Data Sets 91
Fixed-Length Relative-Record Data Sets 92
Variable-Length Relative-Record Data Sets 93
Summary of VSAM Data Set Types 94
Extended-Format Data Sets 95
Restrictions on Defining Extended-Format Data Sets 95
Compressed Data. 96
Access to Records in a VSAM Data Set 97
Access to Entry-Sequenced Data Sets 97
Access to Key-Sequenced Data Sets. 97
Access to Linear Data Sets 98
Access to Fixed-Length Relative-Record Data Sets 99
Access to Variable-Length Relative-Record Data Sets 99

Access to Records through Alternate Indexes 100
Data Compression 103

Chapter 8. Defining VSAM Data Sets 105
Using Cluster Names for Data and Index Components 106
Defining a Data Set with Access Method Services 106
Naming a Cluster 106
Specifying Cluster Information 108
Using Access Method Services Parameters 109
Allocating Space for VSAM Data Sets 111
Calculating Space for the Data Component of a KSDS 113
Calculating Space for the Index Component 113
Using ALTER to Modify Attributes of a Component 114
Using ALTER to Rename Data Sets 114
Defining a Data Set with JCL 114
Loading a VSAM Data Set 115
Using REPRO to Load a Data Set 115
Using a Program to Load a Data Set 116
Reusing a VSAM Data Set as a Work File 118
Copying and Merging Data Sets 119
Defining Alternate Indexes 120
Naming an Alternate Index 121
Specifying Alternate Index Information 121
Building an Alternate Index 122
Maintaining Alternate Indexes 123
Defining a Path 124
Defining a Page Space 124
Checking for Problems in Catalogs and Data Sets 126
Listing Catalog Entries 126
Printing the Contents of Data Sets 127
Deleting Data Sets 127

Chapter 9. Defining and Manipulating VSAM Data Sets: Examples 129
Example of Defining a VSAM Data Set 130
Examples of Defining Temporary VSAM Data Sets 132
Example 1: Defining a Temporary VSAM Data Set Using ALLOCATE . . . 132
Example 2: Creating a Temporary Data Set with Default Parameter Values 133
Examples of Defining Alternate Indexes and Paths 133
JCL Statements 133
Commands 134

Chapter 10. Processing VSAM Data Sets 137
Creating an Access Method Control Block 138
Creating an Exit List 138
Opening a Data Set 139
Creating a Request Parameter List 140
Manipulating the Contents of Control Blocks 142
Generating a Control Block 142
Testing the Contents of ACB, EXLST, and RPL Fields 142
Modifying the Contents of an ACB, EXLST, or RPL 143
Displaying the Contents of ACB, EXLST, and RPL Fields 143
Requesting Access to a Data Set. 143
Inserting and Adding Records 144
Retrieving Records 146
Updating Records 149
Deleting Records 149

Deferring and Forcing Buffer Writing . . . 149
Retaining and Positioning Data Buffers . . . 150
Processing Multiple Strings . . . 150
Making Concurrent Requests . . . 151
Using a Path to Access Records . . . 152
Making Asynchronous Requests . . . 153
Ending a Request . . . 153
Closing Data Sets . . . 154
Operating in SRB or Cross-Memory Mode . . . 155
Using VSAM Macros in Programs . . . 156

Chapter 11. Optimizing VSAM Performance . . . 159
Optimizing Control Interval Size . . . 159
Control Interval Size Limitations . . . 159
Data Control Interval Size . . . 161
Index Control Interval Size . . . 163
How VSAM Adjusts Control Interval Size . . . 164
Optimizing Control Area Size . . . 164
Advantages of a Large Control Area Size . . . 165
Disadvantages of a Large Control Area Size . . . 166
Optimizing Free Space Distribution . . . 166
Selecting the Optimal Percentage of Free Space . . . 167
Altering the Free Space Specification When Loading a Data Set . . . 168
Determining I/O Buffer Space for Nonshared Resource . . . 169
Obtaining Buffers Above 16 MB . . . 170
Tuning for System-Managed Buffering . . . 170
Allocating Buffers for Concurrent Data Set Positioning . . . 175
Allocating Buffers for Direct Access . . . 175
Allocating Buffers for Sequential Access . . . 178
Allocating Buffers for a Path . . . 179
Acquiring Buffers . . . 179
Using Index Options . . . 180
Increasing Virtual Storage for Index Set Records . . . 180
Avoiding Control Area Splits . . . 180
Putting the Index and Data on Separate Volumes . . . 181
Obtaining Diagnostic Information . . . 181
Migrating from the Mass Storage System . . . 181
Using Hiperbatch . . . 181

Chapter 12. Processing Control Intervals . . . 183
Access to a Control Interval . . . 184
Structure of Control Information . . . 185
CIDF—Control Interval Definition Field . . . 186
RDF—Record Definition Field . . . 186
User Buffering . . . 190
Improved Control Interval Access . . . 191
Opening an Object for Improved Control Interval Access . . . 191
Processing a Data Set with Improved Control Interval Access . . . 191
Fixing Control Blocks and Buffers in Real Storage . . . 192
Control Blocks in Common (CBIC) Option . . . 192

Chapter 13. Sharing VSAM Data Sets . . . 195
Subtask Sharing . . . 196
Building a Single Control Block Structure . . . 196
Resolving Exclusive Control Conflicts . . . 197
Preventing Deadlock in Exclusive Control of Shared Resources . . . 199

Cross-Region Sharing 201
Cross-Region Share Options 201
Read Integrity During Cross-Region Sharing 202
Write Integrity During Cross-Region Sharing 203
Cross-System Sharing. 204
Control Block Update Facility (CBUF) 205
Considerations for CBUF Processing 206
Checkpoints for Shared Data Sets 207
Techniques of Data Sharing. 208
Cross-Region Sharing 208
Cross-System Sharing. 209
User Access to VSAM Shared Information 210

Chapter 14. Sharing Resources Among VSAM Data Sets 211
Provision of a Resource Pool 211
Building a Resource Pool: BLDVRP 211
Connecting a Data Set to a Resource Pool: OPEN 215
Deleting a Resource Pool Using the DLVRP Macro 215
Management of I/O Buffers for Shared Resources 216
Deferring Write Requests 216
Relating Deferred Requests by Transaction ID 217
Writing Buffers Whose Writing is Deferred: WRTBFR 217
Accessing a Control Interval with Shared Resources 219
Restrictions and Guidelines for Shared Resources 220

Chapter 15. Using VSAM Record-Level Sharing 223
Record-Level Sharing CF Caching 224
CICS Use of VSAM RLS 225
Recoverable and Nonrecoverable Files 227
CICS Transactional Recovery for VSAM Recoverable Files 227
Non-CICS Use of VSAM RLS 228
Read Sharing of Recoverable Data Sets 228
Read Sharing Integrity Across KSDS CI and CA Splits 228
Read/Write Sharing of Nonrecoverable Data Sets. 228
Non-RLS Access to VSAM Data Sets 229
Differences Between RLS Access and Non-RLS Access 229
Share Options. 229
Locking 229
VSAM Options Not Supported by RLS 231
VSAM RLS Request Execution Mode Requirements. 231
VSAM RLS Read Integrity Options 232

Chapter 16. Checking VSAM Key-Sequenced Data Set Clusters for Structural Errors 235
EXAMINE Command 235
Types of Data Sets 235
EXAMINE Users 235
How to Run EXAMINE 236
Deciding to Run INDEXTEST, DATATEST, or Both Tests 236
Skipping DATATEST on Major INDEXTEST Errors 236
Examining a User Catalog 236
Understanding Message Hierarchy 237
Controlling Message Printout 237
Samples of Output from EXAMINE Runs 238
INDEXTEST and DATATEST Tests of an Error-Free Data Set 238
INDEXTEST and DATATEST Tests of a Data Set with a Structural Error 238

INDEXTEST and DATATEST Tests of a Data Set with a Duplicate Key Error 239

Chapter 17. Coding VSAM User-Written Exit Routines 241
Guidelines for Coding Exit Routines 241
Programming Considerations 242
Return to a Main Program 243
EODAD Exit Routine to Process End-of-Data 243
Register Contents 244
Programming Considerations 244
EXCEPTIONEXIT Exit Routine 244
Register Contents 245
Programming Considerations 245
JRNAD Exit Routine to Journalize Transactions 245
Register Contents 246
Programming Considerations 246
LERAD Exit Routine to Analyze Logical Errors 252
Register Contents 253
Programming Considerations 253
RLSWAIT Exit Routine 253
Register Contents 254
Request Environment 254
SYNAD Exit Routine to Analyze Physical Errors 255
Register Contents 255
Programming Considerations 255
Example of a SYNAD User-Written Exit Routine 256
UPAD Exit Routine for User Processing 257
Register Contents 258
Programming Considerations 259
User-Security-Verification Routine 260

Chapter 18. Using 31-Bit Addressing Mode with VSAM 263

Chapter 19. Using Job Control Language for VSAM. 265
Using JCL Statements and Keywords 265
Data Set Name 265
Disposition 265
Creating VSAM Data Sets with JCL 266
Temporary VSAM Data Sets 268
Examples Using JCL to Allocate VSAM Data Sets 269
Retrieving an Existing VSAM Data Set 272
Migration Consideration 272
Keywords Used to Process VSAM Data Sets 272

Chapter 20. Processing Indexes of Key-Sequenced Data Sets. 275
Access to a Key-Sequenced Data Set Index 275
Access to an Index with GETIX and PUTIX 275
Access to the Index Component Alone 275
Prime Index 276
Index Levels 277
Format of an Index Record 279
Header Portion 279
Free Control Interval Entry Portion 281
Index Entry Portion 281
Key Compression 282
Index Update Following a Control Interval Split. 284
Index Entries for a Spanned Record 285

Part 3. Non-VSAM Data Sets 287

Chapter 21. Selecting Record Formats for Non-VSAM Data Sets 289
Format Selection 289
Fixed-Length Record Formats 290
Standard Format 290
Restrictions 291
Variable-Length Record Formats 292
Format-V Records 292
Spanned Format-VS Records (Sequential Access Method) 294
Spanned Format-V Records (Basic Direct Access Method) 296
Undefined-Length Record Format 298
ISO/ANSI Tapes 299
Character Data Conversion 299
Format-F Records 300
Format-D Records 302
ISO/ANSI Format-DS and Format-DBS Records 304
Format-U Records 307
Record Format—Device Type Considerations 307
Using Optional Control Characters 308
Using Direct Access Storage Devices (DASD) 309
Using Magnetic Tape 309
Using a Printer 310
Using a Card Reader and Punch 311
Using a Paper Tape Reader 312

Chapter 22. Specifying and Initializing Data Control Blocks 313
Processing Sequential and Partitioned Data Sets 314
Using OPEN to Prepare a Data Set for Processing 318
Filling in the DCB 319
Specifying the Forms of Macros, Buffering Requirements, and Addresses 321
Coding Processing Methods 321
Selecting Data Set Options 322
DCB and DCBE Parameters 322
Data Set Organization (DSORG) 328
Key Length (KEYLEN). 328
Record Length (LRECL) 328
Record Format (RECFM). 328
Write Validity Check Option (OPTCD=W) 329
DD Statement Parameters 329
Changing and Testing the DCB and DCBE 330
Using the DCBD Macro 331
Changing an Address in the DCB 331
Using the IHADCBE Macro 332
Using CLOSE to End the Processing of a Data Set 332
Issuing the CHECK Macro 332
Closing a Data Set Temporarily 332
Using CLOSE TYPE=T with Sequential Data Sets 333
Releasing Space. 334
Managing Buffer Pools When Closing Data Sets 335
Opening and Closing Data Sets: Considerations 335
Parameter Lists with 31-Bit Addresses 335
Open and Close of Multiple Data Sets at the Same Time 335
Factors to Consider When Allocating Direct Access Data Sets 336
Factors to Consider When Opening and Closing Data Sets 336
Open/Close/EOV Errors 336

Installation Exits . . . 337
Positioning Volumes . . . 338
Releasing Data Sets and Volumes . . . 338
Processing End-of-Volume . . . 338
Positioning During End-of-Volume . . . 339
Forcing End-of-Volume . . . 340
Managing SAM Buffer Space . . . 341
Constructing a Buffer Pool . . . 342
Building a Buffer Pool . . . 343
Building a Buffer Pool and a Record Area . . . 343
Getting a Buffer Pool . . . 344
Constructing a Buffer Pool Automatically . . . 344
Freeing a Buffer Pool . . . 345
Constructing a Buffer Pool: Examples . . . 345
Controlling Buffers . . . 346
Queued Access Method . . . 346
Basic Access Method . . . 347
QSAM in an Application . . . 347
Exchange Buffering . . . 350
Choosing Buffering Techniques and GET/PUT Processing Modes . . . 350
Using Buffering Macros with Queued Access Method . . . 351
RELSE—Release an Input Buffer . . . 351
TRUNC—Truncate an Output Buffer . . . 351
Using Buffering Macros with Basic Access Method . . . 351
GETBUF—Get a Buffer from a Pool. . . . 351
FREEBUF—Return a Buffer to a Pool . . . 352

Chapter 23. Accessing Records . . . 353
Accessing Data with READ and WRITE . . . 353
Using the Data Event Control Block (DECB) . . . 353
Grouping Related Control Blocks in a Paging Environment . . . 353
Using Overlapped I/O with BSAM . . . 353
Reading a Block . . . 355
Writing a Block . . . 356
Ensuring I/O Initiation with the TRUNC Macro . . . 356
Testing Completion of a Read or Write Operation . . . 356
Waiting for Completion of a Read or Write Operation . . . 357
Handling Exceptional Conditions on Tape . . . 358
Accessing Data with GET and PUT . . . 359
GET—Retrieve a Record . . . 359
PUT—Write a Record . . . 359
PUTX—Write an Updated Record . . . 360
PDAB—Parallel Input Processing (QSAM Only) . . . 360
Analyzing I/O Errors . . . 362
SYNADAF—Perform SYNAD Analysis Function . . . 362
SYNADRLS—Release SYNADAF Message and Save Areas . . . 364
Device Support Facilities (ICKDSF): Diagnosing I/O Problems . . . 364

Chapter 24. Sharing Non-VSAM Data Sets . . . 365
PDSEs . . . 368
Direct Data Sets (BDAM). . . . 368
Factors to Consider When Opening and Closing Data Sets . . . 369
Control of Checkpoint Data Sets on Shared DASD Volumes . . . 369
System Use of Search Direct for Input Operations . . . 371

Chapter 25. Spooling and Scheduling Data Sets . . . 373

Job Entry Subsystem 373
SYSIN Data Set 374
SYSOUT Data Set 374

Chapter 26. Processing Sequential Data Sets 377
Creating a Sequential Data Set 377
Retrieving a Sequential Data Set 378
Concatenating Data Sets Sequentially 379
Concatenating Like Data Sets 379
Concatenating Unlike Data Sets 384
Modifying Sequential Data Sets 386
Updating in Place 386
Using Overlapped Operations 386
Extending a Data Set 387
Achieving Device Independence 387
Device-Dependent Macros 389
DCB and DCBE Subparameters 389
Improving Performance for Sequential Data Sets 390
Limitations on Using Chained Scheduling with Non-DASD Data Sets . . . 390
DASD and Tape Performance 391
Determining the Length of a Block when Reading with BSAM, BPAM, or BDAM 393
Writing a Short Format-FB Block with BSAM or BPAM 394
Using Hiperbatch 395
Processing Non-VSAM Extended-Format Data Sets 395
Allocating Compressed-Format Data Sets 397
Opening and Closing Extended Format Data Sets 399
Reading, Writing, and Updating Extended Format Data Sets Using BSAM and QSAM 399
Concatenating Extended-Format Data Sets with Other Data Sets 399
Migrating to Extended-Format Data Sets 400
Accessing USS Files Using BSAM, QSAM, or VSAM 401
Record Processing 401
Restrictions 402
Services and Utilities 402
Sequential Concatenation 403

Chapter 27. Processing a Partitioned Data Set (PDS) 405
Structure of a PDS 405
PDS Directory 406
Allocating Space for a PDS 409
Calculating Space 409
Allocating Space with SPACE and AVGREC. 410
Creating a PDS 410
Creating a PDS Member with BSAM or QSAM. 410
Converting PDSs. 411
Adding Members 411
Processing a Member of a PDS 413
BLDL—Construct a Directory Entry List 413
DESERV. 414
FIND—Position to the Starting Address of a Member 417
STOW—Update the Directory 418
Retrieving a Member of a PDS 419
Modifying a PDS 423
Updating in Place 423
Rewriting a Member 426
Concatenating PDSs 426

Sequential Concatenation 426
Partitioned Concatenation 426
Reading a PDS Directory Sequentially 427

Chapter 28. Processing a Partitioned Data Set Extended (PDSE) 429
Advantages of PDSEs 429
PDSE and PDS Similarities 431
PDSE and PDS Differences. 431
Structure of a PDSE 431
PDSE Logical Block Size. 432
Reuse of Space 432
Directory Structure 433
Relative Track Addresses (TTR) 433
Processing PDSE Records 434
Using BLKSIZE with PDSEs 435
Using KEYLEN with PDSEs. 435
Reblocking PDSE Records 435
Processing Short Blocks 436
Processing SAM Null Segments 437
Allocating Space for a PDSE 437
PDSE Space Considerations 437
Summary of PDSE Storage Requirements 440
Defining a PDSE. 440
Creating a PDSE Member 441
Creating a PDSE Member with BSAM or QSAM 441
Adding or Replacing PDSE Members Serially 442
Adding or Replacing Multiple PDSE Members Concurrently 443
Processing a Member of a PDSE 444
Establishing Connections to Members 444
Using the BLDL Macro to Construct a Directory Entry List 445
Using the BSP Macro to Backspace a Physical Record 446
Using the Directory Entry Services 446
Using the FIND Macro to Position to the Beginning of a Member 453
Using ISITMGD to Determine Whether the Data Set Is System Managed 454
Using the NOTE Macro to Provide Relative Position. 455
Using the POINT Macro to Position to a Block 455
Switching between Members 456
Using the STOW Macro to Update the Directory 457
Retrieving a Member of a PDSE 458
Sharing PDSEs 460
Sharing within a Computer System 461
Sharing Violations 461
Multiple System Sharing of PDSEs 462
Modifying a Member of a PDSE 464
Updating in Place 464
Extending a Member 464
Deleting a Member 465
Renaming a Member 465
Reading a PDSE Directory 466
Concatenating PDSEs. 466
Sequential Concatenation 466
Partitioned Concatenation 466
Converting PDSs to PDSEs and Back 467
PDSE to PDS Conversion 468
Restrictions on Converting PDSEs 468
Improving Performance 468

Recovering Space in Fragmented PDSEs 468

Chapter 29. Processing Generation Data Groups 469
Absolute Generation and Version Numbers 470
Relative Generation Number 471
Programming Considerations for Multiple Step Job 471
Naming Generation Data Groups for ISO/ANSI Version 3 or Version 4 Labels 472
Creating a New Generation 473
Allocating a Generation Data Set 473
Passing a Generation Data Set 476
Rolling In a Generation Data Set 476
Controlling Expiration of a Rolled-Off Generation Data Set 477
Creating an ISAM Data Set as Part of a Generation Data Group 477
Retrieving a Generation Data Set 477
Building a Generation Data Group Index 478

Chapter 30. Using I/O Device Control Macros 479
Using the CNTRL Macro to Control an I/O Device 479
Using the PRTOV Macro to Test for Printer Overflow 480
Using the SETPRT Macro to Set Up the Printer 480
Using the BSP Macro to Backspace a Magnetic Tape or Direct Access Volume 481
Using the NOTE Macro to Return the Relative Address of a Block 481
Using the POINT Macro to Position to a Block 482
Using the SYNCDEV Macro to Synchronize Data 483

Chapter 31. Identifying Non-VSAM User-Written Exit Routines 485
General Guidance 485
Programming Considerations 486
Status Information Following an Input/Output Operation 486
EODAD End-of-Data-Set Exit Routine 493
Register Contents 493
Programming Considerations 493
SYNAD Synchronous Error Routine Exit 494
Register Contents 497
Programming Considerations 499
DCB Exit List 501
Register Contents for Exits from EXLST 503
Serialization 503
Allocation Retrieval List 504
Programming Conventions 504
Restrictions 504
DCB ABEND Exit 505
Recovery Requirements 507
DCB Abend Installation Exit 508
DCB OPEN Exit 509
Calls to DCB OPEN Exit for Sequential Concatenation 509
Installation DCB OPEN Exit 509
Defer Nonstandard Input Trailer Label Exit List Entry 510
Block Count Unequal Exit 510
EOV Exit for Sequential Data Sets 511
FCB Image Exit 511
JFCB Exit 513
JFCBE Exit 513
Open/Close/EOV Standard User Label Exit 514
Open/EOV Nonspecific Tape Volume Mount Exit 518
Open/EOV Volume Security and Verification Exit 521

QSAM Parallel Input Exit . . . 523
User Totaling for BSAM and QSAM . . . 523

Part 4. Appendixes . . . 525

Appendix A. Using Direct Access Labels . . . 527
Direct Access Storage Device Architecture . . . 527
Volume Label Group . . . 528
Data Set Control Block (DSCB) . . . 530
User Label Groups . . . 530

Appendix B. Using the Double-Byte Character Set (DBCS) . . . 533
DBCS Character Support . . . 533
Record Length When Using DBCS Characters . . . 533

Appendix C. Processing Direct Data Sets . . . 535
Using the Basic Direct Access Method (BDAM) . . . 535
Processing a Direct Data Set Sequentially . . . 536
Organizing a Direct Data Set . . . 536
By Range of Keys . . . 536
By Number of Records . . . 536
With Indirect Addressing . . . 537
Creating a Direct Data Set . . . 537
Restrictions in Creating a Direct Data Set Using QSAM . . . 537
With Direct Addressing with Keys. . . . 537
With BDAM to Allocate a VIO Data Set . . . 538
Referring to a Record . . . 539
Record Addressing . . . 539
Extended Search . . . 539
Exclusive Control for Updating. . . . 540
Feedback Option. . . . 540
Adding or Updating Records . . . 540
Format-F with Keys. . . . 540
Format-F without Keys . . . 541
Format-V or Format-U with Keys . . . 541
Format-V or Format-U without Keys. . . . 541
Tape-to-Disk Add—Direct Data Set . . . 542
Tape-to-Disk Update—Direct Data Set . . . 543
With User Labels . . . 543
Sharing DCBs. . . . 544

Appendix D. Processing Indexed Sequential Data Sets . . . 545
Using the Basic Indexed Sequential Access Method (BISAM) . . . 545
Using the Queued Indexed Sequential Access Method (QISAM) . . . 545
Processing . . . 546
Data Set Organization. . . . 546
Prime Area . . . 548
Index Areas . . . 548
Overflow Areas . . . 550
Creating an ISAM Data Set . . . 550
One-Step Method . . . 550
Full-Track-Index Write Option . . . 551
Multiple-Step Method . . . 552
Resume Load . . . 553
Allocating Space . . . 553
Prime Data Area . . . 555

Specifying a Separate Index Area 556
Specifying an Independent Overflow Area 556
Calculating Space Requirements 556
Step 1. Number of Tracks Required 556
Step 2. Overflow Tracks Required 557
Step 3. Index Entries Per Track 557
Step 4. Determine Unused Space 558
Step 5. Calculate Tracks for Prime Data Records 558
Step 6. Cylinders Required 559
Step 7. Space for Cylinder Indexes and Track Indexes 559
Step 8. Space for Master Indexes 559
Summary of Indexed Sequential Space Requirements Calculations 560
Retrieving and Updating 561
Sequential Retrieval and Update 561
Direct Retrieval and Update 562
Adding Records 566
Inserting New Records 567
Adding New Records to the End of a Data Set 568
Maintaining an Indexed Sequential Data Set 569
Buffer Requirements 571
Work Area Requirements 572
Space for the Highest-Level Index 573
Device Control 574
SETL—Specifying Start of Sequential Retrieval 574
ESETL—Ending Sequential Retrieval 577

Appendix E. Using ISAM Programs with VSAM Data Sets 579
How an ISAM Program Can Process a VSAM Data Set 580
Converting an Indexed Sequential Data Set 585
JCL for Processing with the ISAM Interface 586
Restrictions on the Use of the ISAM Interface 588
Example: Converting a Data Set 590
Example: Issuing a SYNADAF Macro 591

Appendix F. Converting Character Sets 593
Coded Character Sets Sorted by CCSID 593
Coded Character Sets Sorted by Default LOCALNAME 596
CCSID Conversion Groups 603
CCSID Decision Tables 605
Tables for Default Conversion Codes 610
Converting from EBCDIC to ASCII 610
Converting from ASCII to EBCDIC 610

Accessibility 611
Using assistive technologies 611
Keyboard navigation of the user interface 611

Notices 613
Programming Interface Information 614
Trademarks 614

Glossary 615

Index 627

Figures

1. DASD Volume Track Formats . . . 9
2. USS Directories and Files in a File System . . . 22
3. REPRO Encipher/Decipher Operations . . . 67
4. VSAM Logical Record Retrieval . . . 75
5. Control Interval Format . . . 76
6. Record Definition Fields of Control Intervals . . . 77
7. Data Set with Nonspanned Records . . . 79
8. Data Set with Spanned Records . . . 79
9. Primary and Secondary Space Allocations for Striped Data Sets . . . 81
10. Control Interval / Control Area . . . 81
11. Layering (Four Stripe Data Set) . . . 82
12. Entry-Sequenced Data Set . . . 85
13. Example of RBAs of an Entry-Sequenced Data Set . . . 85
14. Record of a Key-Sequenced Data Set . . . 88
15. Inserting Records into a Key-Sequenced Data Set . . . 88
16. Inserting a Logical Record into a CI . . . 90
17. Fixed-length Relative-Record Data Set . . . 92
18. Control Interval Size . . . 95
19. Alternate Index Structure for a Key-Sequenced Data Set. . . . 101
20. Alternate Index Structure for an Entry-Sequenced Data Set. . . . 102
21. VSAM Macro Relationships . . . 157
22. Skeleton VSAM Program . . . 158
23. Control Interval Size, Physical Track Size, and Track Capacity . . . 160
24. Determining Free Space . . . 167
25. Scheduling Buffers for Direct Access . . . 177
26. General Format of a Control Interval . . . 185
27. Format of Control Information for Nonspanned Records . . . 188
28. Format of Control Information for Spanned Records . . . 189
29. Exclusive Control Conflict Resolution . . . 198
30. Relationship Between the Base Cluster and the Alternate Index . . . 200
31. VSAM RLS Address/Data Spaces and Requestor Address Spaces . . . 224
32. CICS VSAM Non-RLS Access . . . 226
33. CICS VSAM RLS . . . 226
34. Example of a JRNAD Exit . . . 248
35. Example of a SYNAD Exit Routine . . . 257
36. Relation of Index Entry to Data Control Interval . . . 276
37. Relation of Index Entry to Data Control Interval . . . 277
38. Levels of a Prime Index . . . 278
39. General Format of an Index Record . . . 279
40. Format of the Index Entry Portion of an Index Record . . . 281
41. Format of an Index Record . . . 282
42. Example of Key Compression . . . 284
43. Control Interval Split and Index Update . . . 285
44. Fixed-Length Records . . . 290
45. Nonspanned, Format-V Records. . . . 292
46. Spanned Format-VS Records (Sequential Access Method) . . . 294
47. Spanned Format-V Records for Direct Data Sets . . . 297
48. Undefined-Length Records. . . . 298
49. Fixed-Length Records for ISO/ANSI Tapes . . . 301
50. Nonspanned Format-D Records for ISO/ANSI Tapes As Seen by the Program. . . . 304
51. Spanned Variable-Length (Format-DS) Records for ISO/ANSI Tapes As Seen by the Program 305
52. Reading a Sequential Data Set . . . 316
53. Reentrant—Above the 16 MB Line . . . 317

© Copyright IBM Corp. 1987, 2002

54. Sources and Sequence of Operations for Completing the DCB . . . 319
55. Opening Three Data Sets at the Same Time . . . 322
56. Changing a Field in the DCB . . . 331
57. Closing Three Data Sets at the Same Time . . . 332
58. Record Processed when LEAVE or REREAD is Specified for CLOSE TYPE=T . . . 333
59. Constructing a Buffer Pool from a Static Storage Area. . . . 345
60. Constructing a Buffer Pool Using GETPOOL and FREEPOOL. . . . 346
61. Simple Buffering with MACRF=GL and MACRF=PM . . . 348
62. Simple Buffering with MACRF=GM and MACRF=PM . . . 349
63. Simple Buffering with MACRF=GL and MACRF=PL . . . 349
64. Simple Buffering with MACRF=GL and MACRF=PM-UPDAT Mode . . . 350
65. Parallel Processing of Three Data Sets . . . 361
66. JCL, Macros, and Procedures Required to Share a Data Set Using Multiple DCBs . . . 366
67. Macros and Procedures Required to Share a Data Set Using a Single DCB . . . 367
68. Creating a Sequential Data Set—Move Mode, Simple Buffering . . . 378
69. Retrieving a Sequential Data Set—Locate Mode, Simple Buffering . . . 379
70. Like Concatenation Read through BSAM . . . 384
71. Reissuing a READ or GET for Unlike Concatenated Data Sets . . . 385
72. One Method of Determining the Length of a Record when Using BSAM to Read Undefined-Length or Blocked Records . . . 394
73. A Partitioned Data Set (PDS) . . . 406
74. A PDS Directory Block . . . 406
75. A PDS Directory Entry . . . 407
76. Creating One Member of a PDS. . . . 411
77. Creating Members of a PDS Using STOW . . . 412
78. BLDL List Format . . . 414
79. DESERV GET by NAME_LIST Control Block Structure . . . 415
80. DESERV GET by PDSDE Control Block Structure . . . 416
81. DESERV GET_ALL Control Block Structure . . . 417
82. Retrieving One Member of a PDS . . . 419
83. Retrieving Several Members and Subgroups of a PDS without Overlapping I/O Time and CPU Time . . . 421
84. Reading a Member of a PDS or PDSE using Asynchronous BPAM . . . 422
85. Updating a Member of a PDS . . . 425
86. A Partitioned Data Set Extended (PDSE) . . . 430
87. TTRs for a PDSE Member (Unblocked Records). . . . 433
88. TTRs for Two PDSE Members (LRECL=80, BLKSIZE=800) . . . 434
89. Example of How PDSE Records Are Reblocked . . . 436
90. Example of Reblocking When the Block Size Has Been Changed . . . 436
91. Creating One Member of a PDSE . . . 441
92. Adding PDSE Members Serially . . . 443
93. Replacing Multiple PDSE Members Concurrently . . . 443
94. DESERV GET by NAME_LIST Control Block Structure . . . 448
95. DESERV GET by PDSDE Control Block Structure . . . 449
96. DESERV GET_ALL Control Block Structure . . . 450
97. DESERV GET_NAMES Control Block Structure . . . 451
98. DESERV RELEASE Input Control Block Structure . . . 452
99. DESERV UPDATE. . . . 453
100. ISITMGD Example. . . . 454
101. Using NOTE and FIND to Switch Between Members of a Concatenated PDSE . . . 457
102. STOW INITIALIZE Example . . . 458
103. Retrieving One Member of a PDSE . . . 458
104. Retrieving Several Members of a PDSE or PDS . . . 460
105. OPEN Success/Failure . . . 462
106. OPEN for UPDAT and Positioning to a Member Decision Table . . . 463
107. Status Indicators—BDAM, BPAM, BSAM, and QSAM . . . 492

108. Parameter List Passed to DCB Abend Exit Routine 505
109. Recovery Work Area 507
110. Defining an FCB Image for a 3211 512
111. Parameter List Passed to User Label Exit Routine 514
112. IECOENTE Macro Parameter List 519
113. IECOEVSE Macro Parameter List 522
114. Direct Access Labeling 528
115. Initial Volume Label Format 529
116. User Header and Trailer Labels on DASD or Tape 530
117. Creating a Direct Data Set (Tape-to-Disk) 538
118. Adding Records to a Direct Data Set 542
119. Updating a Direct Data Set 543
120. Indexed Sequential Data Set Organization 548
121. Format of Track Index Entries 549
122. Creating an Indexed Sequential Data Set 552
123. Requests for Indexed Sequential Data Sets 555
124. Sequentially Updating an Indexed Sequential Data Set 562
125. Directly Updating an Indexed Sequential Data Set 565
126. Directly Updating an Indexed Sequential Data Set with Variable-Length Records 567
127. Adding Records to an Indexed Sequential Data Set 569
128. Deleting Records from an Indexed Sequential Data Set 570
129. Use of ISAM Processing Programs. 580
130. CCSID Conversion Group 1 603
131. CCSID Conversion Group 2 603
132. CCSID Conversion Group 3 603
133. CCSID Conversion Group 4 604
134. CCSID Conversion Group 5 604

Tables

1. Data Management Access Methods . . . 13
2. Data Set Activity for Non-System-Managed and System-Managed Data Sets . . . 28
3. Entry-Sequenced Data Set Processing . . . 86
4. Key-Sequenced Data Set Processing . . . 89
5. RRDS Processing . . . 93
6. Variable-Length RRDS Processing . . . 94
7. Comparison of ESDS, KSDS, Fixed-Length RRDS, Variable-Length RRDS, and Linear Data Sets 94
8. Adding Data to Various Types of Output Data Sets . . . 120
9. Effect of RPL Options on Data Buffers and Positioning . . . 150
10. Relationship between SHAREOPTIONS and VSAM Functions . . . 205
11. VSAM User-Written Exit Routines . . . 242
12. Contents of Registers at Entry to EODAD Exit Routine . . . 244
13. Contents of Registers at Entry to EXCEPTIONEXIT Routine . . . 245
14. Contents of Registers at Entry to JRNAD Exit Routine . . . 246
15. Contents of Parameter List Built by VSAM for the JRNAD Exit . . . 250
16. Contents of Registers at Entry to LERAD Exit Routine . . . 253
17. Contents of Registers for RLSWAIT Exit Routine . . . 254
18. Contents of Registers at Entry to SYNAD Exit Routine . . . 255
19. Conditions when UPAD Routines are Taken . . . 258
20. Contents of Registers at Entry to UPAD Exit Routine . . . 258
21. Parameter List Passed to UPAD Routine . . . 259
22. Communication with User-Security-Verification Routine . . . 261
23. 31-Bit Address Keyword Parameters . . . 264
24. Format of the Header of an Index Record . . . 279
25. Segment Control Codes . . . 295
26. Tape Density (DEN) Values . . . 309
27. Optimum and Maximum Block Size Supported . . . 325
28. Rules for Setting Block Sizes for Tape Data Sets or Compressed Format Data Sets . . . 327
29. Buffering Technique and GET/PUT Processing Modes . . . 350
30. PDSE and PDS Differences . . . 431
31. DE Services Function Summary . . . 447
32. DCB Exit Routines . . . 486
33. Data Event Control Block . . . 487
34. Exception Code Bits—BISAM . . . 488
35. Event Control Block . . . 489
36. Exception Code Bits—BDAM . . . 491
37. Contents of Registers at Entry to EODAD Exit Routine . . . 493
38. Exception Code Bits—QISAM . . . 495
39. Register Contents on Entry to SYNAD Routine—BDAM, BPAM, BSAM, and QSAM . . . 497
40. Register Contents on Entry to SYNAD Routine—BISAM . . . 498
41. Register Contents on Entry to SYNAD Routine—QISAM . . . 498
42. DCB Exit List Format and Contents . . . 502
43. Option Mask Byte Settings . . . 506
44. Conditions for Which Recovery Can Be Attempted . . . 507
45. System Response to Block Count Exit Return Code . . . 510
46. System Response to a User Label Exit Routine Return Code . . . 516
47. Saving and Restoring General Registers . . . 520
48. QISAM Error Conditions . . . 581
49. BISAM Error Conditions . . . 582
50. Register Contents for DCB-Specified ISAM SYNAD Routine . . . 583
51. Register Contents for AMP-Specified ISAM SYNAD Routine . . . 583
52. ABEND Codes Issued by the ISAM Interface . . . 584
53. DEB Fields Supported by ISAM Interface . . . 584

© Copyright IBM Corp. 1987, 2002

54. DCB Fields Supported by ISAM Interface 586
55. Output DISP=NEW,OLD. 606
56. Output DISP=MOD (IBM V4 tapes only) 606
57. Input 608

About This Book

This book is intended for use by system and application programmers. This book is intended to help you use access methods to process VSAM data sets, sequential data sets, partitioned data sets (PDSs), partitioned data sets extended (PDSEs), and generation data sets in the DFSMS environment. This book also explains how to use access method services commands, macro instructions, and JCL to process data sets. *z/OS DFSMS Macro Instructions for Data Sets* describes coding macro instructions for data sets.

For information about DFM, see *z/OS DFSMS DFM Guide and Reference*. For information about DFSMS Network File System, see *z/OS Network File System User's Guide*. For information about z/OS UNIX® System Services (z/OS UNIX), see *z/OS UNIX System Services User's Guide*.

This publication, *z/OS DFSMS: Using Data Sets*, shows the following functions only for compatibility. Although IBM® still supports these functions, it does not recommend their use. Where applicable, this publication suggests alternatives to these functions.

BDAM (use VSAM instead)
EXCP and XDAP (use BSAM, BPAM, or QSAM instead)
BISAM (use VSAM instead)
QISAM (use VSAM instead)

The following non-VSAM macros are not recommended:

DCB (BDAM, BISAM, QISAM)
ESETL
FREEDBUF
GET (QISAM)
PUT (QISAM)
READ (BDAM, BISAM)
RELEX
SETL
WRITE (BDAM, BISAM)

The following DASD products are unsupported:

IBM 2305
IBM 3330
IBM 3333
IBM 3340
IBM 3344
IBM 3350
IBM 3375
IBM 3851

The ACQRANGE, CNVTAD, and MNTACQ macros are no longer supported.

References to Customer Information Control System (CICS®)

In this book, the term "CICS" refers to the CICS Transaction Server for OS/390®. Citations are to the CICS Transaction Server for the OS/390 library. CICS/MVS® and CICS/ESA® users should use the corresponding books in those libraries.

© Copyright IBM Corp. 1987, 2002

Organization of This Book

The organization of this book divides information into four major parts:

Part	Page
Part 1 covers general topics for all data sets.	1
Part 2 covers the processing of VSAM data sets.	73
Part 3 covers the processing of non-VSAM data sets.	287
Part 4 covers the following topics: • Using direct access labels • Copying and printing Kanji characters using the double-byte character set • Processing direct data sets • Processing indexed sequential data sets • Using ISAM programs with VSAM data sets • Using character sets for conversion.	527

Required Product Knowledge

To use this book effectively, you should be familiar with the following information and languages:

- IBM support and how it is structured
- Assembler language
- Job control language (JCL)
- Diagnostic techniques

You should also understand access method services commands, catalogs, and storage administration, which the following publications describe.

Topics	Publications
Access method services commands	*z/OS DFSMS Access Method Services* describes the access method services commands used to process Virtual Storage Access Method (VSAM) data sets.
Catalogs	*z/OS DFSMS: Managing Catalogs* describes how to create master and user catalogs.
Storage administration	*z/OS DFSMSdfp Storage Administration Reference* and *z/OS DFSMS: Implementing System-Managed Storage* describe storage administration.

To learn about the effects of local system-managed and non-system-managed data set access by remote distributed file manager (DFM) source servers in a teleprocessing network, see *z/OS DFSMS DFM Guide and Reference*. *z/OS DFSMS DFM Guide and Reference* identifies the supported data set types and discusses considerations for making data sets available for remote access.

Referenced Publications

This book refers to the following publications.

Publication Title	Order Number
Character Data Representation Architecture Reference and Registry	SC09-2190
Device Support Facilities User's Guide and Reference	GC35-0033
MVS Hiperbatch Guide	GC28-1470
MVS/ESA SML: Managing Data	SC26-3124
z/OS DFSMS Access Method Services	SC26-7394
z/OS DFSMS Checkpoint/Restart	SC26-7401
z/OS DFSMS DFM Guide and Reference	SC26-7395
z/OS DFSMS: Implementing System-Managed Storage	SC26-7407
z/OS DFSMS Installation Exits	SC26-7396
z/OS DFSMS Macro Instructions for Data Sets	SC26-7408
z/OS DFSMS: Managing Catalogs	SC26-7409
z/OS DFSMS Program Management	SC27-1130
z/OS DFSMS Migration	GC26-7398
z/OS DFSMS OAM Application Programmer's Reference	SC35-0425
z/OS DFSMS OAM Planning, Installation, and Storage Administration Guide for Object Support	SC35-0426
z/OS DFSMS: Using Magnetic Tapes	SC26-7412
z/OS DFSMS: Using Magnetic Tapes	SC26-7412
z/OS DFSMS: Using the Interactive Storage Management Facility	SC26-7411
z/OS DFSMSdfp Advanced Services	SC26-7400
z/OS DFSMSdfp Diagnosis Reference	GY27-7618
z/OS DFSMSdfp Storage Administration Reference	SC26-7402
z/OS DFSMSdfp Utilities	SC26-7414
z/OS DFSMSdss Storage Administration Guide	SC35-0423
z/OS DFSMSdss Storage Administration Reference	SC35-0424
z/OS DFSMShsm Managing Your Own Data	SC35-0420
z/OS ISPF User's Guide Volume I	SC34-4822
z/OS DFSMSrmm Implementation and Customization Guide	SC26-7405
z/OS MVS Initialization and Tuning Reference	SA22-7592
z/OS MVS JCL Reference	SA22-7597
z/OS MVS JCL User's Guide	SA22-7598
z/OS MVS Planning: Global Resource Serialization	SA22-7600
z/OS MVS Programming: Assembler Services Guide	SA22-7605
z/OS MVS Programming: Assembler Services Reference ABE-HSP	SA22-7606
z/OS MVS Programming: Authorized Assembler Services Guide	SA22-7608
z/OS MVS Programming: Authorized Assembler Services Reference ALE-DYN	SA22-7609
z/OS MVS Programming: Authorized Assembler Services Reference ENF-IXG	SA22-7610

Publication Title	Order Number
z/OS MVS Programming: Authorized Assembler Services Reference LLA-SDU	SA22-7611
z/OS MVS System Codes	SA22-7626
z/OS MVS System Commands	SA22-7627
z/OS MVS System Management Facilities (SMF)	SA22-7630
z/OS MVS System Messages, Vol 7 (IEB-IEE)	SA22-7637
z/OS MVS System Messages, Vol 8 (IEF-IGD)	SA22-7638
z/OS MVS System Messages, Vol 9 (IGF-IWM)	SA22-7639
z/OS MVS System Messages, Vol 10 (IXC-IZP)	SA22-7640
z/OS Network File System User's Guide	SC26-7419
z/OS Security Server RACF Command Language Reference	SA22-7687
z/OS Security Server RACF Security Administrator's Guide	SA22-7683
z/OS TSO/E Command Reference	SA22-7782
z/OS TSO/E Primer	SA22-7787
z/OS TSO/E User's Guide	SA22-7794
z/OS UNIX System Services Planning	GA22-7800
z/OS UNIX System Services User's Guide	SA22-7801

Accessing z/OS DFSMS Books on the Internet

In addition to making softcopy books available on CD-ROM, IBM provides access to unlicensed z/OS softcopy books on the Internet. To find z/OS books on the Internet, first go to the z/OS home page: http://www.ibm.com/servers/eserver/zseries/zos

From this Web site, you can link directly to the z/OS softcopy books by selecting the Library icon. You can also link to IBM Direct to order hardcopy books.

Using LookAt to look up message explanations

LookAt is an online facility that allows you to look up explanations for z/OS messages, system abends, and some codes. Using LookAt to find information is faster than a conventional search because in most cases LookAt goes directly to the message explanation.

You can access LookAt from the Internet at:

`http://www.ibm.com/servers/eserver/zseries/zos/bkserv/lookat/lookat.html`

or from anywhere in z/OS where you can access a TSO command line (for example, TSO prompt, ISPF, z/OS UNIX System Services running OMVS).

To find a message explanation on the Internet, go to the LookAt Web site and simply enter the message identifier (for example, `IAT1836` or `IAT*`). You can select a specific release to narrow your search. You can also download code from the *z/OS Collection*, SK3T-4269 and the LookAt Web site so you can access LookAt from a PalmPilot (Palm VIIx suggested).

To use LookAt as a TSO command, you must have LookAt installed on your host system. You can obtain the LookAt code for TSO from a disk on your *z/OS Collection*, SK3T-4269 or from the LookAt Web site. To obtain the code from the LookAt Web site, do the following:

1. Go to http://www.ibm.com/servers/eserver/zseries/zos/bkserv/lookat/lookat.html.
2. Click the **News** button.
3. Scroll to **Download LookAt Code for TSO and VM.**
4. Click the ftp link, which will take you to a list of operating systems. Select the appropriate operating system. Then select the appropriate release.
5. Find the **lookat.me** file and follow its detailed instructions.

To find a message explanation from a TSO command line, simply enter: **lookat** *message-id*. LookAt will display the message explanation for the message requested.

Note: Some messages have information in more than one book. For example, IEC192I has routing and descriptor codes listed in *z/OS MVS Routing and Descriptor Codes*. For such messages, LookAt prompts you to choose which book to open.

Accessing Licensed Books on the Web

z/OS licensed documentation in PDF format is available on the Internet at the IBM Resource Link Web site at:

`http://www.ibm.com/servers/resourcelink`

Licensed books are available only to customers with a z/OS license. Access to these books requires an IBM Resource Link Web userid and password, and a key code. With your z/OS order you received a memo that includes this key code.

To obtain your IBM Resource Link Web userid and password log on to:

`http://www.ibm.com/servers/resourcelink`

To register for access to the z/OS licensed books:

1. Log on to Resource Link using your Resource Link user ID and password.
2. Click on **User Profiles** located on the left-hand navigation bar.
3. Click on **Access Profile.**
4. Click on **Request Access to Licensed books.**
5. Supply your key code where requested and click on the **Submit** button.

If you supplied the correct key code you will receive confirmation that your request is being processed. After your request is processed, you will receive an e-mail confirmation.

Note: You cannot access the z/OS licensed books unless you have registered for access to them and received an e-mail confirmation informing you that your request has been processed.

To access the licensed books:

1. Log on to Resource Link using your Resource Link userid and password.
2. Click on **Library**.
3. Click on **zSeries**.

4. Click on **Software**.
5. Click on **z/OS**.
6. Access the licensed book by selecting the appropriate element.

How to Send Your Comments

Your feedback is important in helping to provide the most accurate and high-quality information. If you have any comments about this book or about any other DFSMS documentation, you can send them in either of the following ways:

- Send your comments by electronic mail (e-mail) to any of the following addresses:
  - IBMLink™ from US: starpubs@us.ibm.com
  - IBMLink from Canada: STARPUBS at TORIBM
  - IBM Mail Exchange: USIB3VVD at IBMMAIL
  - Internet: starpubs@us.ibm.com

  Be sure to include the name the book, part number, version, product name, and if applicable, the specific location of the text on which you are commenting (for example, a page number or a table number).
- Fill out one of the forms at the back of this book and return it by mail or by giving it to an IBM representative. If the form has been removed, address your comments to IBM Corporation, RCF Processing Department M86/050, 5600 Cottle Road, San Jose, California 95193-0001, U.S.A.

Summary of Changes

This book contains terminology, maintenance, and editorial changes. Technical changes or additions to the text and illustrations are indicated by a vertical line to the left of each change.

You might notice changes in the style and structure of some content in this book—for example, more specific headings for notes, such as Tip and Requirement. The changes are ongoing improvements to the consistency and retrievability of information in DFSMS books.

Summary of Changes for SC26-7410-02 z/OS Version 1 Release 3

This book contains information that was previously presented in *z/OS Version 1 Release 1 DFSMS: Using Data Sets* (SC26-7410-01).

The following sections summarize the changes to that information.

New Information

This edition includes the following new information:

- Data set naming conventions
- Virtual input/output (VIO) limit
- Real addresses greater than 2 GB available for all VSAM data sets
- Caching all or some of the VSAM record level sharing (RLS) data in a coupling facility (CF) cache structure
- VSAM RLS system-managed duplexing rebuild process, and validity checking for a user-managed rebuild or alter process
- Dynamic volume count for space constraint relief when you store data sets on DASD volumes
- A summary of the effects of specifying all extents (ALX) or maximum contiguous extents (MXIG) for virtual input/output (VIO) data sets

Changed Information

The following information changed in this edition:

- Allocation of HFS data sets requiring directory block size
- Requirements for the ENCIPHER and DECIPHER functions of the REPRO command
- Relative byte address (RBA) in an entry-sequenced data set
- Processing techniques for VSAM system-managed buffering
- Calculating the index control interval size for a key-sequenced data set
- Extensions of VSAM data sets when you specify zero as a secondary allocation quantity
- Sharing of non-VSAM data sets
- Use of the BYPASSLLA option with the BLDL list
- Partitioned concatenation of PDSs and PDSEs
- Processing restrictions for PDSEs
- Rules for updating a data set in place
- Conversion of the exclamation point character between EBCDIC and ASCII
- Direct insertion of records into a key-sequenced data set
- Addition of QSAM LRECL value
- Sharing of PDSEs by multiple systems in a sysplex

© Copyright IBM Corp. 1987, 2002

Part 1. All Data Sets

© Copyright IBM Corp. 1987, 2002

Chapter 1. Introducing Data Sets

This chapter covers the following topics.

Topic	Page
System-Managed Data Sets	4
Access Methods	4
Magnetic Tape and DASD Labels	7
Direct Access Storage Device (DASD) Volumes	8
Magnetic Tape Volumes	11
Data Management Macros	12
Data Set Processing	13
Data Set Processing without an Access Method	21
Distributed Data Management (DDM) Attributes	22
Virtual I/O for Temporary Data Sets	22
Data Set Names	23
Catalogs and VTOCs	24

A data set is a collection of logically related data and can be a source program, a library of macros, or a file of data records used by a processing program. Data records are the basic unit of information used by a processing program. By placing your data into volumes of organized data sets, you can save and process the data. You can also print the contents of a data set or display the contents on a terminal.

You can store data on secondary storage devices, such as a direct access storage device (DASD) or magnetic tape volume. The term DASD applies to disks or to simulated equivalents of disks and magnetic drums. A volume is a standard unit of secondary storage. You can store all types of data sets on DASD but only sequential data sets on magnetic tape. Mountable tape volumes can reside in an automated tape library. For information about magnetic-tape volumes, see *z/OS DFSMS: Using Magnetic Tapes*. You can also direct a sequential data set to or from spool, a USS (UNIX system services) file (including NFS), a TSO/E terminal, a unit record device, virtual I/O (VIO), or a dummy data set.

Each block of data on a DASD volume has a distinct location and a unique address, making it possible to find any record without extensive searching. Records can be stored and retrieved either directly or sequentially. DASD volumes are used for storing data and executable programs, including the operating system itself, and for temporary working storage. One DASD volume can be used for many different data sets, and space on it can be reallocated and reused.

Data management is the part of the operating system that organizes, identifies, stores, catalogs, and retrieves all the information (including programs) that your installation uses. Data management does these main tasks:

- Sets aside (allocates) space on DASD volumes
- Automatically retrieves cataloged data sets by name
- Causes magnetic tape volumes to be mounted in the drive
- Establishes a logical connection between the application program and the medium
- Controls access to data
- Transfers data between the application program and the medium.

© Copyright IBM Corp. 1987, 2002

Introducing Data Sets

With distributed file manager target server support, applications running on a processor with distributed file manager (DFM) source server support can create or access certain types of SMS-managed data sets. They can also access certain types of non-SMS-managed data sets on an S/390® processor running DFSMS, with distributed file manager target server support. See *z/OS DFSMS DFM Guide and Reference* for details about the supported data set types and a discussion of considerations in making them available for remote access. Also see “Distributed Data Management (DDM) Attributes” on page 22.

System-Managed Data Sets

The Storage Management Subsystem (SMS) is an operating environment that automates the management of storage. Storage management uses the values provided at allocation time to determine, for example, on which volume to place your data set, and how many tracks to allocate for it. Storage management also manages tape data sets on mountable volumes that reside in an automated tape library. With SMS, users can allocate data sets more easily.

The data sets allocated through SMS are called system-managed data sets or SMS-managed data sets. For information about allocating system-managed data sets, see “Chapter 2. Using the Storage Management Subsystem” on page 27. If you are a storage administrator, also see *z/OS DFSMSdfp Storage Administration Reference* for information about using SMS.

Access Methods

An access method defines the technique that is used to store and retrieve data. Access methods have their own data set structures to organize data, macros to define and process data sets, and utility programs to process data sets.

Access methods are identified primarily by the data set organization. For example, use the basic sequential access method (BSAM) with sequential data sets. However, there are times when an access method identified with one organization can be used to process a data set organized in a different manner. For example, a sequential data set (not extended format data set) created using BSAM can be processed by the basic direct access method (BDAM), and vice versa.

Basic Direct Access Method (BDAM)

BDAM arranges records in any sequence your program indicates, and retrieves records by actual or relative address. If you do not know the exact location of a record, you can specify a point in the data set where a search for the record is to begin. Data sets organized this way are called direct data sets.

Recommendation: Do not use BDAM because it tends to require using device-dependent code.

In addition, using keys is much less efficient than in virtual sequential access method (VSAM). DFSMS supports BDAM only for compatibility with other IBM operating systems. See “Appendix C. Processing Direct Data Sets” on page 535.

Basic Partitioned Access Method (BPAM)

BPAM arranges records as members of a partitioned data set (PDS) or a partitioned data set extended (PDSE) on DASD. You can view each member like a sequential data set. A PDS or PDSE includes a directory that relates member names to locations within the data set. The directory is used to retrieve individual members.

For program libraries (load modules and program objects), the directory contains program attributes that are required to load and rebind the member. Some differences between the two types of partitioned organized data sets follow:

- **Partitioned data set (PDS).** PDSs can have any type of sequential data.
- **Partitioned data set extended (PDSE).** A PDSE has a different internal storage format than a PDS, which gives PDSEs improved usability characteristics. You can use a PDSE in place of most PDS, but you cannot use a PDSE for certain system data sets.

Basic Sequential Access Method (BSAM)

BSAM arranges records sequentially in the order in which they are entered. A data set that has this organization is a sequential data set. The user organizes records with other records into blocks. This is basic access. You can use BSAM data organization types as follows:

- **Extended format data sets.** Extended format data sets have a different internal storage format than sequential data sets that are not extended. This storage format gives extended format data sets improved usability characteristics, such as being able to use striping to improve performance. Extended format data sets must be SMS-managed and must reside on DASD. You cannot use an extended format data set for certain system data sets.

  You can allocate an extended format data set in the compressed format and refer to it as a compressed format data set. A compressed format data set is a type of extended format data set that allows for data compression.
- **HFS files.** BSAM accesses an HFS file as if it were a single-volume, physical sequential data set residing on DASD. Although HFS files are not actually stored as physical sequential data sets, the system attempts to simulate the characteristics of such a data set.

  HFS files are POSIX-conforming files that reside in HFS data sets. They are byte-oriented rather than record-oriented, as data sets are. To identify or access an HFS data set, you specify the path that leads to it.

Data-in-Virtual (DIV)

DIV provides access to VSAM linear data sets. See *z/OS MVS Programming: Assembler Services Guide* for more information.

Indexed Sequential Access Method (ISAM)

ISAM refers to two access methods: basic indexed sequential access method (BISAM) and queued indexed sequential access method (QISAM). Data sets that ISAM processes are called indexed sequential data sets. ISAM arranges records in sequence by key fields, and retrieves records by key. The structure of multilevel indexes in ISAM is similar to the structure of indexes in VSAM.

Recommendation: Do not use ISAM. The system supports ISAM only for compatibility with other IBM operating systems. You should plan to convert all indexed sequential data sets to VSAM data sets. Indexed sequential data sets cannot be system managed. See “Appendix D. Processing Indexed Sequential Data Sets” on page 545.

Object Access Method (OAM)

OAM processes very large, named byte streams (objects) that have no record boundary or other internal orientation. These objects can be recorded in a DB2® database or on an optical storage volume. For information about OAM, see *z/OS*

DFSMS OAM Application Programmer's Reference and *z/OS DFSMS OAM Planning, Installation, and Storage Administration Guide for Object Support*.

Queued Sequential Access Method (QSAM)

QSAM arranges records sequentially in the order that they are entered to form sequential data sets, which are the same as those data sets that BSAM creates. The system organizes records with other records. QSAM anticipates the need for records based on their order. To improve performance, QSAM reads these records into storage before they are requested. This is called queued access.

- **Extended format data sets.** See extended format data sets under BSAM for parameter definition.
- **HFS files.** See HFS files under BSAM for parameter definition.

Virtual Storage Access Method (VSAM)

VSAM arranges records by an index key, relative record number, or relative byte addressing. VSAM is used for direct or sequential processing of fixed-length and variable-length records on DASD. Data that is organized by VSAM is cataloged for easy retrieval and is stored in one of five types of data sets.

- **Entry-sequenced data set (ESDS).** Contains records in the order in which they were entered. Records are added to the end of the data set and can be accessed.
- **Key-sequenced data set (KSDS).** Contains records in ascending collating sequence. Records can be accessed by a field, called a key, or by a relative byte address.
- **Linear data set (LDS).** Contains data that has no record boundaries. Linear data sets contain none of the control information that other VSAM data sets do. Linear data sets must be cataloged in a catalog.
- **Relative record data set (RRDS).** Contains records in relative record number order, and the records can be accessed only by this number. There are two types of relative record data sets.
  - ***Fixed-length RRDS:*** The records must be of fixed length.
  - ***Variable-length RRDS:*** The records can vary in length.

  Throughout this book, the term RRDS refers to both types of relative record data sets, unless they need to be differentiated.
- **HFS files.** VSAM accesses an HFS file as if it were an ESDS. Although HFS files are not actually stored as entry-sequenced data sets, the system attempts to simulate the characteristics of such a data set.

  HFS files are POSIX-conforming files that reside in HFS data sets. They are byte oriented rather than record oriented, as are data sets. To identify or access an HFS file, specify the path that leads to it.

Any type of VSAM data set can be in extended format. (HFS files are not VSAM data sets.) Extended format data sets have a different internal storage format than data sets that are not extended. This storage format gives extended format data sets additional usability characteristics and possibly better performance due to striping. You can choose for an extended format key-sequenced data set to be in the compressed format. Extended format data sets must be SMS managed. You cannot use an extended format data set for certain system data sets.

Recommendation: Do not use BISAM or QISAM. Use VSAM instead.

Access to Files in HFS Data Sets

Programs can access the information in HFS files through z/OS UNIX System Services (USS) calls, such as `open(pathname)`, `read(file descriptor)`, and `write(file descriptor)`. Programs can also access the information in HFS files through the MVS BSAM, QSAM, and VSAM. When you use BSAM or QSAM, an HFS file is simulated as a single-volume sequential data set. When using VSAM, an HFS file is simulated as an ESDS.

The following types of HFS files are supported through the MVS access methods:
- Regular files, including files accessed through NFS
- Character special files
- First-in-first-out (FIFO) special files
- Symbolic links

The following types of HFS files are not supported through the MVS access methods:
- Directories
- External links

Files can reside on other systems. The access method user can use NFS to access them.

Selection of an Access Method

In selecting an access method for a data set, consider the data set's organization, what you need to specify through macros, and the device type:

- VSAM data sets, PDSEs, PDSs, extended format data sets, direct data sets, indexed sequential data sets, and HFS files must be stored on DASD volumes.
- Sequential data sets can be on DASD or tape volumes, or these data sets can be read from or written to a unit record device or TSO/E terminal. They can be spooled data sets. Spooled data sets named SYSOUT can be directed over a network. Sequential data sets also can be dummy data sets.

In addition, you should select a data organization according to the type of processing you want to do: sequential or direct. For example, RRDSs or key-sequenced data sets are best for applications that use only direct access, or both direct and sequential access. Sequential or VSAM entry-sequenced data sets are best for batch processing applications and for sequential access.

Restriction: You cannot process VSAM data sets with non-VSAM access methods, although you can use DIV macros to access linear data sets. You cannot process non-VSAM data sets other than HFS files with VSAM.

See *z/OS TSO/E Command Reference* for information about using BSAM and QSAM to read from and write to a TSO/E terminal in line mode.

Magnetic Tape and DASD Labels

The operating system uses various groups of labels to identify magnetic tape and DASD volumes, and the data sets they contain. Application programs generally do not use these labels directly. Magnetic tape volumes can have standard or nonstandard labels, or they can be unlabeled. DASD volumes must use standard labels. Standard labels include a volume label, a data set label for each data set, and optional user labels. A volume label, stored at track 0 of cylinder 0, identifies each DASD volume.

A utility program initializes each DASD volume before it is used on the system. The initialization program generates the volume label and builds the volume table of contents (VTOC). The VTOC is a structure that contains the data set labels. See “Appendix A. Using Direct Access Labels” on page 527 for information about direct access labels.

Direct Access Storage Device (DASD) Volumes

Although DASD volumes differ in physical appearance, capacity, and speed, they are similar in data recording, data checking, data format, and programming. The recording surface of each volume is divided into many concentric tracks. The number of tracks and their capacity vary with the device. Each device has an access mechanism that contains read/write heads to transfer data as the recording surface rotates past them.

Track Format

Information is recorded on all DASD volumes in a standard format.

Each track contains a record 0 (also called track descriptor record or capacity record) and data records. Historically, System/390® hardware manuals and software manuals have used inconsistent terminology to refer to units of data written on DASD volumes. Hardware manuals call them *records*. Software manuals call them *blocks* and use “record” for something else. The DASD sections of this book use both terms as appropriate. Software records are described in “Chapter 7. Organizing VSAM Data Sets” on page 75 and “Chapter 21. Selecting Record Formats for Non-VSAM Data Sets” on page 289.

For various data formats one or more of the following is true:

- The application program deals with control intervals with VSAM. Each control interval consists of one or more contiguous blocks. Control intervals are grouped into control areas.
- The application program deals with non-VSAM blocks. Each block contains part of a record or one or more records. Examples of these programming interfaces are BSAM, BDAM, and EXCP.
- The application program deals with records. Each VSAM record occupies multiple control intervals or all or part of a control interval. Each non-VSAM record occupies multiple blocks or all or part of a block. An example is QSAM.
- The application program deals with byte streams that the application program might regard as being grouped in records. The program does not see blocks. Examples include HFS files and OAM objects.

The process of grouping records into blocks is called *blocking*. The extraction of records from blocks is called *unblocking*. Blocking or unblocking might be done by the application program or the operating system. In z/OS UNIX, blocking means suspension of program execution.

Although IBM does not recommend the use of BDAM and ISAM, these two access methods use the data area of record zero to contain information about the remaining space on the track.

BDAM uses the track descriptor record to contain the number of empty bytes following the last user data record on the track. ISAM uses the track descriptor record for that purpose for variable-length format data sets.

Figure 1 shows the two different data formats, count-data and count-key-data, only one of which can be used for a particular data set. An exception is PDSs that are not PDSEs. The directory blocks are in count-key-data format, and the member blocks normally are in count-data format.

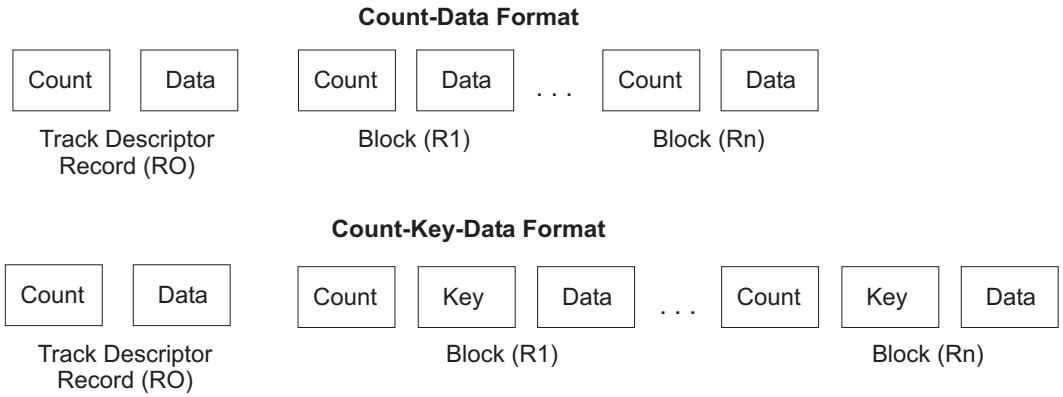


Figure 1. DASD Volume Track Formats

Count-Data Format: Records are formatted without keys. The key length is 0. The count area contains 8-bytes that identify the location of the block by cylinder, head, and record numbers, and its data length.

Count-Key-Data Format: The blocks are written with hardware keys. The key area (1 - 255 bytes) contains a record key that specifies the data record, such as the part number, account number, sequence number, or some other identifier.

In data sets, only ISAM, BDAM, BSAM, EXCP, and PDS directories support blocks with hardware keys. Outside data sets, the VTOC and the volume label contain hardware keys.

Tip: The use of hardware keys is less efficient than the use of software keys (which VSAM supports).

Track Overflow

The operating system no longer supports the track overflow feature. The system ignores any request for it.

VSAM Record Addressing

You identify VSAM records by their key, record number, or relative byte address in the data set. See “Selection of VSAM Data Set Types” on page 84.

Actual and Relative Addressing with Non-VSAM Access Methods

With certain access methods, you can access data non-sequentially. You can use addresses to identify block locations. Use two types of addresses to store and retrieve data on DASD volumes: actual addresses and relative addresses. When sequentially processing a multiple volume data set (except for extended format data sets), you can refer to only records of the current volume.

Actual Addresses

When the system returns the actual address of a block on a direct access volume to your program, it is in the form `MBBCCHHR`, in which the characters represent the following values:

`M` 1-byte binary number specifying the relative extent number. Each extent is a set of consecutive tracks allocated for the data set.

`BBCCHH` Three 2-byte binary numbers specifying the cell (bin), cylinder, and head number for the block (its track address). The cylinder and head numbers are recorded in the count area for each block. All currently supported DASDs require that the bin number (BB) be zero.

`R` 1-byte binary number specifying the relative block number on the track. The block number is also recorded in the count area.

If your program stores actual addresses in your data set, and you refer to those addresses, the data set must be treated as unmovable. Data sets that are unmovable cannot reside on system-managed volumes.

Relative Addresses

BDAM, BSAM, BPAM, and ISAM support two kinds of relative block addresses: relative block addresses and relative track addresses. BDAM uses relative block addresses. BSAM (except with extended format data sets), BPAM, and BDAM use relative track addresses. ISAM records are accessed by key or sequentially.

BSAM relative addresses are relative to the current volume. BDAM relative addresses are relative to the data set and go across all volumes.

The relative block address is a 3-byte binary number that shows the position of the block, starting from the first block of the data set. Allocation of noncontiguous sets of blocks does not affect the number. The first block of a data set always has a relative block address of 0.

The relative track address has the form `TTR`:

`TT` An unsigned 2-byte binary number specifying the position of the track starting from the first track allocated for the data set. The `TT` for the first track is 0. Allocation of noncontiguous sets of tracks does not affect the `TT` number.

`R` 1-byte binary number specifying the number of the block starting from the first block on the track `TT`. The `R` value for the first block of data on a track is 1.

With some devices, such as the IBM 3380 Model K, a data set can contain more than 32 767 tracks. Therefore, assembler halfword instructions could result in invalid data being processed.

Relative Block Addresses for Extended Format Data Sets. For extended format data sets, addressing capability is provided through the use of block locator tokens (BLTs). A BLT can be used transparently, as if it were a relative track record (`TTR`). The NOTE macro returns a 4-byte value in which the three high-order bytes are the BLT value and the fourth byte is a zero. Your program uses the value from the NOTE macro as input to the POINT macro, which provides positioning within a

sequential data set through BSAM. The BLT is essentially the relative block number (RBN) within each logical volume of the data set (where the first block has an RBN of 1). For compressed format data sets, the relative block numbers represent uncompressed simulated blocks, not the real compressed blocks.

A multistriped data appears to the user as a single logical volume. Therefore, for a multistriped data set, the RBN is relative to the beginning of the data set and incorporates all stripes.

Relative Track Addresses for PDSEs. For PDSEs, the relative track addresses (TTRs) do not represent the actual track and record location. Instead, the TTRs are tokens that define the record's position within the data set. See "Relative Track Addresses (TTR)" on page 433 for a description of TTRs for PDSE members and blocks.

Magnetic Tape Volumes

Because data sets on magnetic tape devices must be organized sequentially, the procedure for allocating space is different from DASD. When a new data set is to be placed on a magnetic tape volume, you must specify the data set sequence number if the data set is not the first one on the reel or cartridge. The system positions a volume with IBM standard labels, ISO/ANSI labels, or no labels so that the data set can be read or written. If the data set has nonstandard labels, your installation must provide volume positioning in the nonstandard label processing routines. All data sets stored on a given magnetic tape volume must be recorded in the same density. See *z/OS DFSMS: Using Magnetic Tapes* for information about magnetic tape volume labels and tape processing. See *z/OS DFSMS Installation Exits* for information about nonstandard label processing routines.

Label Character Coding

International Organization for Standardization (ISO) and the American National Standards Institute (ANSI) tape labels are similar to IBM standard labels. IBM standard labels are coded in the extended binary-coded-decimal interchange code (EBCDIC). ISO/ANSI labels are coded in the American National Standards Code for Information Interchange (ASCII).

Data Conversion

The data on magnetic tape volumes can be in any format. For example it might be in EBCDIC, ASCII, or binary. ASCII is a 7-bit code consisting of 128 characters. It permits data on magnetic tape to be transferred from one computer to another, even though the two computers can be products of different manufacturers.

For ISO/ANSI Version 3 tapes, data management converts ASCII records on input tapes into EBCDIC for internal processing and converts EBCDIC records into ASCII for output. For ISO/ANSI Version 4, the type of conversion performed depends on whether Coded Character Set Identifiers (CCSIDs) are supplied through any source. If CCSIDs are supplied, data management converts records from one CCSID to another. See *z/OS DFSMS: Using Magnetic Tapes* for information about magnetic tape data set labels. See "Character Data Conversion" on page 299 for information about CCSID conversion. If no CCSIDs are supplied, data conversion is the same as for ISO/ANSI Version 3.

Identification of Unlabeled Tapes

When a data set is to be stored on unlabeled tape volumes, the system needs a volume serial number to identify each volume. If the data set is in an Automatic Tape Library, the system uses the volume serial number that is encoded in the bar code on the outside of each cartridge. If the data set is not in an Automatic Tape Library, it is advisable to specify enough volume serial numbers to contain the data set. If you do not specify any volume serial numbers or do not specify enough of them, the system or a tape management system assigns a serial number to each unidentified volume. If the system assigns a serial number, the serial number is in the form *Lxxxyy*, in which *xxx* is the data set sequence number and *yy* is the volume sequence number for the data set.

If data sets that reside on unlabeled volumes are to be cataloged or passed, you should specify the volume serial numbers for the volumes required. Specifying the volume serial numbers ensures that data sets residing on multiple volumes are not cataloged or passed with duplicate volume serial numbers. Retrieving such data sets can give unpredictable errors.

Tape Marks

A tape mark must follow each data set and data set label group. Tape marks cannot exist within a data set. When a program writes data on a standard labeled or unlabeled tape, the system automatically reads and writes labels and tape marks. Two tape marks follow the last trailer label group on a standard-label volume. On an unlabeled volume, the two tape marks appear after the last data set.

When a program writes data on a nonstandard labeled tape, the installation must supply routines to process labels and tape marks and to position the tape. If you want the system to retrieve a data set, the installation routine that creates nonstandard labels must write tape marks. Otherwise, tape marks are not required after nonstandard labels because installation routines handle positioning of the tape volumes.

Data Management Macros

You can use macros to process all the data set types supported by the access methods just described. Macros control data set allocation, input and output, the buffering techniques used, and data security. See *z/OS DFSMS Macro Instructions for Data Sets* for information about data management macros. See *z/OS DFSMSdfp Advanced Services* for information about system programming macros.

Table 1 on page 13 contains a summary of data management access methods:

Table 1. Data Management Access Methods

VSAM Data Set Organization	Access Method	Basic	Queued
Direct	BDAM		
ESDS			VSAM
Indexed Sequential	BISAM	QISAM	
KSDS			VSAM
LDS			VSAM, DIV[1]
Partitioned[2]	BPAM, BSAM[3]	QSAM	
RRDS[4]			VSAM
Sequential[5]	BSAM	QSAM	
HFS file	BSAM[6]	QSAM[6]	VSAM[7]

Notes:

1. The data-in-virtual (DIV) macro, which is used to access a linear data set, is described in *z/OS MVS Programming: Assembler Services Guide*.
2. PDSs and PDSEs are both partitioned organization data sets.
3. BSAM and QSAM cannot be used to create or modify user data in directory entries.
4. Refers to fixed-length and variable-length RRDSs.
5. Sequential data sets and extended format data sets are both sequential organization data sets.
6. When you access an HFS file with BSAM or QSAM, the file is simulated as a single-volume sequential data set.
7. When you access an HFS file with VSAM, the file is simulated as an ESDS.

Data sets can also be organized as PDSE program libraries. PDSE program libraries can be accessed with BSAM, QSAM, or the program management binder. The first member written in a PDSE library determines the library type, either program or data.

Data Set Processing

To process a data set, first allocate it (establish a link to it), and then access the data using macros for the access method that you have chosen. See *z/OS DFSMS: Using Magnetic Tapes* for information about magnetic tape volume labels. See “USS File Processing” on page 21 for information about accessing HFS files.

Allocating Data Sets

Allocate means either or both of two things:

- To set aside (create) space for a new data set on a disk
- To establish a logical link between a job step and any data set

Related reading: For information about dynamic allocation, see *z/OS MVS Programming: Authorized Assembler Services Guide*.

You can use any of the following methods to allocate a data set.

Access Method Services

You can define data sets and establish catalogs by using a multifunction services program called access method services. For information about access method services commands see *z/OS DFSMS Access Method Services*. Use the following commands with all data sets:

ALLOCATE - Allocate a new data set
ALTER - Change the attributes of a data set
DEFINE NONVSAM - Catalog a data set
DELETE - Delete a data set
LISTCAT - List catalog entries
PRINT - Print a data set

ALLOCATE Command

You can issue the ALLOCATE command either through access method services or TSO/E to define VSAM and non-VSAM data sets. For information about TSO commands, see *z/OS TSO/E Command Reference*.

JCL

All data sets can be defined directly through JCL.

Related reading: For information about using JCL, see *z/OS MVS JCL Reference* and *z/OS MVS JCL User's Guide*.

Processing Data Sets through Programs

Programs process data sets in the following sequence:

1. Allocate the data set to establish the logical link between a program and a data set. You can do this either outside the program with JCL or the TSO ALLOCATE command or inside the program with dynamic allocation.
2. Open the data set, identifying it with a DDNAME.
3. Do reads and writes using an access method.
4. Close the data set.
5. Unallocate the data set. There are three ways to do this:
   - Closing the data set can perform this function if FREE=CLOSE was specified for the allocation.
   - Your program can call dynamic unallocation.
   - During the step termination process, the operating system automatically unallocates any remaining allocated data sets.

Using Access Methods

All the access methods described in this book allow you to do the following:

- Share a data set among different systems, different jobs in a single system, multiple access method control blocks (ACBs) or data control blocks (DCBs) in a task, or different subtasks in an address space. See "Chapter 13. Sharing VSAM Data Sets" on page 195 for information about sharing a VSAM data set. See "Chapter 24. Sharing Non-VSAM Data Sets" on page 365 for information about sharing a non-VSAM data set.
- Share buffers and control blocks among VSAM data sets. See "Chapter 14. Sharing Resources Among VSAM Data Sets" on page 211.
- Provide user exit routines to analyze logical and physical errors, and to perform end-of-data processing. See "Chapter 31. Identifying Non-VSAM User-Written Exit Routines" on page 485 and "Chapter 17. Coding VSAM User-Written Exit Routines" on page 241.

- Back up and recover data sets. See “Chapter 4. Backing Up and Recovering Data Sets” on page 47.
- Maintain data security and integrity. See “Chapter 5. Protecting Data Sets” on page 55.

BSAM, QSAM, and VSAM convert between record-oriented data and byte-stream oriented data that is stored in HFS files.

Non-VSAM access methods also let you:
- Convert non-VSAM data from ASCII to EBCDIC, and the reverse.
- Position and reposition tape volumes automatically.
- Process user labels for data sets on DASD and magnetic tape.

Using Addressing Modes

The 24-bit and 31-bit access method interfaces support real addresses above the 2 GB bar. When you use a 24-bit or 31-bit addressing mode, different rules apply for VSAM programs and non-VSAM programs.

VSAM Addressing Modes

You can use either 24-bit or 31-bit addressing mode for VSAM programs. You can issue the OPEN and CLOSE macros for any access method in 24-bit or 31-bit addressing mode. VSAM lets you create buffers, user exits, shared resource pools, and some control blocks in virtual storage above 16 MBs. Your program must run in 31-bit addressing mode to access these areas above 16 MBs. See “Chapter 18. Using 31-Bit Addressing Mode with VSAM” on page 263.

Non-VSAM Addressing Modes

You can run most BSAM, BPAM, QSAM, and BDAM macros in 24-bit or 31-bit addressing mode. Data to which the macros refer must reside below the 16 MB line if you run the macro in 24-bit mode.

The BSAM, BPAM, QSAM, and BDAM access methods let you create certain data areas, buffers, certain user exits, and some control blocks in virtual storage above the 16 MB line if the macro is executed in 31-bit mode. See *z/OS DFSMS Macro Instructions for Data Sets*.

Using Hiperspace™ and Hiperbatch™

Hiperbatch can significantly reduce the execution time of batch job streams that access QSAM or VSAM data sets. Hiperspace improves the performance for VSAM applications that use local shared resources (LSR). Batch LSR lets you use the advantages of Hiperspace for VSAM applications that use nonshared resources without changing the application. See “Using Hiperbatch” on page 395 and “Using Hiperspace Buffers with LSR” on page 212.

Processing VSAM Data Sets

There are two types of VSAM macro instructions:
- **Control block macros.** Generate control blocks of information that VSAM needs to process the data set.
- **Request macros.** Retrieve, update, delete, or insert logical records.

Introducing Data Sets

VSAM has two major parts:

- **Catalog management.** VSAM maintains extensive information about data sets and direct access storage space in a catalog. The catalog's collection of information about a particular data set defines that data set's characteristics. Every VSAM data set must be defined in a catalog. You cannot, for example, load records into a VSAM data set until it has been defined. See *z/OS DFSMS: Managing Catalogs* for information about catalog management.
- **Record management.**You can use VSAM to organize records into four types of data sets: key-sequenced, entry-sequenced, linear, or relative record. The primary difference among these types of data sets is the way their records are stored and accessed.

Restriction: VSAM data sets cannot be concatenated in JCL statements.

Processing PDSs and PDSEs

The following guidelines apply to processing PDSs and PDSEs:

- To process the directory of a PDS or PDSE, you use BPAM.
- Each PDS or PDSE must be on one direct-access volume. However, you can concatenate multiple input data sets that are on the same or different volumes.
- A PDSE can be used as a data library or program library, but not both. The first member stowed in a library determines the library type.
- You can use either BSAM or QSAM macros to add or retrieve a member without specifying the BLDL, FIND, or STOW macro. Code the DSORG=PS parameter in the DCB macro, and the DDNAME parameter of the JCL DD statement with both the data set and member names as follows:

  ```
  //ddname DD DSN=LIBNAME(MEMNAME),...
  ```

  (The OPEN and CLOSE macros then handle data set positioning and directory maintenance.) Because you are processing the member as if it were part of a sequential data set, you are not using the complete capabilities of BPAM.
- When you create a PDS, the SPACE parameter defines the size of the data set and its directory so the system can allocate data set space. For a PDS, the SPACE parameter preformats the directory. The specification of SPACE for a PDSE is different from the specification for a PDS. See "Allocating Space for a PDSE" on page 437.
- You can use the STOW macro to add, delete, change, or replace a member name or alias in the PDS or PDSE directory, or clear a PDSE directory. You can also use the STOW macro to delete all the members of a PDSE. However, you cannot use the STOW macro to delete all the members of a PDS. For program libraries, STOW cannot be used to add or replace a member name or alias in the directory.
- You can process multiple data set members by passing a list of members to BLDL. Then use the FIND macro to position to a member before it is processed.
- You can code a DCBE and use 31-bit addressing for PDSs and PDSEs.
- PDSs and PDSEs and members cannot use sequential data striping. See "Chapter 27. Processing a Partitioned Data Set (PDS)" on page 405 and "Chapter 28. Processing a Partitioned Data Set Extended (PDSE)" on page 429. Also see *z/OS DFSMS Macro Instructions for Data Sets* for information about coding the DCB (BPAM) and DCBE macros.

Processing Sequential Data Sets and Members of PDSEs and PDSs

To process a sequential data set or members of a PDS or PDSE, you can use BSAM or QSAM You can be in 31-bit addressing mode when issuing BSAM and QSAM macros. Data areas can be above the 16 MB line for BSAM and QSAM macros, and you can request that OPEN obtain buffers above the 16 MB line for QSAM. This permits larger amounts of data to be transferred.

BSAM Processing

When you use BSAM to process a sequential data set and members of a PDS or PDSE, the following rules apply:

- BSAM can read a member of a PDSE program library, but not write one.
- The application program must block and unblock its own input and output records. BSAM only reads and writes data blocks.
- The application program must manage its own input and output buffers. It must give BSAM a buffer address with the READ macro, and it must fill its own output buffer before issuing the WRITE macro.
- The application program must synchronize its own I/O operations by issuing a CHECK macro for each READ and WRITE macro issued.
- BSAM lets you process blocks in a nonsequential order by repositioning with the NOTE and POINT macros.
- You can read and write direct access storage device record keys with BSAM. PDSEs and extended format data sets are an exception.
- BSAM automatically updates the directory when a member of a PDS or PDSE is added or deleted.

QSAM Processing

When you use QSAM to process a sequential data set and members of a PDS or PDSE, the following rules apply:

- QSAM processes all record formats except blocks with keys.
- QSAM blocks and unblocks records for you automatically.
- QSAM manages all aspects of I/O buffering for you automatically. The GET macro retrieves the next sequential logical record from the input buffer. The PUT macro places the next sequential logical record in the output buffer.
- QSAM gives you three transmittal modes: move, locate, and data. The three modes give you greater flexibility managing buffers and moving data.

Processing USS Files with an Access Method

USS (UNIX system services) files are POSIX-conforming files that reside in USS file systems. For information about USS files, see “USS File Processing” on page 21. You can access a USS file through BSAM, QSAM (DCB DSORG=PS), or VSAM (simulated as an ESDS) by specifying PATH=pathname in the JCL data definition (DD) statement, SVC 99, or TSO ALLOCATE command.

The most common type of USS file system is HFS (hierarchical file system). HFS files reside in HFS data sets, which are hierarchical file systems. Examples of other USS file systems are NFS (Network File System) and TFS (temporary file system).

Before you can create an HFS file, someone with appropriate RACF®, a component of the Security Server for z/OS, authority allocates DASD space and logically mounts the file system. See “Chapter 6. Using Hierarchical File System Data Sets” on page 71.

BSAM, QSAM, and VSAM support the following types of USS files:

- Regular files
- Character special files
- First-in-first-out (FIFO) special files
- Symbolic links

BSAM, QSAM, and VSAM do not support the following types of USS files:

- Directories
- External links

Data can reside on a system other than the one the user program is running on without using shared DASD. The other system can be MVS or non-MVS. NFS transports the data.

You should be familiar with the information about USS and HFS files in *z/OS UNIX System Services User’s Guide*.

Because the system does not actually store USS files as physical sequential data sets or ESDSs, it cannot simulate all the characteristics of a sequential data set. Because of this, certain macros and services have incompatibilities or restrictions when they process USS files. For example:

- Data set labels and UCBs do not exist for USS files. Any service that relies on a DSCB or UCB for information might not work with these files.
- With traditional MVS data sets, other than VSAM linear data sets, the system maintains record boundaries. That is not true with byte-stream files such as USS files.

See “Accessing USS Files Using BSAM, QSAM, or VSAM” on page 401. Also see *z/OS DFSMS Macro Instructions for Data Sets* for information about coding the DCB (BSAM) and DCB (QSAM) macros, and DCBE (BSAM) and DCBE (QSAM) macros.

Allocation Considerations

To allocate a USS file for access through BSAM, QSAM (DCB DSORG=PS), or VSAM, specify the PATH=pathname parameter in the JCL DD statement, SVC 99, or TSO ALLOCATE command. The OPEN macro supports DD statements that specify the PATH parameter only for DCBs that specify DSORG=PS and for ACBs.

Rule: EXCP is not allowed.

See *z/OS MVS JCL Reference* for information about the PATH parameter, including the list of parameters that can be coded with a PATH parameter and the defaults for a DD statement with a PATH parameter.

The JCL DD statement, SVC 99, and TSO ALLOCATE allow the following DCB parameters with the PATH parameter:

- BLKSIZE
- LRECL
- RECFM
- BUFNO
- NCP

BLKSIZE, RECFM, and LRECL values are not stored with a USS file. If you do not want the default values, you must specify values for these fields in JCL, SVC 99, or TSO ALLOCATE or in the DCB.

FILEDATA Parameter

The access methods support both EBCDIC text USS files and binary HFS files. When creating a file, the user is responsible for specifying whether the file consists of text data or binary data by coding the FILEDATA parameter in the JCL DD statement, SVC 99, or TSO ALLOCATE command. This parameter value is saved if the file is an HFS file.

FILEDATA=TEXT—Indicates the data consists of records separated by a delimiter of the EBCDIC newline character (x’15’). (The record delimiters are transparent to the user.)

FILEDATA=BINARY—Indicates the data does not contain record delimiters.

The FILEDATA parameter is currently meaningful only when the PATH parameter is also coded and the program uses BSAM, QSAM, or VSAM.

When FILEDATA is coded for an existing file, the value specified temporarily overrides the value saved with the file. However, there is one exception. If the existing file does not already have a FILEDATA value saved with the file (that is, the HFS file was created without coding FILEDATA) and the user has the proper authority to update the HFS file, an attempt is made during OPEN to save the specified FILEDATA value with the file.

When FILEDATA is not coded and a value has not been saved with the file, the file is treated as binary.

File Status

Because you cannot put the DISP parameter in a DD statement that contains a PATH parameter, use the PATHOPTS parameter to specify the status for an HFS file, as follows:

OLD—PATHOPTS is not on the DD statement
PATHOPTS does not contain a file option
PATHOPTS does not contain OCREAT

MOD—PATHOPTS contains OCREAT but not OEXCL

NEW—PATHOPTS contains both OCREAT and OEXCL

SMS Constructs

For a DD statement containing a PATH parameter, DATACLAS, STORCLAS, and MGMTCLAS cannot be specified and the ACS routines are not called.

Path-Related Parameters

During allocation of a new USS file (when either OCREAT or OCREAT together with OEXCL is specified in the PATHOPTS parameter and the file does not exist), DFSMS performs an open( ) function. The pathname from the PATH parameter, the options from PATHOPTS, and the options from PATHMODE (if specified) are passed to the open( ) function without modification. The system then uses the close( ) function to close the file before the application program receives control.

When the application program issues an OPEN macro for a USS file, the macro performs an open( ) function to establish a connection to the existing file. The pathname from the PATH parameter is passed without modification. The options from PATHMODE are not passed because the USS file must already exist. The options from PATHOPTS are passed with the following modifications:

- The system ignores OCREAT and OEXCL, if specified, because the file must already exist.
- If the file is a regular file and OWRONLY is specified, the system sets ORDWR to ensure that both reads and writes are permitted.
- The open is for input if the file is an FIFO or character special file and either of the following is true:
  - The first option of DCB OPEN is INPUT, INOUT, or omitted.
  - The ACB MACRF=OUT option is not coded.
- The open is for output if the file is an FIFO or character special file and either of the following is true:
  - The first option of DCB OPEN is OUTPUT, OUTIN, EXTEND, or OUTINX.
  - The ACB MACRF=OUT option is coded.

z/OS UNIX Signals

The access methods do not perform any type of signal processing. The only signal that might be expected is when a FIFO “breaks”. That is, if the reader closes the file and the writer tries to write to it. This results in a signal (SIGPIPE) that is sent to the writer. The default action for the signal is to terminate the writer’s task with an ABEND EC6-FF0D.

z/OS UNIX Fork Service

You should not use the z/OS UNIX MVS fork service while a DCB or ACB is open to a USS file. The fork service creates a child process that is a duplicate of the calling (parent) process; however, the service does not duplicate various MVS control blocks. This would give unpredictable results in the child process.

SMF Records

CLOSE does not write SMF type 14/15 or 60 - 69 records for USS files. DFSMS relies on z/OS UNIX system services to write appropriate SMF records as requested by the system programmer.

Migration and Coexistence Considerations

An attempt to access a USS file with BSAM, QSAM, or VSAM on a down-level DFSMS release results in ABEND 213-88.

System Security

DFSMS depends on z/OS UNIX MVS to address security for files being accessed. For USS files, the file system called by z/OS UNIX MVS actually performs authorization (RACF) checking. See “Chapter 26. Processing Sequential Data Sets” on page 377 for information about BSAM and QSAM access. Also see “Chapter 7. Organizing VSAM Data Sets” on page 75 for information about VSAM access.

Data Set Processing without an Access Method

The following sections contain information about using techniques other than access methods to process data sets.

EXCP, EXCPVR, and XDAP Macros

It is possible to control an I/O device directly while processing a data set with almost any data organization without using a specific access method. The EXCP (execute channel program), EXCPVR, and XDAP macros use the system function that provides for scheduling and queuing I/O requests, efficient use of channels and devices, data protection, interruption procedures, error recognition, and retry. See *z/OS DFSMSdfp Advanced Services* for information about the EXCP, EXCPVR, and XDAP macros. Using these macros is not recommended. These macros cannot be used to process PDSEs, extended format data sets, VSAM data sets, HFS data sets or files, dummy data sets, TSO/E terminals, spooled data sets or OAM objects. The use of EXCP, EXCPVR, and XDAP require detailed knowledge of channel programs, error recovery, and physical data format. Your program is less likely to work with future forms of data than if they used standard access methods of VSAM, BSAM, QSAM, and BPAM.

USS File Processing

USS (UNIX system services) files are POSIX-conforming files that reside in USS file systems. See Figure 2 on page 22. The most common type of USS file system is HFS (hierarchical file system). HFS files reside in HFS data sets, which are hierarchical file systems. See “Chapter 6. Using Hierarchical File System Data Sets” on page 71. Examples of other types of USS file systems are NFS (Network File System) and TFS (temporary file system).

Programs can access the information in USS files through z/OS UNIX system calls or through standard MVS access methods and macro instructions. See “Simulated VSAM Access to USS files” on page 86 and “Accessing USS Files Using BSAM, QSAM, or VSAM” on page 401. To identify and access a data file, specify the path leading to it.

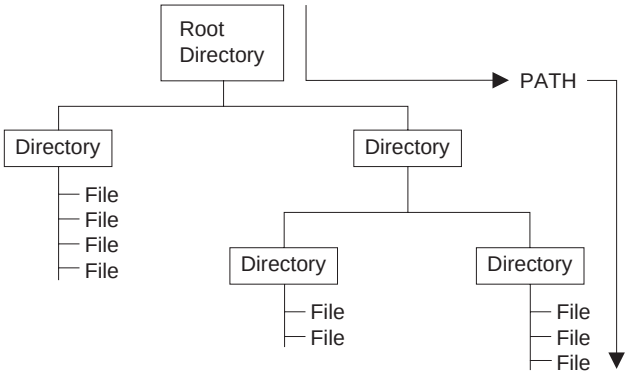


Figure 2. USS Directories and Files in a File System

Distributed Data Management (DDM) Attributes

Distributed file manager is the z/OS implementation of Systems Application Architecture® (SAA®) distributed data management (DDM). Distributed file manager facilitates access by programs running on non-MVS systems to data sets stored on an MVS system. Distributed file manager supports remote record and stream access to sequential and VSAM data sets and PDSE members. Distributed file manager also provides support so that DDM attributes associated with these data sets and members can be propagated when the data sets and members are moved or copied.

Distributed file manager creates and associates DDM attributes with data sets. The DDM attributes describe the characteristics of the data set, such as the file size class and last access date. The end user can determine whether a specific data set has associated DDM attributes by using the ISMF Data Set List panel and the IDCAMS DCOLLECT command.

Distributed file manager also provides the ability to process data sets along with their associated attributes. Any DDM attributes associated with a data set cannot be propagated with the data set unless DFSMShsm™ uses DFSMSdss™ as its data mover. See *z/OS DFSMS DFM Guide and Reference* for information about the DDM file attributes.

Virtual I/O for Temporary Data Sets

Temporary data sets can be handled by a function called virtual I/O (VIO). Data sets for which VIO is specified are located in external page storage. However, to the access methods, the data sets appear to reside on real direct access storage devices. A VIO data set must be a non-VSAM data set; it can be sequential, partitioned, or direct, but not a PDSE or extended format. VIO simulates a real device and provides the following advantages:

- Elimination of some of the usual I/O device allocation and data management overhead for temporary DASD data sets.
- Generally more efficient use of direct access storage space.

A VIO data set appears to the application program to occupy one unshared virtual (simulated) direct access storage volume. This simulated volume is like a real direct access storage volume except for the number of tracks and cylinders. A VIO data set can occupy up to 65 535 tracks even if the device being simulated does not have that many tracks.

A VIO data set always occupies a single extent (area) on the simulated device. The size of the extent is equal to the primary space amount plus 15 times the secondary amount (VIO data size = primary space + (15 × secondary space)). An easy way to specify the largest possible VIO data set in JCL is SPACE=(TRK,65535). You can set this limit lower. Specifying *ALX* (all extents) or *MXIG* (maximum contiguous extents) on the SPACE parameter results in the largest extent allowed on the simulated device, which can be less than 65 535 tracks.

There is no performance or resource penalty for overestimating how much space you need unless your system's accounting functions charge for it.

Do not allocate a VIO data set with zero space. Failure to allocate space to a VIO data set will cause unpredictable results when reading or writing.

A summary of the effects of ALX or MXIG with VIO data sets follows.

Simulated IBM Device	Number of Cylinders
3380	885
3390	1113
9345	1440

Data Set Names

When you allocate a new data set (or when the operating system does), you must give the data set a unique name. Usually, the data set name is the `dsname` value in JCL.

The following rules apply to naming data sets:

- If quotation marks delimit a data set name in a JCL DD statement, JCL processing cannot perform syntax checking on the statement, and SMS does not allow the name to be catalogued because the data set cannot be SMS managed.
- IDCAMS does not allow the specification of invalid data set names.
- When you invoke Dynamic Allocation services or directly invoke SVC 26, it is possible to create invalid data set names. When the CATALOG routine is called to add the data set to a catalog, there is no way to determine whether the original name was in JCL or whether quotation marks delimit the name. The catalog component validates the syntax of a data set name and fails the request if the syntax is not valid, unless the syntax-checking option for data set names is off. See the description of the MODIFY CATALOG command's DSNCHECK parameter in *z/OS DFSMS: Managing Catalogs*.

A data set name can be from one to a series of twenty-two joined name segments. Each name segment represents a level of qualification. For example, the data set name `DEPT58.SMITH.DATA3` is composed of three name segments. The first name on the left is called the high-level qualifier, the last is the low-level qualifier.

Each name segment (qualifier) is 1 to 8 characters, the first of which must be alphabetic (A to Z) or national (# @ $). The remaining seven characters are either alphabetic, numeric (0 - 9), national, a hyphen (-), or X'C0'. Name segments are separated by a period (.).

Data set names must not exceed 44 characters, including all name segments and periods.

See “Naming a Cluster” on page 106 and “Naming an Alternate Index” on page 121 for examples of naming a VSAM data set.

Restriction: The use of name segments longer than 8 characters would produce unpredictable results.

You should use only the low-level qualifier G*xxxx*V*yy*, in which *xxxx* and *yy* are numbers, in the names of generation data sets. Define a data set with G*xxxx*V*yy* as the low-level qualifier of non-generation data sets only if a generation data group with the same base name does not exist. However, IBM recommends that you restrict G*xxxx*V*yy* qualifiers to generation data sets, to avoid confusing generation data sets with other types of non-VSAM data sets.

For example, the following names are not valid data set names:

- A name that is longer than 8 characters (HLQ.**ABCDEFGHI**.XYZ)
- A name that contains two successive periods (HLQ**..**ABC)
- A name that ends with a period (HLQ.ABC**.**)
- A name that contains a segment that does not start with an alphabetic or national character (HLQ.**123**.XYZ)

Catalogs and VTOCs

DFSMS uses a catalog and a volume table of contents (VTOC) on each DASD to manage the storage and placement of data sets. The VTOC lists the data sets that reside on its volume, along with information about the location and size of each data set, and other data set attributes. See *z/OS DFSMSdfp Advanced Services* for information about the VTOC structure. (Application programmers usually do not need to know the contents of the VTOC.) Also see “Appendix A. Using Direct Access Labels” on page 527.

A catalog describes data set attributes and indicates the volumes on which a data set is located. Data sets can be cataloged, uncataloged, or recataloged. All system-managed DASD data sets are cataloged automatically in a catalog. Cataloging of data sets on magnetic tape is not required but usually it simplifies users jobs. All data sets can be cataloged in a catalog.

All types of data sets can be cataloged through:

- Job control language (DISP parameter)
- Access method services (ALLOCATE or DEFINE commands)
- TSO ALLOCATE command
- Dynamic allocation (SVC 99) or DYNALLOC macro

Non-VSAM data sets can also be cataloged through the catalog management macros (CATALOG and CAMLST). An existing data set can be cataloged through the access method services DEFINE RECATALOG command.

Access method services is also used to establish aliases for data set names and to connect user catalogs to the master catalog. See *z/OS DFSMS: Managing Catalogs* for information about using catalog management macros.

Chapter 2. Using the Storage Management Subsystem

This chapter covers the following topics.

Topic	Page
Using Automatic Class Selection Routines	29
Allocating Data Sets	30

When you allocate or define a data set to use SMS, you specify your data set requirements by using a data class, a storage class, and a management class. Typically, you do not need to specify these classes because a storage administrator has set up automatic class selection (ACS) routines to determine which classes to use for a data set.

Descriptions of the classes follow:

- A data class is a named list of data set allocation and space attributes that SMS assigns to a data set when it is allocated. You can also use a data class with a non-system-managed data set.
- A storage class is a named list of data set service or performance objectives that SMS uses to identify performance and availability requirements for data sets. The object access method (OAM) uses storage classes to control the placement of objects in an object storage hierarchy. Each data set has a storage class if and only if the data set is SMS-managed.
- A management class is a named list of management attributes that DFSMShsm uses to control action for retention, migration, backup, and release of allocated but unused space in data sets. OAM uses management classes to control action for the retention, backup, and class transition of objects in an object storage hierarchy. DFSMSrmm™ can use the management class name assigned to a tape data set to identify a policy which should be used to manage the data set. DFSMSrmm extends this support to non-system-managed tape data sets by doing its own call to the Management Class ACS routine for these tape data sets. See *z/OS DFSMSrmm Implementation and Customization Guide*.

Your storage administrator defines the attributes of each class in an SMS configuration. An SMS configuration is a complete set of definitions, ACS routines, and other system information SMS uses to manage your data sets. The definitions group data sets according to common characteristics. As you allocate new data sets, the ACS routines assign those characteristics. With the information contained in the SMS configuration, SMS manages your data sets most effectively with a knowledgeable use of the available hardware. See *z/OS DFSMSdfp Storage Administration Reference* and *MVS/ESA SML: Managing Data* for information about using SMS classes and managing data sets and volumes.

The Storage Management Subsystem (SMS) can manage tape data sets on native volumes in a tape library and on the logical volumes in a Virtual Tape Server (VTS). DFSMSrmm provides some services for the stacked volumes contained in a Virtual Tape Server. See *z/OS DFSMSrmm Implementation and Customization Guide*.

© Copyright IBM Corp. 1987, 2002

Using the Storage Management Subsystem

Some requirements for using SMS follow:

- Extended-format data sets and compressed-format data sets must be system managed.
- SMS must be active when you allocate a new data set to be SMS managed.
- Job steps in which a JOBCAT or STEPCAT DD statement is used cannot use system-managed data sets.
- Your storage administrator must be aware that ACS routines are used for data sets created with distributed file manager (DFM). These data sets must be system managed. If the storage class ACS routine does not assign a storage class, distributed file manager deletes the just-created data set, because distributed file manager does not support the creation of non-system-managed data sets. Distributed file manager does, however, support the access of non-system-managed data sets.

Table 2 lists the storage management functions and products you can use with system-managed and non-system-managed data sets. For details, see *MVS/ESA SML: Managing Data*.

Table 2. Data Set Activity for Non-System-Managed and System-Managed Data Sets

Activity Allocation	**Non-System-Managed Data**	**System-Managed Data**
Data placement	JCL, storage pools	ACS, storage groups
Allocation control	Software user installation exits	ACS
Allocation authorization, definition	RACF[3], JCL, IDCAMS, TSO/E, DYNALLOC	RACF[3], data class, JCL, IDCAMS, TSO/E, DYNALLOC
Access		
Access authorization	RACF[3]	RACF[3]
Read/write performance, availability	Manual placement, JCL, DFSMSdss[1], DFSMShsm[2]	Management and storage class
Access method access to POSIX (UNIX) byte stream	Not available	JCL (PATH=)
Space Management		
Backup	DFSMShsm[2], DFSMSdss[1], utilities	Management class
Expiration	JCL	Management class
Release unused space	DFSMSdss[1], JCL	Management class, JCL
Deletion	DFSMShsm[2], JCL, utilities	Management class, JCL
Migration	DFSMShsm[2]	Data and management class, JCL

Notes:

1. DFSMSdss: Moves data (dump, restore, copy, and move) between volumes on DASD devices, manages space, and converts data sets or volumes to SMS control. See *z/OS DFSMSdss Storage Administration Guide* for information about using DFSMSdss.
2. DFSMShsm: Manages space, migrates data, and backs up data through SMS classes and groups. See *z/OS DFSMShsm Managing Your Own Data* for information about using DFSMShsm.
3. RACF: Controls access to data sets and use of system facilities.

The following types of data sets cannot be system managed:

- Data sets having the same name as an already cataloged data set
- DASD data sets not cataloged in catalogs
- Unmovable data sets (DSORG is xxU) except when set by a checkpoint function
- Data sets with absolute track allocations (ABSTR value for SPACE parameter on DD statement)
- ISAM data sets
- Tape data sets
- Spooled data sets

Direct data sets (BDAM) can be system-managed but if a program uses OPTCD=A, the program might become dependent on where the data set is on the disk. For example, the program might record the cylinder and head numbers in a data set. Such a data set should not be migrated or moved. You can specify a management class that prevents automatic migration.

Tape volumes in a system-managed tape library can be managed as system-managed storage classes.

Using Automatic Class Selection Routines

ACS routines determine if a data set is system managed and which classes are to be used.

You can use a storage class and a management class only with system-managed data sets. You can use a data class for data sets that are either system managed or not system managed, and for data sets on either DASD or tape volumes. SMS can manage tape data sets on physical volumes in a tape library and on the logical volumes in a Virtual Tape Server (VTS). DFSMSrmm provides some services for the stacked volumes contained in a Virtual Tape Server (see *z/OS DFSMSrmm Implementation and Customization Guide*). Your storage administrator defines the data classes, storage classes, and management classes your installation will use. Your storage administrator provides a description of each named class, including when to use the class.

Recommendation: Your storage administrator must code storage class ACS routines to ensure data sets allocated by remote applications using distributed file management are system managed.

Using a data class, you can easily allocate data sets without specifying all of the data set attributes normally required. Your storage administrator can define standard data set attributes and use them to create data classes, for use when you allocate your data set. For example, your storage administrator might define a data class for data sets whose names end in LIST and OUTLIST because they have similar allocation attributes. The ACS routines can then be used to assign this data class, if the data set names end in LIST or OUTLIST.

Using the Storage Management Subsystem

You can request a data class explicitly, by specifying it in the DD statement, DYNALLOC macro, the TSO or IDCAMS ALLOCATE command, or the DEFINE CLUSTER command. Request a data class implicitly, by not specifying a data class and letting the ACS routines assign the data class defined by your storage administrator. Whichever method is used, you need to know:

- The data set characteristics
- The data class that describes what this data set looks like
- Whether the ACS routines will pick this data class automatically
- Which characteristics to code in the JCL to override the data class attributes

You can override any of the attributes specified in the assigned data class by specifying the values in the DD statement, or the ALLOCATE or the DEFINE CLUSTER commands. See *z/OS DFSMS Access Method Services* (ALLOCATE and DEFINE CLUSTER command sections) for information about the attributes that can be assigned through the SMS class parameters, and examples of defining data sets.

Another way to allocate a data set without specifying all of the data set attributes normally required is to model the data set after an existing data set. You can do this by referring to the existing data set in the DD statement for the new data set, using the JCL keywords LIKE or REFDD. See *z/OS MVS JCL Reference* and *z/OS MVS JCL User's Guide*.

Allocating Data Sets

Allocation has two related meanings:

1. Setting aside DASD space for a new data set.
2. Connecting a program to a new or existing data set or to a device.

   The program can then use the DD name to issue an OPEN macro and use an access method to read or write data.

To allocate a new data set on DASD, you can use any of the following methods:

- JCL DD statements. See *z/OS MVS JCL Reference*.
- Access method services ALLOCATE command or DEFINE command. See *z/OS DFSMS Access Method Services* for the syntax and more examples.
- TSO ALLOCATE command. See *z/OS TSO/E Command Reference* for the syntax and more examples.
- DYNALLOC macro using the SVC 99 parameter list. See *z/OS MVS Programming: Authorized Assembler Services Guide*.

There are two ways to cause a new data set to be system-managed:

- Specify the SMS parameter STORCLAS explicitly. You can also specify MGMTCLAS and DATACLAS.
- Let the ACS routines assign the SMS classes to the data set.

To allocate non-system-managed data sets, you can specify the DATACLAS parameter. Do not specify the MGMTCLAS and STORCLAS parameters.

Allocating Data Sets with JCL

To allocate a new data set using JCL, specify DISP=(NEW,CATLG,DELETE). If the application program completes normally, the data set is cataloged, but if the application program fails, the data set is deleted. All system-managed data sets are automatically cataloged, even if you use a disposition of KEEP.

To update an existing data set, specify a DISP value of OLD, MOD, or SHR. Do not use DISP=SHR while updating a sequential data set unless you have some other means of serialization because you might lose data.

To share a data set during access, specify a DISP value of SHR. If a DISP value of NEW, OLD, or MOD is specified, the data set cannot be shared.

Note: If SMS is active and a new data set is a type that SMS can manage, it is impossible to determine if the data set will be system-managed based solely on the JCL because an ACS routine can assign a storage class to any data set.

Allocating an HFS Data Set

An HFS data set is allocated if the DSNTYPE value is HFS and the SPACE parameter specifies the number of directory blocks, in either JCL or the data class. You must specify the number of directory blocks for an HFS data set, but the value has no effect on allocation.

If the DSNTYPE value is PIPE and the PATH parameter is also specified, the result is a first-in-first-out (FIFO) special file defined in an z/OS UNIX file system. IBM provides HFS data sets as one type of file system that supports FIFO special files. See “Chapter 6. Using Hierarchical File System Data Sets” on page 71.

The following example creates an SMS-managed HFS data set:

```
//FSJOB   JOB
//STEP1   EXEC PGM=IEFBR14
//MKFS1   DD   DSNAME=FILE.SYSTEM.FS0001,DISP=(NEW,KEEP),
//             DSNTYPE=HFS,DATACLAS=FILESYS,
//             MGMTCLAS=NEVER,STORCLAS=SECURE
```

The following example creates a non-SMS-managed HFS data set:

```
//FSJOB   JOB
//STEP1   EXEC PGM=IEFBR14
//MKFS1   DD   DSNAME=FILE.SYSTEM.FS0001,DISP=(NEW,CATLG),
//             DSNTYPE=HFS,SPACE=(CYL,(100,100,1)),DATACLAS=FILESYS,
//             MGMTCLAS=NEVER,VOL=SER=XXXXXX,UNIT=SYSDA
```

Allocating System-Managed Data Sets

Allocating a new data set under SMS, using the ACS routines defined by your storage administrator, is easier than without SMS. With SMS it is unnecessary to specify the UNIT, VOL=SER, SPACE, or the DCB parameters in the DD statement. For this allocation to succeed, the ACS routines must select a data class that defines the space and data set attributes required by the application.

You can request the name of the data class, storage class, and management class in the JCL DD statement. However, in most cases, the ACS routines pick the classes needed for the data set.

Allocating a PDSE. The DSNTYPE parameter determines if the data set is allocated as a PDSE or as a PDS. A DSNTYPE of LIBRARY causes the data set to be a PDSE. The DSNTYPE parameter can be specified in the JCL DD statement, the data class, or the system default, or by using the JCL LIKE keyword to refer to an existing PDSE.

If the SPACE parameter is omitted in the DD statement, it must be supplied by the data class. You can omit STORCLAS and DATACLAS in the DD statement if the default storage class and data class contain the data set attributes you want. A PDSE also can be allocated using access method services.

When first allocated, the PDSE is neither a program library or a data library. If the first member written, by either the binder or by IEBCOPY, is a program object, the library becomes a program library and remains such until the last member has been deleted. If the first member written is not a program object, then the PDSE becomes a data library. Program objects and other types of data cannot be mixed in the same PDSE library.

Allocating an Extended-Format Data Set. Extended format data sets must be system-managed. The mechanism for requesting extended format is through the SMS data class DSNTYPE=EXT parameter and subparameters R (required) or P (preferred). The storage administrator can specify R to ensure the data set is extended. Then, for VSAM data sets the storage administrator can set the Extended Addressability attribute to Y to request extended addressability.

In addition to a DSNTYPE of EXTENDED, COMPACTION=YES in a data class definition must be specified if you want to request allocation of an extended format data set in the compressed format. A compressed data set can be created using the LIKE keyword on the DD statement and not just through a data class.

Allocating a VSAM Data Set. See “Chapter 19. Using Job Control Language for VSAM” on page 265 for information about allocating VSAM data sets using JCL.

Allocating Non-System-Managed Data Sets

If your installation is running SMS, you can use a data class with a non-system-managed data set, such as a tape data set. The DCB information defined in the data class is used as the default. If you do not use a data class, you need to supply in the JCL, or in the program, the DCB information such as LRECL and RECFM, and the DSORG. You cannot use data class if your installation is not running SMS.

Allocating System-Managed Data Sets with the Access Method Services ALLOCATE Command

The examples in the following sections show you how to allocate a new data set to the job step using the access method services ALLOCATE command. See *z/OS DFSMS Access Method Services* (ALLOCATE section).

Allocating a Data Set Using Class Specifications

In the following example, the ALLOCATE command is used to allocate a new data set using the SMS classes. SMS must be active. The data set can be VSAM or non-VSAM.

```
//ALLOC    JOB  ...
//STEP1    EXEC PGM=IDCAMS,DYNAMNBR=1
//SYSPRINT DD   SYSOUT=A
//SYSIN    DD   *
           ALLOC -
              DSNAME(ALX.ALLOCATE.EXAMP1) -
              NEW CATALOG -
              DATACLAS(STANDARD) -
              STORCLAS(FAST) -
              MGMTCLAS(VSAM)
/*
```

The command parameters follow:

- DSNAME specifies the name of the data set being allocated is ALX.ALLOCATE.EXAMP1.
- NEW specifies the data set being allocated is new.

Allocating a VSAM Data Set Using Class Specifications

The following example uses the ALLOCATE command to allocate a new VSAM data set. Data class is not assigned, and attributes assigned through the default data class will be overridden by explicitly specified parameters:

```
//ALLOC    JOB  ...
//STEP1    EXEC PGM=IDCAMS,DYNAMNBR=1
//SYSPRINT DD   SYSOUT=A
//SYSIN    DD   *
           ALLOC -
              DSNAME(ALX.ALLOCATE.EXAMP2) -
              NEW CATALOG -
              SPACE(10,2) -
              AVBLOCK(80) -
              AVGREC(K) -
              LRECL(80) -
              RECORG(ES) -
              STORCLAS(FAST) -
              MGMTCLAS(VSAM)
/*
```

Allocating a System-Managed Non-VSAM Data Set

The following example uses the ALLOCATE command to allocate a non-VSAM data set. ALLOCATE, unlike DEFINE NONVSAM, lets you specify the SMS classes for a non-VSAM data set:

```
//ALLOC    JOB  ...
//STEP1    EXEC PGM=IDCAMS
//SYSPRINT DD   SYSOUT=A
//SYSABEND DD   SYSOUT=A
//SYSIN    DD   *
           ALLOC -
              DSNAME(NONVSAM.EXAMPLE) -
              NEW -
              DATACLAS(PS000000) -
              MGMTCLAS(S1P01M01) -
              STORCLAS(S1P01S01)
/*
```

Allocating a PDSE

The following example shows the ALLOCATE command used with the DSNTYPE keyword to create a PDSE. DSNTYPE(LIBRARY) indicates the data set being allocated is a PDSE.

```
//ALLOC    EXEC  PGM=IDCAMS,DYNAMNBR=1
//SYSPRINT DD    SYSOUT=A
//SYSIN    DD    *
           ALLOC -
              DSNAME(XMP.ALLOCATE.EXAMPLE1) -
              NEW -
              STORCLAS(SC06) -
              MGMTCLAS(MC06) -
              DSNTYPE(LIBRARY)
/*
```

Allocating a New Non-System-Managed Data Set

The following example uses the ALLOCATE command to allocate a new data set:

```
//ALLOC    JOB  ...
//STEP1    EXEC PGM=IDCAMS,DYNAMNBR=1
//SYSPRINT DD   SYSOUT=A
//SYSIN    DD   *
           ALLOC -
              DSNAME(XMP.ALLOCATE.EXAMP3) -
              NEW CATALOG -
              SPACE(10,5) TRACKS -
              BLKSIZE(1000) -
              LRECL(100) -
              DSORG(PS) -
              UNIT(3380) -
              VOL(338002) -
              RECFM(F,B)
/*
```

Allocating Data Sets with the TSO ALLOCATE Command

The following example allocates a new sequential data set with space allocated in tracks:

```
ALLOC DA(EX1.DATA) DSORG(PS) SPACE(2,0) TRACKS LRECL(80) RECFM(F,B) NEW
```

```
The new data set name: GOLD.EX1.DATA
The number of tracks:  2
The logical record length: 80
The block size: determined by the system
The record format: fixed block
```

You do not have to specify the user `ID`, `GOLD`, as an explicit qualifier. Because the BLKSIZE parameter is omitted, the system determines a block size that optimizes space usage.

The following example allocates a new VSAM entry-sequenced data set, with a logical record length of 80, a block size of 8000, on two tracks. To allocate a VSAM data set, specify the RECORG keyword on the ALLOCATE command. RECORG is mutually exclusive with DSORG and with RECFM. To allocate a key-sequenced data set, you also must specify the KEYLEN parameter. RECORG specifies the type of data set you want.

```
ALLOC DA(EX2.DATA) RECORG(ES) SPACE(2,0) TRACKS LRECL(80)
BLKSIZE(8192) NEW
```

ES—Entry-sequenced data set
KS—Key-sequenced data set
LS—Linear data set
RR—Relative record data set

Allocating Data Sets with Dynamic Allocation

See *z/OS MVS Programming: Authorized Assembler Services Guide* for examples of allocating a data set using the DYNALLOC macro. To allocate a VSAM data set using the DYNALLOC macro with the SVC 99 parameter list, specify text unit 800B - RECORG.

Chapter 3. Allocating Space on Direct Access Volumes

This chapter covers the following topics.

Topic	Page
Specification of Space Requirements	37
Maximum Data Set Size	39
Primary and Secondary Space Allocation without the Guaranteed Space Attribute	40
Allocation of Data Sets with the Guaranteed Space Attribute	41
Allocation of Data Sets with the Space Constraint Relief Attributes	42
Extension to Another DASD Volume	43
Multiple Volume Considerations for Sequential Data Sets	45
Additional Information on Space Allocation	45

Specification of Space Requirements

You can specify the amount of space required in blocks, records, tracks, or cylinders. When creating a DASD data set, specify the amount of space needed explicitly by using the SPACE parameter, or specify the amount of space implicitly by using the information available in a data class. The data class is not used if SMS is inactive at the time of your allocation.

The system can use a data class if SMS is active even if the data set is not SMS managed. For system-managed data sets, the system selects the volumes. Therefore, you do not need to specify a volume when you define your data set.

If you specify your space request by average record length, space allocation is independent of device type. Device independence is especially important to system-managed storage.

Blocks

When the amount of space required is expressed in blocks, you must specify the number and average length of the blocks within the data set, as in this example:

```
//  DD SPACE=(300,(5000,100)), ...

       300 = average block length in bytes
      5000 = primary quantity (number of blocks)
       100 = secondary quantity, to be allocated if the primary
             quantity is not enough (in blocks)
```

From this information, the operating system estimates and allocates the number of tracks required.

Recommendation: IBM recommends, for sequential and partitioned data sets, to let the system calculate the block size and that you not request space by average block length. See “System-Determined Block Size” on page 324.

If the average block length of the real data does not match the value coded here, the system might allocate much too little or much too much space.

© Copyright IBM Corp. 1987, 2002

Average Record Length

When the amount of space required is expressed in average record length, you must specify the number of records within the data set and their average length. Use the AVGREC keyword to modify the scale of your request. When AVGREC is specified, the first subparameter of SPACE becomes the average record length. The system applies the scale value to the primary and secondary quantities specified for the SPACE keyword. Possible values for the AVGREC keyword follow:

U—Use a scale of 1
K—Use a scale of 1024
M—Use a scale of 1048576

When the AVGREC keyword is specified, the values specified for primary and secondary quantities in the SPACE keyword are multiplied by the scale and those new values will be used in the space allocation. For example, the following request results in the primary and secondary quantities being multiplied by 1024:

```
//  DD SPACE=(80,(20,2)),AVGREC=K, ...

                     80  = average record length in bytes
 80 * 20 * 1024 = 1.6 MB  = primary space
 80 *  2 * 1024 = 160 KB  = secondary space, to be allocated if the
                            primary space is not enough
```

From this information, the operating system estimates and allocates the number of tracks required using one of the following block lengths, in the order indicated:

1. 4096, if the data set is a PDSE.
2. The BLKSIZE parameter on the DD statement or the BLKSIZE subparameter of the DCB parameter on the DD.
3. The system determined block size, if available.
4. A default value of 4096.

For an extended-format data set, the operating system uses a value 32 larger than the above block size. The primary and secondary space are divided by the block length to determine the number of blocks requested. The operating system determines how many blocks of the block length can be written on one track of the device. The primary and secondary space in blocks is then divided by the number of blocks per track to obtain a track value, as shown in the examples below. These examples assume a block length of 23200. Two blocks of block length 23200 can be written on a 3380 device:

```
(1.6MB / 23200) / 2 = 36 = primary space in tracks
(160KB / 23200) / 2 =  4 = secondary space in tracks
```

In the preceding calculations, the system does not consider if it is a compressed format data set. This means the calculation is done with the user-perceived uncompressed block size and not the actual block size that the system calculates.

Tracks or Cylinders

The following example shows the amount of space required in tracks or cylinders:

```
//  DD SPACE=(TRK,(100,5)), ...
//  DD SPACE=(CYL,(3,1)), ...
```

Absolute Track

If the data set contains location-dependent information in the form of an actual track address (such as MBBCCHHR or CCHHR), you can request space in the number of

tracks and the beginning address. In this example, 500 tracks is required, beginning at relative track 15, which is cylinder 1, track 0:

```
//  DD SPACE=(ABSTR,(500,15)),UNIT=3380, ...
```

Restriction: Data sets that request space by absolute track are not eligible to be system managed and they interfere with DASD space management done by the space management products and the storage administrator. Avoid using absolute track allocation.

Additional Space-Allocation Options

The DD statement provides flexibility in specifying space requirements. See *z/OS MVS JCL Reference* about option information.

Maximum Data Set Size

This section contains information about the following maximum amounts for data sets:

- Maximum size on one volume
- Maximum number of volumes
- Maximum size for a VSAM data set

Maximum Size on One Volume

Most types of data sets are limited to 65 535 total tracks allocated on any one volume, and if a greater number of tracks is required, this and subsequent attempts to create data sets will fail

Data sets that are not limited to 65 535 total tracks allocated on any one volume are:

- Extended-format sequential
- HFS
- PDSE
- VSAM

If a VIO data set is to be SMS managed, the maximum VIO maximum size is 2 000 000 KB, as defined in the Storage Group Vio Maxsize parameter.

Maximum Number of Volumes

PDS and PDSE data sets are limited to one volume. Extended-format sequential data sets with multiple stripes are limited to 16 volumes. A data set on a virtual input-output (VIO) simulated device is limited to 65 535 tracks and is on one volume. All other DASD data sets are limited to 59 volumes. Tape data sets are limited to 255 volumes.

Maximum VSAM Data Set Size

A VSAM data set is limited to 4 GB across all volumes unless Extended Addressability is specified in the SMS data class definition. System requirements restrict the number of volumes that can be used for one data set to 59.

Primary and Secondary Space Allocation without the Guaranteed Space Attribute

Space is allocated for non-system-managed data sets or system-managed data sets without the guaranteed space attribute in the storage class as follows. If you allocate a new data set and specify SPACE=(TRK,(2,4)); this initially allocates two tracks for the data set. As each record is written to the data set and these two tracks are used up, the system automatically obtains four more tracks. When these four tracks are used, another four tracks are obtained. The same sequence is followed until the extent limit for the type of data set is reached.

- A physical sequential data set can have 16 extents on each volume. An extended-format data set can have 123 extents per volume. Extended-format sequential data sets are also physical sequential data sets, and must have DSORG=PS.
- A PDS can have 16 extents.
- A direct data set can have 16 extents on each volume.
- A VSAM data set can have up to 255 extents per component.
- A striped VSAM data set can have up to 255 extents per stripe.
- A PDSE can have 123 extents.
- An HFS data set (file system) can have 123 extents on each volume.

You can allocate space for a multivolume data set the same as for a single volume data set. DASD space is initially allocated on the first volume only (exceptions are striped extended-format data sets and guaranteed space data sets). When the primary allocation of space is filled, space is allocated in secondary storage amounts (if specified). The extents can be allocated on other volumes. VIO space allocation is handled differently from other data sets. See “Virtual I/O for Temporary Data Sets” on page 22.

Multivolume VSAM Data Sets

When a multivolume VSAM data set extends to the next volume, the data class specifies if the initial space allocated on that volume is the primary or secondary amount. The default is the primary amount. After the primary amount of space is used up, space is allocated in secondary amounts. By using a data class, it is possible to indicate whether to take a primary or secondary amount when VSAM extends to a new volume. The previous comments do not pertain to VSAM data that is striped. See “Chapter 7. Organizing VSAM Data Sets” on page 75 about VSAM data in the striped format.

Multivolume Non-VSAM Data Sets

When a multivolume non-VSAM, non-extended-format data set extends to the next volume, the initial space allocated on that volume is the secondary amount.

Extended-Format Data Sets

When space for a striped extended-format data set is allocated, the system divides the primary amount among the volumes. If it does not divide evenly, the system rounds the amount up. For extended-format data sets, when the primary space on any volume is filled, the system allocates space on that volume. The amount is the secondary amount divided by the number of stripes. If the secondary amount cannot be divided evenly, the system rounds up the amount.

Data sets allocated in the extended-format achieve the added benefits of improved error detection when writing to DASD as well as the use of a more efficient and

functionally complete interface to the I/O subsystem. In addition, a VSAM data set must be in this format to be eligible for the following advanced functions:
- Data compression
- Data striping

VSAM data sets must also be in extended-format to be eligible for the following advanced functions:
- Partial space release (PARTREL)
- Candidate volume space
- Extended addressability (>4 GB)
- System-managed buffering (SMB)

Allocation of Data Sets with the Guaranteed Space Attribute

You can allocate space and load a guaranteed space data set in one step or in separate steps.

Guaranteed Space with DISP=NEW or MOD

When you code DISP=NEW or DISP=MOD, space is allocated to system-managed multivolume (non-extended-format) data sets with the guaranteed space attribute in the storage class, as follows:

1. Initially, primary space is preallocated on all the volumes.
2. When the primary amount on the first volume is used up, a secondary amount is allocated on the first volume until the volume is out of space or the data set has reached its extent limit.
3. The preallocated primary space on the next volume is then used.
4. When the primary space on the next volume is used up, a secondary amount is allocated.
5. Secondary amounts continue to be allocated until the volume is out of space or the data set extent limit is reached.

The same sequence is followed for all succeeding volumes.

Guaranteed Space with DISP=OLD or SHR

When you code DISP=OLD or DISP=SHR, space is allocated to system-managed multivolume (non-extended-format) data sets with the guaranteed space attribute in the storage class, as follows:

1. Initially, the system preallocated primary space on all the volumes when you coded DISP=NEW.
2. When the allocated space on each volume is used up, the system switches to the next volume. Some volumes might already have secondary space allocations because you extended the data set when you coded DISP=NEW or DISP=MOD earlier. The system will use those secondary allocations.
3. The existing space on the next volume is then used.
4. The system will attempt to allocate new space only on the last volume. On that volume secondary amounts continue to be allocated until the volume is out of space or the data set extent limit is reached.

The system works this way so that it is similar to nonguaranteed preallocated space on non-SMS volumes.

Guaranteed Space with Extended-Format Data Sets

When guaranteed space is specified for a multivolume extended-format sequential data set, the primary space is preallocated on all the volumes. Because data is written to an extended-format data set using data striping (logically consecutive tracks or CIs are written to the data set in a circular manner), secondary space is not allocated until the preallocated primary space on all volumes is used up. For a striped VSAM data set with guaranteed space that has more than 16 volumes, only the first 16 volumes have preallocated space. The secondary amount specified is divided by the number of volumes and rounded up for allocation on each volume.

The amount of preallocated space for VSAM striped data is limited to 16 volumes.

Guaranteed Space Example

The following example allocates 100 MB of primary space on each of five volumes:

```
//DD1  DD  DSN=ENG.MULTIFILE,DISP=(,KEEP),STORCLAS=GS,
//         SPACE=(1,(100,25)),AVGREC=M,UNIT=(3380,5)
```

1. After 100 MB is used on the first volume, 25 MB extents of secondary space is allocated on it until the extent limit is reached or the volume is full. The system assumes DISP=NEW because the user omitted the first DISP value.
2. If more space is needed, the 100 MB of preallocated primary space is used on the second volume. Then, more secondary space is allocated on that volume.
3. The same process is repeated on each volume.

Allocation of Data Sets with the Space Constraint Relief Attributes

To reduce allocation failures, three data class attributes can influence the allocation and extension of data sets to new volumes. Allocations that might have failed for lack of space can succeed.

These three attributes follow:

- Space Constraint Relief (values are YES or NO)
- Reduce Space Up To % (0 - 99%)
- Dynamic Volume Count (1 - 59 or blank)

Allocations and extends to new volumes proceed normally until it is established that space cannot be obtained by normal means.

Restriction: Space constraint relief is not supported for VSAM striped components.

Space constraint relief, if requested, occurs in one or two steps, depending on the volume count specified for the failing allocation.

Step 1. If the volume count is greater than 1, SMS attempts to satisfy the allocation by spreading the requested primary allocation over more than one volume, but no more than the volume count specified.

Step 2. If Step 1 also fails or if the volume count specified was 1, in which case Step 1 would not be entered, then SMS takes two actions simultaneously. First, it will modify the requested primary space, or secondary space for extension, by the percentage specified in the REDUCE SPACE UP TO parameter. Note that the user can specify 0% in the data class for this parameter so space is not reduced. Second, SMS removes the 5 extent limit (for example, for physical sequential data sets as many as 16 extents can be used to satisfy the allocation).

The allocation will fail as before if either or both Step 1 and 2 are not successful.

Extension to Another DASD Volume

The system attempts to extend to another DASD volume if all of the following conditions exist:

- The current volume does not have the secondary space amount available, or the data set reached the extent limit for that type of data set, or the application program issued the FEOV macro.
- The volume count has not yet been reached. For a system-managed data set, the volume count was determined when the data set was created (DISP=NEW) and is the largest of the volume count (VOL=(,,,*nn*)), the number of volumes coded with the VOL parameter, and the unit count (UNIT=(*xxxx*,*nn*)).

  For either system-managed or non-system-managed data sets, the volume count can come from the data class. The volume count can be increased after data set creation with the IDCAMS ADDVOL command.
- The dynamic volume count has not been reached for a system-managed data set. You can use the Space Constraint Relief and Dynamic Volume Count data set attributes to specify a dynamic volume count in ISMF. If Space Constraint Relief=YES, you can specify a dynamic volume count from 1 to 59 for SMS to extend a data set automatically to another volume or volumes. For more information, see *z/OS DFSMSdss Storage Administration Guide*.
- A secondary allocation amount is available. This can come from the data class when the data set was created even if the data set is not system managed. When the data set is created with DISP=NEW or IDCAMS DEFINE, the secondary amount permanently overrides data class for the data set. For a non-VSAM data set the above two sources of the secondary amount can be overridden temporarily with DISP other than NEW.

Examples of Dynamic Volume Count When Defining a Data Set

In any of the following examples, ″Volume Count″ can be the Volume Count attribute from the data class or volume count information from a JCL, TSO, or IDCAMS command. Using the dynamic volume count causes the number of primary volumes to increase, if necessary. The use of dynamic volume count does not add any candidate volumes to the catalog.

Example 1

```
Volume count: 6
Dynamic volume count: 0
Required volumes: 1
Volumes in catalog: 1 primary, 5 candidates
```

Example 2

```
Volume count: 6
Dynamic volume count: 12
Required volumes: 1
Volumes in catalog: 1 primary, 5 candidates
```

Example 3

```
Volume Count: 6
Dynamic volume count:  12
Required volumes: 7
Volumes in catalog: 7 primary, 0 candidates
```

Example 4

```
Volume count: 6
Dynamic volume count:  12
Required volumes: 13
Volumes in catalog: None; request fails
```

Examples of Dynamic Volume Count When Allocating an Existing Data Set

In the following examples, a list of specific volumes is returned to allocation, as well as a count of nonspecific volumes. Also, the allocation in Example 7 fails because the total volume count exceeds the supported limit of 59 volumes.

Example 5: VSAM KSDS

```
Specific volume count: 2
Nonspecific volume count: 4
Cluster dynamic volume count: 20
Specific volumes returned to allocation:   2
Nonspecific volumes returned to allocation: 18
Total count of volumes returned to allocation: 20
```

Example 6: VSAM Path

```
Specific volume count (base cluster): 5
Nonspecific volume count (base cluster): 1
Specific and total volume count (alternate index): 1
Base cluster dynamic volume count: 20
Specific volumes returned to allocation:   6
Nonspecific volumes returned to allocation: 15
Total count of volumes returned to allocation: 21
```

Example 7: Alternate Indexes in Upgrade Set

```
Specific and total volume count (base cluster): 5
Specific and total volume count (first alternate index): 1
Specific and total volume count (second alternate index): 1
Base cluster dynamic volume count: 59
Specific volumes returned to allocation:   7
Nonspecific volumes returned to allocation: 54
Total count of volumes returned to allocation: 61
```

Multiple Volume Considerations for Sequential Data Sets

Consider the following when working with multiple volumes: Your program is extending a sequential data set if it uses the EXTEND or OUTINX option of OPEN or it uses the OUTPUT or OUTIN option of OPEN with DISP=MOD on the DD statement. If you plan to rewrite a multivolume sequential data set that is not SMS managed, and you might later extend the data set, you should delete and reallocate the data set. This avoids the problems described in item 2 below and the system will extend on the volume that you want.

1. When writing to a sequential data set, EOV turns off the last volume bit as it finishes each volume and CLOSE turns on the last volume bit in the DSCB on the current volume. It identifies the last volume containing data, not necessarily the last volume allocated to the data set. The DSCB on a later volume can also have this bit on, either due to earlier writings or due to guaranteed space.
2. Writing with the DISP=MOD, OPEN EXTEND, or OUTINX option works differently with system-managed and non-system-managed data sets. On system-managed volumes, OPEN determines where to start writing using the following algorithm. No matter which volume you finish writing on, OPEN will find that volume to resume. Starting with the first volume, OPEN searches for the last volume bit. The first new block will be written immediately after the previous last block. If the old last block was short, it does not get larger. This specifically applies to SMS volumes.

   Writing with the DISP=MOD, OPEN EXTEND, or OUTINX option on non-SMS volumes, OPEN determines where to start writing using the following algorithm: It looks first on the last volume in the JFCB or its extensions to see if its DSCB has the last volume bit ON. If it is not ON, OPEN searches the other volumes in order starting with the first volume. This means that if the last volume and an earlier volume each have the last volume bit ON, your added data will not be reachable when reading sequentially.

   For striped data sets, which are SMS only, the last volume bit works a little different but it has the same effect as for other SMS data sets. The bit is ON on the last volume, even if that volume does not contain the last record of the data set. OPEN uses the DS1LSTAR fields to calculate the volume containing the last record.
3. With partial release, CLOSE releases the unused space only on the current volume except with a striped data set. It does not release space on later volumes that can contain data either from a prior writing or due to guaranteed space. With a system-managed data sets this has no effect on a later use of DISP=MOD but it does mean that the space on the later volumes can be there due to guaranteed space allocation.

Additional Information on Space Allocation

If you want to know how many DASD tracks your data set requires, see the appropriate device book. See *MVS/ESA SML: Managing Data* for information about allocating space for system-managed data sets. See “Allocating Space for a PDS” on page 409 and “Allocating Space for a PDSE” on page 437 for information about PDS/PDSE space allocation.

Chapter 4. Backing Up and Recovering Data Sets

This chapter covers the following topics.

Topic	Page
Using REPRO for Backup and Recovery	48
Using EXPORT and IMPORT for Backup and Recovery of VSAM Data Sets	50
Writing a Program for Backup and Recovery	51
Using Concurrent Copy for Backup and Recovery	51
Updating a Data Set After Recovery	51
Synchronizing Catalog and VSAM Data Set Information During Recovery	52

It is important to establish backup and recovery procedures for data sets so you can replace a destroyed or damaged data set with its backup copy. Generally data administrators set up automated procedures for backup so you do not have to be concerned with doing it yourself. SMS facilitates this automation by means of management class.

There are several methods of backing up and recovering VSAM and non-VSAM data sets:

- Using Data Facility Storage Management Subsystem Hierarchical Storage Manager (DFSMShsm). You can use DFSMShsm only if DSS and DFSMShsm are installed on your system and your data sets are cataloged in a catalog. See *z/OS DFSMShsm Managing Your Own Data* for information about using DFSMShsm backup and recovery.
- Using the access method services REPRO command.
- Using the Data Facility Storage Management Subsystem Data Set Services (DFSMSdss) DUMP and RESTORE commands. You can use DSS if it is installed on your system and your data sets are cataloged in a catalog. For uncataloged data sets, DSS provides full volume, and physical or logical data set dump functions. For compressed extended format data sets, DFSMShsm processes the compressed data sets using DFSMSdss as the data mover. When using DFSMSdss for logical dump/restore with VSAM compressed data sets, the target data set allocation must be consistent with the source data set allocation. For DFSMShsm, a VSAM extended format data set migrated and/or backed up will only be recalled and/or recovered as an extended format data set. See *z/OS DFSMSdss Storage Administration Reference* for information about using DSS.
- Writing your own program for backup and recovery.
- For VSAM data sets, using the access method services EXPORT and IMPORT commands.
- For PDSs using IEBCOPY utility.
- Using concurrent copy to take an instantaneous copy. You can use concurrent copy if your data set resides on DASD attached to IBM storage controls that support the concurrent copy function.

Each of these methods of backup and recovery has its advantages. You need to decide the best method for the particular data you want to back up. For the requirements and processes of archiving, backing up, and recovering data sets using DFSMShsm, DSS, or ISMF, see *MVS/ESA SML: Managing Data*, which also contains information on disaster recovery.

© Copyright IBM Corp. 1987, 2002

Using REPRO for Backup and Recovery

Use the REPRO command to create a duplicate data set for back up. See “Copying and Merging Data Sets” on page 119 for information about using REPRO.

Using REPRO for backup and recovery has the following advantages:

- **Backup copy is accessible.** The backup copy obtained by using REPRO is accessible for processing. It can be a VSAM data set or a sequential data set.
- **Type of data set can be changed.** The backup copy obtained by using REPRO can be a different type of data set than the original. For example, you could back up a VSAM key-sequenced data set by copying it to a VSAM entry-sequenced data set. A compressed VSAM key-sequenced data set cannot be copied to a VSAM entry-sequenced data set using REPRO. The data component of a compressed key-sequenced data set cannot be accessed by itself.
- **Key-sequenced data set or variable-length RRDS is reorganized.** Using REPRO for backup results in data reorganization and the recreation of an index for a key-sequenced data set or variable-length RRDS. The data records are rearranged physically in ascending key sequence and free-space quantities are restored. (Control interval and control area splits can have placed the records physically out of order.) When a key-sequenced data set is reorganized, absolute references using the relative byte address (RBA) become invalid.

If you are accessing a data set using RLS, see “Chapter 15. Using VSAM Record-Level Sharing” on page 223.

REPRO provides you with several options for creating backup copies and using them for data set recovery. The following are suggested ways to use REPRO:

1. Use REPRO to copy the data set to a data set with a different name.

   To use the copy either change your references to the original copy or delete the original and rename the copy.
2. Create a backup copy on another catalog, then use the backup copy to replace the original.
   - To create a backup copy, define a data set on another catalog, and use REPRO to copy the original data set into the new data set you have defined.
   - Because a backup copy created by REPRO is accessible for processing, when you want to replace the original with the backup copy, you can leave the backup copy on the catalog it was copied to. If you do this, the JOBCAT and STEPCAT statements in the JCL must be changed to reflect the name of the catalog containing the backup copy.

     JOBCAT and STEPCAT DD statements are not supported for system-managed data sets. The JOBCAT or STEPCAT DD statement will fail if it references a system-managed catalog, or if the data set being searched is system managed. Also, all referenced catalogs must be connected to the system master catalog.

3. Create a copy of a nonreusable VSAM data set on the same catalog, then delete the original data set, define a new data set, and load the backup copy into the newly defined data set.
   - To create a backup copy, define a data set, and use REPRO to copy the original data set into the newly defined data set. If you define the backup data set on the same catalog as the original data set or if the data set is SMS managed, the backup data set must have a different name.
   - To recover the data set, use the DELETE command to delete the original data set if it still exists. Next, redefine the data set using the DEFINE command, then restore it with the backup copy using the REPRO command.
4. Create a copy of a reusable VSAM data set, then load the backup copy into the original data set. When using REPRO, the REUSE attribute permits repeated backups to the same VSAM reusable target data set.
   - To create a backup copy, define a data set, and use REPRO to copy the original reusable data set into the newly defined data set.
   - To recover the data set, load the backup copy into the original reusable data set.
5. Create a backup copy of a data set, then merge the backup copy with the damaged data set. When using REPRO, the REPLACE parameter lets you merge a backup copy into the damaged data set. You cannot use the REPLACE parameter with entry-sequenced data sets, because records are always added to the end of an entry-sequenced data set.
   - To create a backup copy, define a data set, and use REPRO to copy the original data set into the newly defined data set.
   - To recover the data set, use the REPRO command with the REPLACE parameter to merge the backup copy with the destroyed data set. With a key-sequenced data set, each source record whose key matches a target record's key replaces the target record. Otherwise, the source record is inserted into its appropriate place in the target cluster. With a fixed-length or variable-length RRDS, each source record, whose relative record number identifies a data record in the target data set, replaces the target record. Otherwise, the source record is inserted into the empty slot its relative record number identifies. When only part of a data set is damaged, you can replace only the records in the damaged part of the data set. The REPRO command lets you specify a location to begin copying and a location to end copying.
6. If the index of a key-sequenced data set or variable-length RRDS becomes damaged, follow this procedure to rebuild the index and recover the data set. This does not apply to a compressed key-sequenced data set. It is not possible to REPRO just the data component of a compressed key-sequenced data set.
   - Use REPRO to copy the data component only. Sort the data.
   - Use REPRO with the REPLACE parameter to copy the cluster and rebuild the index.

Using EXPORT and IMPORT for Backup and Recovery of VSAM Data Sets

Using EXPORT/IMPORT for backup and recovery has the following advantages:

- **Key-sequenced data set or variable-length RRDS is reorganized.** Using EXPORT for backup results in data reorganization and the recreation of an index for a key-sequenced data set or variable-length RRDS. The data records are rearranged physically in ascending key sequence and free-space quantities are balanced. (Control interval and control area splits can have placed the records physically out of order.) When a key-sequenced data set is reorganized, absolute references using the RBA become invalid.
- **Redefinition is easy.** Because most catalog information is exported along with the data set, you are not required to define a data set before importing the copy. The IMPORT command deletes the original copy, defines the new object, and copies the data from the exported copy into the newly defined data set.
- **Attributes can be changed or added.** When you IMPORT a data set for recovery, you can specify the OBJECTS parameter to show new or changed attributes for the data set. Importing a data set lets you change the name of the data set, the key ranges, the volumes on which the data set is to reside, and the SMS classes. See “Chapter 15. Using VSAM Record-Level Sharing” on page 223 about accessing a data set using RLS.

Structure of an Exported Data Set

An exported data set is an unloaded copy of the data set. The backup copy can be only a sequential data set.

Most catalog information is exported along with the data set, easing the problem of redefinition. The backup copy contains all of the information necessary to redefine the VSAM cluster or alternate index when you IMPORT the copy.

EXPORT and IMPORT Commands

When you export a copy of a data set for backup, specify the TEMPORARY attribute. Exporting a data set means that the data set is not to be deleted from the original system.

You can export entry-sequenced or linear data set base clusters in control interval mode by specifying the CIMODE parameter. When CIMODE is forced for a linear data set, a RECORDMODE specification is overridden.

Use the IMPORT command to totally replace a VSAM cluster whose backup copy was built using the EXPORT command. The IMPORT command uses the backup copy to replace the cluster’s contents and catalog information.

You can protect an exported data set by specifying the INHIBITSOURCE or INHIBITTARGET parameters. Using these parameters means the source or target data set cannot be accessed for any operation other than retrieval.

IMPORT will not propagate distributed data management (DDM) attributes if you specify the INTOEMPTY parameter. Distributed file manager (DFM) will reestablish the DDM attributes when the imported data set is first accessed.

Compressed data must not be considered portable. IMPORT will not propagate extended format or compression information if the user specifies the INTOEMPTY parameter.

Writing a Program for Backup and Recovery

There are two methods of creating your own program for backup and recovery:

- If you periodically process a data set sequentially, you can easily create a backup copy as a by-product of normal processing. The backup copy can be used like one made by REPRO.
- You can write your own program to back up your data sets. Whenever possible, this program should be integrated into the regular processing procedures.

  In VSAM, the JRNAD user exit routine is one way to write your own backup program. When you request a record for update, VSAM calls the JRNAD exit routine to copy the record you are going to update, and write it to a different data set. When you return to VSAM, VSAM completes the requested update. If something goes wrong, you have a backup copy. See “JRNAD Exit Routine to Journalize Transactions” on page 245.

Using Concurrent Copy for Backup and Recovery

Concurrent copy takes what appears to be an instantaneous copy of data. The copy can be a backup copy (such as to tape) or for replicating a database from one set of DASD volumes to another. Concurrent copy also benefits the nondatabase environment by permitting a backup or copy occur with only a very short serialization.

Using concurrent copy for backup has the following advantages:

- It has little or no disruption.
- It is logically consistent.
- It is not necessary to take down the application using the data.
- It executes without regard to how the data is being used by the application.
- It works for any kind of DSS dump or copy operation.
- It eliminates the unavailability of DFSMShsm while control data sets are being backed up.

DFSMShsm can use concurrent copy to copy its own control data sets and journal.

Running concurrent copy (like any copy or backup) during off-peak hours results in better system throughput.

See *z/OS DFSMSdss Storage Administration Guide* for information about using concurrent copy.

Updating a Data Set After Recovery

After replacing a damaged data set with its backup copy, you can update the restored data set. To update the restored data set, rerun the jobs that updated the original between the time it was backed up and the time it became inaccessible.

Synchronizing Catalog and VSAM Data Set Information During Recovery

Because the physical and logical description of a VSAM data set is contained in its catalog entries, VSAM requires up-to-date catalog entries to access data sets. If either your data set or your catalog is damaged, your recovery procedure must match both data set and catalog entry status. Recovery by reloading the data set automatically takes care of this problem. A new catalog entry is built when the data set is reloaded.

Backing up the data sets in a user catalog lets you recover from damage to the catalog. You can import the backup copy of a data set whose entry is lost or you can redefine the entry and reload the backup copy.

See *z/OS DFSMS: Managing Catalogs* for information about backing up and recovering a catalog. Also see *z/OS DFSMShsm Managing Your Own Data*.

Handling an Abnormal Termination

When a user program closes a VSAM data set, the system uses the data set's end-of-data information to update its cataloged information. If a system failure occurs before the user program closes the data set, its cataloged information is not updated and any records in unwritten buffers are not written to the data set.

If an error occurs while a component is opened for update processing, it can improperly close (leaving the open-for-output indicator on). At OPEN, VSAM implicitly issues a VERIFY command when it detects an open-for-output indicator on and issues an informational message stating whether the VERIFY command is successful.

If a subsequent OPEN is issued for update, VSAM turns off the open-for-output indicator at CLOSE. If the data set is opened for input, however, VSAM leaves the open-for-output indicator on. It is the successful CLOSE after an OPEN for output that causes the open-for-output indicator to turn off. Before you use any data set that was not successfully closed, determine the status of the data in the data set. Turning off the open-for-output indicator in the catalog does not make the data set error free.

When the data set is subsequently opened and the user's program attempts to process records beyond end-of-data or end-of-key range, a read operation results in a "no record found" error, and a write operation might write records over previously written records. To avoid this, use the VERIFY command to correct the catalog information. See *z/OS DFSMS: Managing Catalogs* for information about recovering a data set.

Using VERIFY to Process Improperly Closed Data Sets

You can use a VSAM VERIFY macro call with certain types of opened VSAM data sets to ensure that various fields in the VSAM control blocks are accurate. The VERIFY macro does not change the data in the data set. VERIFY does not correct missing or duplicate records or repair any damage in the index structure. The verification of control-block fields enables you to perform recovery actions on the improperly closed data set, if necessary.

You can also use the IDCAMS VERIFY command to verify a VSAM data set. When you issue this command, IDCAMS opens the VSAM data set for output, issues a VSAM VERIFY macro call, and then closes the data set. The IDCAMS VERIFY command and the verification by VSAM OPEN are the same. Neither changes the data in the verified data set.

The catalog will be updated from the verified information in the VSAM control blocks when the VSAM data set which was opened for output is successfully closed.

The actual VSAM control-block fields that get updated depend on the type of data set being verified. VSAM control block fields that can be updated include “High used RBA/CI” for the data set, “High key RBA/CI”, “number of index levels”, and “RBA/CI of the first sequence set record”.

The VERIFY command should be used following a system failure that caused a component opened for update processing to be improperly closed. Clusters, alternate indexes, entry-sequenced data sets, and catalogs can be verified. Paths over an alternate index and linear data sets cannot be verified. Paths defined directly over a base cluster can be verified. The VERIFY macro will perform no function when VSAM RLS is being used. VSAM RLS is responsible for maintaining data set information in a shared environment.

Although the data and index components of a key-sequenced cluster or alternate index can be verified, the timestamps of the two components are different following the separate verifies, possibly causing further OPEN errors. Therefore, use the cluster or alternate index name as the target of your VERIFY command. You should issue the VERIFY command every time you open a VSAM cluster that is shared across systems. See “Cross-System Sharing” on page 204 for information about using VERIFY with clusters that are shared.

Recovering from Errors Due to an Improperly Closed VSAM Data Set

Sometimes a data set is closed properly, but an error occurred. The most likely error is an incorrect high RBA in the catalog. Other possible errors are an incomplete write to a DASD or duplicate data exists. One way to avoid these errors is by doing synchronous direct inserts. Another way is by using abnormal termination user exits in which you issue a CLOSE (perhaps with the TYPE=T parameter) to close the data set properly.

If you suspect that a write operation is incomplete, issue either an IMPORT or REPRO command to get an old copy of the data. Intermediate updates or inserts are lost. You must have an exported version of the data set available to use IMPORT. Use a backup copy for REPRO.

Duplicate data in a key-sequenced data set, the least likely error to occur, can result from a failure during a control interval or control area split. To reduce the number of splits, specify free space for both control intervals and control areas. If the failure occurred before the index was updated, the insert is lost, no duplicate exits, and the data set is usable.

If the failure occurred between updating the index and writing the updated control interval into secondary storage, some data is duplicated. However, you can access both versions of the data by using addressed processing. If you want the current version, use REPRO to copy it to a temporary data set and again to copy it back to a new key-sequenced data set. If you have an exported copy of the data, use the IMPORT command to obtain a reorganized data set without duplicate data.

If the index is replicated and the error occurred between the write operations for the index control intervals, but the output was not affected, both versions of the data can be retrieved. The sequence of operations for a control area split is similar to that for a control interval split. To recover the data, use the REPRO or IMPORT command in the same way as for the failure described in the previous paragraph.

Use the journal exit (JRNAD) to determine control interval and control area splits and the RBA range affected.

Using VERIFY with Catalogs

To use the VERIFY command to verify a catalog, access method services must be authorized. As supplied by IBM, VERIFY is authorized. See *z/OS MVS Programming: Authorized Assembler Services Guide* for information about program authorization.

You cannot use VERIFY to correct catalog records for a key-sequenced data set, or a fixed-length or variable-length RRDS after load-mode failure. An entry-sequenced data set defined with the RECOVERY attribute can be verified after a create (load) mode failure; however, you cannot run VERIFY against an empty data set or a linear data set. Any attempt to do either will result in a VSAM logical error. See “Opening a Data Set” on page 139 for information about VSAM issuing the implicit VERIFY command.

Chapter 5. Protecting Data Sets

You can prevent unauthorized access to payroll data, sales forecast data, and all other data sets that require special security attention. You can protect confidential data in a data set using Resource Access Control Facility (RACF) or passwords.

This chapter covers the following topics.

Topic	Page
z/OS Security Server (RACF)	55
Data Set Password Protection	57
User-Security-Verification Routine	62
Erasure of Residual Data	62
Authorized Program Facility and Access Method Services	64
Access Method Services Cryptographic Option	65

z/OS Security Server (RACF)

The z/OS Security Server is the primary tool that IBM recommends for managing security. Often the Security Server is called the Resource Access Control Facility (RACF). In the MVS environment, you can use RACF identify and verify users' authority to access data and to use system facilities. RACF protection can apply to a catalog and to individual VSAM data sets.

The system ignores password protection for SMS-managed data sets. See "Data Set Password Protection" on page 57.

If a discrete profile or a generic profile does not protect a data set, password protection is in effect.

For more information about RACF, see *MVS/ESA SML: Managing Data* and *z/OS Security Server RACF Security Administrator's Guide*.

RACF Protection for VSAM Data Sets

A catalog that contains a VSAM data set does not have to be RACF protected for its data sets to be RACF protected.

RACF and password protection can coexist for the same VSAM data set. The RACF authorization levels of alter, control, update, and read correspond to the VSAM password levels of master, control, update, and read.

To have password protection take effect for a non-system-managed data set, the catalog that contains the data set must be either RACF protected or password protected, and the data set itself must not be defined to RACF. Although passwords are not supported for an RACF-protected data set, they can still provide protection if the data set is moved to a system that does not have RACF protection.

If a user-security-verification routine (USVR) exists, it is not invoked for RACF-defined data sets.

© Copyright IBM Corp. 1987, 2002

Deleting any type of RACF-protected entry from an RACF-protected catalog requires alter-level authorization for the catalog or the entry being deleted. Altering the passwords in an RACF-protected catalog entry requires RACF alter authority for the entry being altered or the operations attribute. Alter authority for the catalog itself is not sufficient for this operation.

Note: VSAM OPEN routines bypass RACF security checking if the program issuing OPEN is in supervisor state or protection key 0.

Generic and Discrete Profiles for VSAM Data Sets

For cataloged clusters, a generic profile is used to verify access to the entire cluster, or any of its components. Discrete profiles for the individual components might exist, but only the cluster's profile (generic or discrete) is used to protect the components in the cluster.

Profiles that automatic data set protection (ADSP) processing defines during a data set define operation are cluster profiles only.

If a data set protected by a discrete profile is moved to a system where RACF is not installed, no user is given authority to access the data set. However, if the data set is protected with a generic profile, access authority is determined by normal VSAM password protection.

RACF Protection for Non-VSAM Data Sets

You can define a data set to RACF automatically or explicitly. The automatic definition occurs when space is allocated for the DASD data set, if you have the automatic data set protection attribute, or if you code PROTECT=YES or SECMODEL=(,) in the DD statement. SECMODEL=(,) lets you specify the name of the model profile RACF should use in creating a discrete profile for your data set. The explicit definition of a data set to RACF is by use of the RACF command language.

Multivolume data sets. To protect multivolume non-VSAM DASD and tape data sets, you must define each volume of the data set to RACF as part of the same volume set.

- When an RACF-protected data set is opened for output and extended to a new volume, the new volume is automatically defined to RACF as part of the same volume set.
- When a multivolume physical-sequential data set is opened for output, and any of the data set's volumes are defined to RACF, either each subsequent volume must be RACF-protected as part of the same volume set, or the data set must not yet exist on the volume.
- The system automatically defines all volumes of an extended sequential data set to RACF when the space is allocated.
- When an RACF-protected multivolume tape data set is opened for output, either each subsequent volume must be RACF-protected as part of the same volume set, or the tape volume must not yet be defined to RACF.
- If the first volume opened is not RACF protected, no subsequent volume can be RACF protected. If a multivolume data set is opened for input (or a nonphysical-sequential data set is opened for output), no such consistency check is performed when subsequent volumes are accessed.

Tape data sets. You can use RACF to provide access control to tape volumes that have no labels (NL), IBM standard labels (SL), ISO/ANSI standard labels (AL), or tape volumes referred to with bypass label processing (BLP).

RACF protection of tape data sets is provided on a volume basis or on a data set basis. A tape volume is defined to RACF explicitly by use of the RACF command language, or automatically. A tape data set is defined to RACF whenever a data set is opened for OUTPUT, OUTIN, or OUTINX and RACF tape data set protection is active, or when the data set is the first file in a sequence. All data sets on a tape volume are RACF protected if the volume is RACF protected.

If a data set is defined to RACF and is password protected, access to the data set is authorized only through RACF. If a tape volume is defined to RACF and the data sets on the tape volume are password protected, access to any of the data sets is authorized only through RACF. Tape volume protection is activated by issuing the RACF command SETROPTS CLASSACT(TAPEVOL). Tape data set name protection is activated by issuing the RACF command SETROPTS CLASSACT(TAPEDSN). Data set password protection is bypassed. The system ignores data set password protection for system-managed DASD data sets.

ISO/ANSI Version 3 and Version 4 installation exits that execute under RACF will receive control during ISO/ANSI volume label processing. Control goes to the RACHECK preprocessing and postprocessing installation exits. The same IECIEPRM exit parameter list passed to ISO/ANSI installation exits is passed to the RACF installation exits if the accessibility code is any alphabetic character from A through Z. For more information about these exits, see *z/OS DFSMS Installation Exits*.

Note: ISO/ANSI Version 4 tapes also permits special characters !*″%&’()+,-./:;<=>?_ and numeric 0-9.

Data Set Password Protection

Passwords are ignored for all system-managed data sets, new and existing. However, passwords can still be defined for system-managed data sets, and used to protect those data sets when you are sharing them with non-SMS systems that do not have RACF or an equivalent product. See “z/OS Security Server (RACF)” on page 55 for information about protecting system-managed data sets.

Passwords for VSAM Data Sets

To use password protection effectively, you need to understand the difference between operations on a catalog and operations on a data set represented by a catalog entry:

- Referring to a catalog entry when new entries are defined (ALLOCATE or DEFINE), or existing entries are altered (ALTER), deleted (DELETE), or listed (LISTCAT).
- Using the data set represented by a catalog entry when it is connected to a user’s program (OPEN), or disconnected (CLOSE).

Different passwords might be needed for each type of operation. Operations on a catalog can be authorized by the catalog’s password or, sometimes, by the password of the data set defined in the catalog. See *z/OS DFSMS Access Method Services* for information about password levels required for each operation.

The following are examples of passwords required for defining, listing, and deleting non-system-managed catalog entries:

- Defining a non-system-managed data set in a password-protected catalog requires the catalog's update (or higher) password.
- Listing, altering, or deleting a data set's catalog entry requires the appropriate password of either the catalog or the data set. However, if the catalog, but not the data set, is protected, no password is needed to list, alter, or delete the data set's catalog entry.

OPEN and CLOSE operations on a data set can be authorized by the password pointed to by the PASSWD parameter of the ACB macro. See *z/OS DFSMS Macro Instructions for Data Sets* for information about the password level required for each type of operation.

Passwords to Authorize Access

You can define passwords for access to clusters, cluster components (data and index), page spaces, alternate indexes, alternate index components (data and index), paths, master and user catalogs. Different passwords have various degrees of security, with higher levels providing greater protection than lower levels:

- **Full access.** The full access password is the master password, which lets you perform all operations (retrieving, updating, inserting, and deleting) on an entire VSAM data set and any index and catalog record associated with it. The master password permits all operations and bypasses any additional verification checking by the user-security-verification routine.
- **Control access.** The control-access password authorizes you to use control interval access. For more information see "Chapter 12. Processing Control Intervals" on page 183.
- **Update access.** The update-access password authorizes you to retrieve, update, insert, or delete records in a data set. The update password does not let you alter passwords or other security information.
- **Read access.** The read-only password allows you to examine data records and catalog records, but not add, alter, or delete them, nor see password information in a catalog record.

Each higher-level password allows all operations permitted by lower levels. Any level can be null (not specified), but if a low-level password is specified, the DEFINE and ALTER commands give the higher passwords the value of the highest password specified. For example, if only a read-level password is specified, the read-level becomes the update-, control-, and master-level password as well. If you specify a read password and a control password, the control password value becomes the master-level password as well. However, in this case, the update-level password is null because the value of the read-level password is not given to higher passwords.

Catalogs are themselves VSAM data sets, and can have passwords. For some operations (for example, listing all the catalog's entries with their passwords or deleting catalog entries), the catalog's passwords can be used instead of the entry's passwords. If the master catalog is protected, the update- or higher-level password is required when defining a user catalog, because all user catalogs have an entry in the master catalog. When deleting a protected user catalog, the user catalog's master password must be specified.

Some access method services operations might involve more than one password authorization. For example, importing a data set involves defining the data set and loading records into it. If the catalog into which the data set is being imported is

password protected, its update-level (or higher-level) password is required for the definition; if the data set is password protected, its update-level (or higher-level) password is required for the load. The IMPORT command lets you specify the password of the catalog; the password, if any, of the data set being imported is obtained by the commands from the exported data.

Every VSAM data set is represented in a catalog by two or more components: a cluster component and a data component, or, if the data set is a key-sequenced data set, a cluster component, a data component, and an index component. Of the two or three components, the cluster component is the controlling component. Each of the two or three components can have its own set of four passwords; the passwords you assign have no relationship to each other. For example, password protecting a cluster but not the cluster's data component, lets you issue LISTCAT to determine the name of your cluster's data component, open the data component, and access records in it, even though the cluster itself is password protected.

One reason for password-protecting the components of a cluster is to prevent access to the index of a key-sequenced data set. (One way to gain access to an index is to open it independently of the cluster.)

Password-Protection Precautions

When you use protection commands for a non-system-managed catalog or for a data set, you need to observe certain password-protection precautions, which the following lists describe.

For a Catalog. Observe the following precautions when you use protection commands for a non-system-managed catalog:

- ***To create a non-system-managed catalog entry using the DEFINE command,*** the update-level or higher-level password of the catalog is required.
- ***To modify a catalog entry using the ALTER command,*** the master password of the entry, or the master password of the catalog that contains the entry, is required. However, if the entry to be modified is a non-VSAM or generation data group entry, the update-level password of the catalog is sufficient.
- ***To gain access to passwords in a catalog (for example, to list or change passwords),*** specify the master-level password of either the entry or the catalog. A master-level password must be specified with the DEFINE command to model an entry's passwords.
- ***To delete a protected data set entry from a catalog,*** requires the master-level password of the entry or the master-level password of the catalog containing the entry. However, if the entry in a catalog describes a VSAM data space, the update-level password of the catalog is sufficient.
- ***To delete a non-VSAM, generation data group, or alias entry,*** the update-level password of the catalog is sufficient.
- ***To list catalog entries with the read-level passwords,*** specify the read password of the entry or the catalog's read-level password. However, entries without passwords can be listed without specifying the catalog's read-level password.
- ***To list the passwords associated with a catalog entry,*** specify the master password of the entry or the catalog's master password.

  To avoid unnecessary prompts, specify the catalog's password, which permits access to all entries the operation affects. A catalog's master-level password lets you refer to all catalog entries. However, a protected cluster cannot be processed with the catalog's master password.

  Specification of a password where none is required is always ignored.

For a Data Set. Observe the following precautions when you use protection commands for a data set:

- ***To access a VSAM data set using its cluster name instead of data or index names,*** specify the proper level password for the cluster even if the data or index passwords are null.
- ***To access a VSAM data set using its data or index name instead of its cluster name,*** specify the proper data or index password. However, if cluster passwords are defined, the master password of the cluster can be specified instead of the data or index password.
- ***Null means no password was specified,*** if a cluster has only null passwords, access the data set using the cluster name without specifying passwords, even if the data and index entries of the cluster have defined passwords. Using null passwords permits unrestricted access to the VSAM cluster but protects against unauthorized modification of the data or index as separate components.

Data Set and Catalog Protection

If you define passwords for any data sets in a catalog, you must also protect the catalog by defining passwords for the catalog or by defining the catalog to RACF. If you do not protect the catalog, no password checking takes place during operations on the data set's catalog entries or during open processing of data sets cataloged in that catalog.

Password Prompting

Computer operators and TSO/E terminal users should supply a correct password if a processing program does not give the correct one when it tries to open a password-protected data set. When the data set is defined, use the CODE parameter to specify a code instead of the data set name to prompt the operator or terminal user for a password. The prompting code keeps your data secure by not permitting the operator or terminal user to know both the name of the data set and its password.

A data set's code is used for prompting for any operation against a password-protected data set. The catalog code is used for prompting when the catalog is opened as a data set, when an attempt is made to locate catalog entries that describe the catalog, and when an entry is to be defined in the catalog.

If you do not specify a prompting code, VSAM identifies the job for which a password is needed with the JOBNAME and DSNAME for background jobs or with the DSNAME alone for foreground (TSO/E) jobs.

When you define a data set, use the ATTEMPTS parameter to specify the number of times the computer operator or terminal user is permitted to give the password when a processing program is trying to open a data set.

If the ATTEMPTS parameter is coded with 0, no password prompting is done. If you exceed the allowed number of attempts when using System Management Facilities, a record is written to the SMF data set to indicate a security violation.

If you are logged in to TSO/E, VSAM tries the login password before prompting at your terminal. Using the TSO/E login password counts as one attempt.

Passwords for Non-VSAM Data Sets

IBM recommends not using passwords for data sets. The security provided by data set passwords is not as good as security provided by RACF. See *z/OS DFSMSdfp Advanced Services*.

The system ignores data set password protection for system-managed data sets.

Assigning a Password

Use the PROTECT macro or the IEHPROGM PROTECT command to assign a password to the non-VSAM data set. See *z/OS DFSMSdfp Advanced Services* and *z/OS DFSMSdfp Utilities*.

Protecting a Data Set When You Define It

When you define a non-VSAM data set in a catalog, the data set is not protected with passwords in its catalog entry. However, you can password-protect the catalog.

Two levels of protection options for your data set are available. Specify these options in the LABEL field of a DD statement with the parameter PASSWORD or NOPWREAD. See *z/OS MVS JCL Reference*.

- Password protection (specified by the PASSWORD parameter) makes a data set unavailable for all types of processing until a correct password is entered by the system operator, or for a TSO/E job by the TSO/E user.
- No-password-read protection (specified by the NOPWREAD parameter) makes a data set available for input without a password, but requires that the password be entered for output or delete operations.

The system sets the data set security indicator either in the standard header label 1, as shown in *z/OS DFSMS: Using Magnetic Tapes*, or in the data set control block (DSCB). After you have requested security protection for magnetic tapes, you cannot remove it with JCL unless you overwrite the protected data set.

Supplying a Password for a Catalog

If the catalog is update protected, you must supply the catalog's update- or higher-level password to define, delete, or alter a non-VSAM data set. The password can be supplied as a subparameter of the command's CATALOG parameter, or as a response to the password-prompting message.

Handling Incorrect Passwords

If an incorrect password is entered twice when a password is being requested by the open or EOV routine, the system issues an ABEND 913. For a SCRATCH or RENAME request, a return code is given.

Entering a Record in the PASSWORD Data Set

In addition to requesting password protection in your JCL, you must enter at least one record for each protected data set in a data set named PASSWORD. The PASSWORD data set must be created on the system-residence volume. The system-residence volume contains the nucleus of the operating system. The system programmer should also request password protection for the PASSWORD data set itself to prevent both reading and writing without knowledge of the password.

Protecting Data Sets

For a data set on direct access storage devices, place the data set under protection when you enter its password in the PASSWORD data set. Use the PROTECT macro or the IEHPROGM utility program to add, change, or delete an entry in the PASSWORD data set. Using either of these methods, the system updates the DSCB of the data set to reflect its protected status. Therefore, you do not need to use JCL whenever you add, change, or remove security protection for a data set on direct access storage devices. See *z/OS DFSMSdfp Advanced Services* for information about maintaining the PASSWORD data set, including the PROTECT macro. Also see *z/OS DFSMSdfp Utilities* for information about the IEHPROGM utility program.

User-Security-Verification Routine

Besides password protection, VSAM lets you protect data by specifying a program that verifies a user's authorization. "User-Security-Verification Routine" on page 260 describes specific requirements. To use this additional protection, specify the name of your authorization routine in the AUTHORIZATION parameter of the DEFINE or ALTER command.

If a password exists for the type of operation you are performing, you must specify the password, either in the command or in response to prompting. VSAM calls the user-security-verification routine only after it verifies the password. VSAM bypasses this routine whenever you specify a correct master password, whether or not the operation requires the master password.

You can use the USVR to detect VSAM password usage.

Erasure of Residual Data

When you release media space, you can erase your data.

Erasing DASD Data

When you delete any DASD data set or release part of the space, the system makes the space available for allocation for new data sets. There are ways that the creator of the new data set can read residual data that was in the previous data set. To prevent others from reading your deleted data, run a program that overwrites the data before you delete it. Alternatively, you can have the system erase (overwrite) the data during data set deletion or space release, with its erase-on-scratch function. If the system erasure fails, then the deletion or space release fails.

To have the system erase sensitive data with RACF, the system programmer can start the erase feature with the RACF SETROPTS command. This feature controls the erasure of DASD space when it is releases. Space release occurs when you delete a data set or release part of a data set. SETROPTS selects one of the following methods for erasing the space:

- The system erases all released space.
- The system erases space only in data sets that have a security level greater than or equal to a certain level.
- The system erases space in a data set only if its RACF data set profile specifies the ERASE option.
- The system never erases space.

If the ERASE option is set in the RACF profile, you cannot override the option by specifying NOERASE in access methods services commands.

System Erasure of Data

If DASD data erasure is in effect and you use any of the following items, the system overwrites the entire data set area:

- The DELETE subparameter in the JCL DISP parameter of a data definition (DD) statement
- The TSO DELETE command (for non-VSAM objects)
- The SCRATCH macro
- The SCRATCH control statement for the IEHPROGM utility program
- The access method services DELETE command

For a sequential, partitioned, PDSE, or VSAM extended-format data set, if DASD data erasure is in effect, the system also overwrites the released area when you use any of the following:

- RLSE subparameter in the JCL SPACE parameter in a DD statement to which a program writes
- Partial release option in the management class
- PARTREL macro

RAMAC® Virtual Array

With RAMAC Virtual Array, the DDSR option of IXFP does almost the same thing as the erase-on-scratch function. The storage administrator uses DDSR to manage disk space more efficiently. DDSR has the side effect of **usually** erasing released tracks, even if you do not request the ERASE option. On other types of disks, DDSR is faster than the erase-on-scratch function. Without erase-on-scratch, however, DDSR is less secure. The erasure might not complete before data set deletion or space release. After a successful erasure, your data remains physically on disk, in a compressed form, but is not accessible by any software.

If you request erase-on-scratch on a RAMAC Virtual Array for which DDSR is active, the system optimizes the erasure so that it happens much faster than on other kinds of disks. The erasure is guaranteed to complete before data set deletion or space release. After a successful erasure, your data is not accessible by any software.

Erasing Tape Data

If you want to prevent the reading of residual data on tape volumes, you can implement some method of overwriting the volume. A DFSMSrmm user can define security classes in EDGR*xx*, a DFSMSrmm parmlib member, by using name masks to identify data sets that must be erased before the volume can be released for use as scratch. If DFSMSrmm determines that the security class of a data set requires erasure, DFSMSrmm sets release actions of ERASE and INIT for any volume that contains the data set. When all data sets on the volume have expired, DFSMSrmm holds the volume until these actions have been confirmed.

To automate the overwriting of residual data, schedule a regular EDGINERS job to process volumes that have the erase action pending. DFSMSrmm selects the volumes to process and prompts the operator to mount each one. After verifying that the correct volume is mounted, DFSMSrmm erases the volume, using the hardware-security erase feature, where supported, to free the channel for other activity during the erasure.

If the hardware-security erase feature is not available, DFSMSrmm overwrites volumes with a bit pattern of X'FF'. When erasing volumes, DFSMSrmm also reinitializes them so that the correct volume labels are written, and the volumes are ready for reuse in a single operation.

Authorized Program Facility and Access Method Services

The authorized program facility (APF) limits the use of sensitive system services and resources to authorized system and user programs. For information about program authorization see *z/OS MVS Programming: Authorized Assembler Services Guide*.

All access method services load modules are contained in SYS1.LINKLIB, and the root segment load module (IDCAMS) is link edited with the SETCODE AC(1) attribute.

APF authorization is established at the job step level. If, during the execution of an APF-authorized job step, a load request is satisfied from an unauthorized library, the task is abnormally terminated. It is the installation's responsibility to ensure that a load request cannot be satisfied from an unauthorized library during access method services processing.

The following situations could cause the invalidation of APF authorization for access method services:

- An access method services module is loaded from an unauthorized library.
- A user-security-verification routine (USVR) is loaded from an unauthorized library during access method services processing.
- An exception exit routine is loaded from an unauthorized library during access method services processing.
- A user-supplied special graphics table is loaded from an unauthorized library during access method services processing.

Because APF authorization is established at the job-step task level, access method services is not authorized if invoked by an unauthorized application program or unauthorized terminal monitor program (TMP).

The system programmer must enter the names of those access method services commands that require APF authorization to execute under TSO/E in the authorized command list.

Programs that are designed to be called from an APF-authorized program should never be linked or bound with APF authorization. Someone could invoke the routine directly through JCL, and it would be operating with APF authorization in an environment for which it was not designed. Programs that you intend to be called by an APF-authorized program should be in APF-authorized libraries.

The following restricted access method services functions cannot be requested in an unauthorized state:

CNVTCAT—When converting to a catalog

DEFINE—When the RECATALOG parameter is specified

DELETE—When the RECOVERY parameter is specified

EXPORT—When the object to be exported is a catalog

IMPORT—When the object to be imported is a catalog

PRINT—When the object to be printed is a catalog

REPRO—When copying a catalog or when the catalog unload/reload is to be used

VERIFY—When a catalog is to be verified

If the above functions are required and access method services is invoked from an application program or TSO/E terminal monitor program, the invoking program must be authorized.

See *z/OS DFSMSdfp Storage Administration Reference* for information about authorizing for TSO/E and ISPF.

Access Method Services Cryptographic Option

Although security can be provided for online data by using such facilities as VSAM password protection and Resource Access Control Facility (RACF), these facilities do not protect data when it is stored offline. Sensitive data stored offline is susceptible to misuse.

It is generally recognized that data cryptography is an effective means of protecting offline data, if the enciphering techniques are adequate. The enciphering function is available by using the ENCIPHER option of the access method services REPRO command. The REPRO command uses the services of the IBM Programmed Cryptographic Facility (5740-XY5) or the Cryptographic Unit Support (5740-XY6). The Programmed Cryptographic Facility conforms to the Data Encryption Standard (DES) of the United States National Institute of Science and Technology, previously called National Bureau of Standards for enciphering data. The data remains protected until the REPRO DECIPHER option is used to decipher it with the correct key.

Related reading: See *z/OS DFSMS Access Method Services* for the syntax and examples of the REPRO command.

Data Enciphering and Deciphering

In the following three types of offline environments, the enciphering of sensitive data adds to data security:

- Data sets are transported to another installation, where data security is required during transportation and while the data is stored at the other location
- Data sets are stored for long periods of time at a permanent storage location
- Data sets are stored offline at the site at which they are normally used.

You can use REPRO to copy a plaintext (not enciphered) data set to another data set in enciphered form. Enciphering converts data to an unintelligible form called a ciphertext. The enciphered data set can then be stored offline or sent to a remote location. When desired, the enciphered data set can be brought back online and REPRO can be used to recover the plaintext from the ciphertext by copying the enciphered data set to another data set in plaintext (deciphered) form.

Enciphering and deciphering are based on an 8-byte binary value called the key. Using the REPRO DECIPHER option, the data can be either deciphered on the system it was enciphered on, or deciphered on another system that has this functional capability and the required key to decipher the data. Given the same key, encipher and decipher are inverse operations. The REPRO DECIPHER option uses the services of the Programmed Cryptographic Facility or the Cryptographic Unit Support to encipher/decipher the data, and uses block chaining with ciphertext feedback during the encipher/decipher operation.

Except for catalogs, all data sets supported for copying by REPRO are supported as input (SAM, ISAM, VSAM) for enciphering and as output (SAM, VSAM) for deciphering. The input data set for the decipher operation must be an enciphered copy of a data set produced by REPRO. The output data set for the encipher operation can only be a VSAM entry-sequenced, linear, or sequential (SAM) data set. The target (output) data set of both an encipher and a decipher operation must be empty. If the target data set is a VSAM data set that has been defined with the reusable attribute, use the REUSE parameter of REPRO to reset it to an empty status.

For both REPRO ENCIPHER and REPRO DECIPHER, if the input data set (INDATASET) is system managed, the output data set (OUTDATASET) can be either system managed or not system managed and must be cataloged in the catalog determined by the catalog search order. For information about the order of catalog search, see *z/OS DFSMS Access Method Services*.

The REPRO ENCIPHER parameter indicates that REPRO is to produce an enciphered copy of the data set, and to supply information needed for the encipherment. The INFILE or INDATASET parameter identifies and allocates the plaintext (not enciphered) source data set. The OUTFILE or OUTDATASET parameter identifies and allocates a target data set to contain the enciphered data.

The REPRO DECIPHER parameter indicates that REPRO is to produce a deciphered copy of the data set, and to supply information needed for deciphering. The INFILE or INDATASET parameter identifies and allocates the enciphered source data set. The OUTFILE or OUTDATASET parameter identifies and allocates a target data set to contain the plaintext data.

Figure 3 is a graphic representation of the input and output data sets involved in REPRO ENCIPHER/DECIPHER operations.

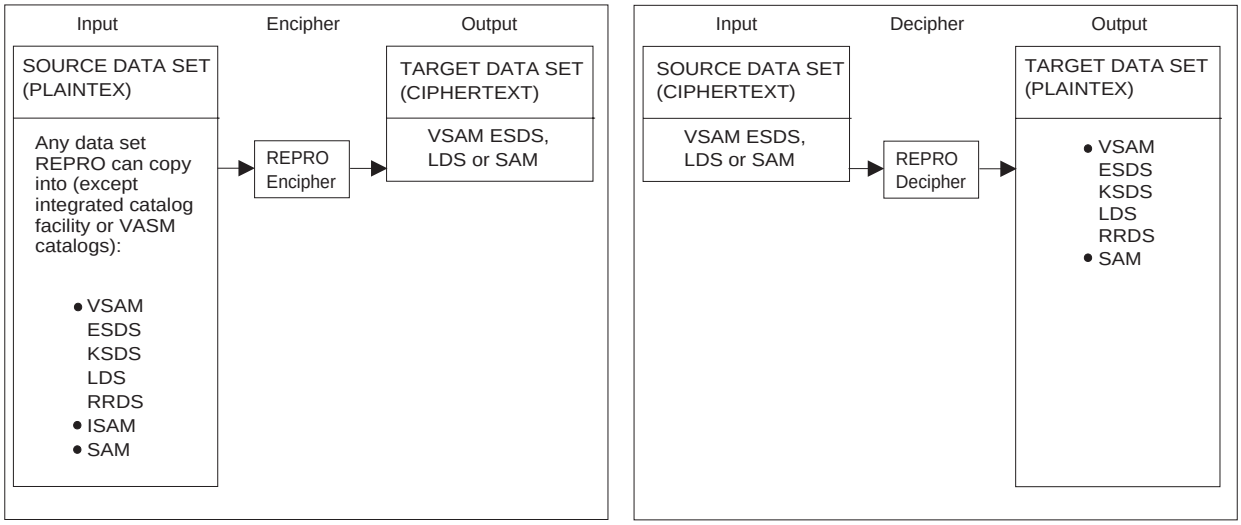


Figure 3. REPRO Encipher/Decipher Operations

You should not build an alternate index over a VSAM entry-sequenced data set that is the output of a REPRO ENCIPHER operation.

When a VSAM relative record data set is enciphered, the record size of the output data set must be at least four bytes greater than the record size of the relative record data set. (The extra four bytes are needed to prefix a relative record number to the output record.) Specify the record size of an output VSAM entry-sequenced data set through the RECORDSIZE parameter of the DEFINE CLUSTER command. Specify the record size of an output sequential (SAM) data set through the DCB LRECL parameter in the output data set's DD statement. When an enciphered VSAM relative record data set is subsequently deciphered with a relative record data set as the target, any empty slots in the original data set are reestablished. When a linear data set is enciphered, both the source (input) and target (output) data sets must be linear data sets.

When you encipher a data set, specify any of the delimiter parameters available with the REPRO command (SKIP, COUNT, FROMADDRESS, FROMKEY, FROMNUMBER, TOADDRESS, TOKEY, TONUMBER) that are appropriate to the data set being enciphered. However, no delimiter parameter can be specified when a data set is deciphered. If DECIPHER is specified together with any REPRO delimiter parameter, your REPRO command terminates with a message.

If you specify the REPRO command's REPLACE parameter and either the ENCIPHER or the DECIPHER parameter, access method services ignores REPLACE.

When the REPRO command copies and enciphers a data set, it precedes the enciphered data records with one or more records of clear header data. The header data preceding the enciphered data contains information necessary for the deciphering of the enciphered data, such as:

- Number of header records
- Number of records to be ciphered as a unit
- Key verification data
- Enciphered data encrypting keys.

If the output data set for the encipher operation is a compressed format data set, little or no space is saved. This is because enciphered data is pseudorandom. Save space for the output if the input data set is in compressed format and is compressed.

Key Management

This section provides an overview of key management, for enciphering or deciphering data through the REPRO command.

Data Encryption Keys

A key is defined as an 8-byte value. When you use the encipher/decipher function of the REPRO command, specify keys the Programmed Cryptographic Facility or the Cryptographic Unit Support generates and manages for you (system-key management), or keys you manually generate and privately manage (private-key management). In either case, REPRO invokes the appropriate Programmed Cryptographic Facility service.

For both private-key and system-key management, REPRO lets you supply an 8-byte value to use as the plaintext data encrypting key. If you do not supply the data encrypting key, REPRO provides an 8-byte value to be used as the plaintext data encrypting key. The plaintext data encrypting key is used to encipher/decipher the data using the Data Encryption Standard.

If you supply your own plaintext data encrypting key on ENCIPHER or DECIPHER through the REPRO command, you risk exposing that key when the command is listed on SYSPRINT. To avoid this exposure, direct REPRO to a data encrypting key data set to obtain the plaintext data encrypting key. Identify the DD statement for this data set through the DATAKEYFILE parameter of the REPRO command. REPRO obtains the data encrypting key from this data set by locating the first nonblank column in the first record of the data set and processing 16 consecutive bytes of data. These 16 consecutive bytes contain the hexadecimal character representation for the 8-byte key. Each byte of data must contain the EBCDIC character representation of a hexadecimal digit (that is, 0 through F). If a blank byte is found before 16 nonblank bytes have been processed, the data key is padded to the right with EBCDIC blanks. If the end of the first record is encountered before 16 bytes have been processed, the data key is considered not valid. If a valid data encrypting key cannot be obtained from the first record of the data encrypting key data set, then REPRO either generates the data encrypting key (ENCIPHER) or terminates with a message (DECIPHER).

If you allow (or force, by a not valid data encrypting key data set record) REPRO to provide the data encrypting key on ENCIPHER, you risk exposing the data encrypting key only if you manage your keys privately. REPRO lists the generated data encrypting key only for privately managed keys.

Secondary File Keys

When you want to decipher the data, you must supply the data encrypting key that was used to encipher the data. However, as a security precaution, you might want to supply the data encrypting key in a disguised form. When enciphering the data set, supply the name of a system-managed key, called a secondary file key. REPRO uses the secondary file key indicated by the supplied name to disguise (encipher) the data encrypting key. When deciphering the data set, the name of the file key and the disguised data encrypting key can be supplied rather than the plaintext data encrypting key. In this way, the actual plaintext data encrypting key is not revealed. (Note that the secondary file key is used only in key communication, not when enciphering data.)

The Programmed Cryptographic Facility or the Cryptographic Unit Support offers the installation of this secondary file-key-management facility. To use this facility, your installation must first execute the Programmed Cryptographic Facility key generator utility. The Programmed Cryptographic Facility key generator utility generates the secondary-file keys to be used by the installation.

The Programmed Cryptographic Facility key generator utility generates the secondary-file keys you request and stores the keys, in enciphered form, in the cryptographic key data set (CKDS). It lists the external name (key name) of each secondary key and the plaintext form of the secondary key. To access a particular secondary file key, you supply the key name of the secondary file key. If the secondary key is to be used on a system other than the system on which the keys were generated, the utility must also be executed at the other system to define the same plaintext secondary file keys. The plaintext secondary file keys can be defined in the CKDS of the other system with different key names.

If you want to manage your own private keys, no secondary file keys are used to encipher the data encrypting key; it is your responsibility to ensure the secure nature of your private data encrypting key.

If you select the Programmed Cryptographic Facility or Cryptographic Unit Support to manage your keys, REPRO uses secondary file keys to encipher the data encrypting key. You specify a secondary file key to REPRO through the key name by which it is known in the CKDS. Two types of secondary file keys are recognized by REPRO:

1. An internal file key, defined in the CKDS by using a remote or cross key 2 statement as input to the Programmed Cryptographic Facility key generator utility. Use of a cross key 2 statement for DECIPHER requires APF (Authorized Program Facility) authorization and can provide an additional level of security.
2. For ENCIPHER, an external file key, defined in the CKDS by using a local or cross key 1 statement as input to the Programmed Cryptographic Facility key generator utility. An internal file key can be used for ENCIPHER but not for DECIPHER operations.

Although both internal and external secondary file keys can be used when doing a REPRO ENCIPHER, only internal secondary file keys can be used when doing a REPRO DECIPHER. Thus, external secondary file keys cannot be used to do a REPRO ENCIPHER and REPRO DECIPHER on the same system. External secondary file keys are used when the enciphered data is to be deciphered on another system. The REPRO DECIPHER is done on the other system using the same secondary file key. However, in this other system's CKDS, the secondary file key must be defined as an internal secondary file key.

An internal secondary file key can be used to do a REPRO ENCIPHER and REPRO DECIPHER on the same system, and, if the same secondary file key is also defined as an internal secondary file key in another system's CKDS, then REPRO ENCIPHER and REPRO DECIPHER can also be done on that system.

Requirements for REPRO ENCIPHER and DECIPHER

In planning to use the ENCIPHER and DECIPHER functions of the REPRO command, you should be aware of the following requirements:

- The data set for the integrated cryptographic feature (ICSF) options must have COMPAT(YES) coded before AMS REPRO can invoke the Programmed Cryptographic Facility and Cryptographic Unit Support macros to work on ICSF.
- The Programmed Cryptographic Facility/Cryptographic Unit Support CKDS must be converted to ICSF format.
- Before you issue the REPRO command (through a batch job or from a TSO/E terminal), the Programmed Cryptographic Facility must have already been started through a START command at the operator's console.

In planning to use REPRO DECIPHER with the SYSTEMKEYS parameter specifying a cross key 2 secondary file key, you should be aware of the following requirement:

- Access method services must be authorized. See "Authorized Program Facility and Access Method Services" on page 64.

Chapter 6. Using Hierarchical File System Data Sets

This chapter covers the following topics.

Topic	Page
Creating Hierarchical File System Data Sets	72
Creating Additional Directories	72

A hierarchical file system (HFS) data set contains a POSIX-compliant HFS, which is a collection of files and directories that are organized in a hierarchical structure. You can access an HFS by using the z/OS UNIX system services (USS) in an open system environment. You can use many of the standard BSAM, QSAM, and VSAM interfaces to access files within a z/OS UNIX HFS file system. Most applications that use these access methods can access HFS files without reassembly or recompilation. See “Processing USS Files with an Access Method” on page 18, “Simulated VSAM Access to USS files” on page 86, and “Accessing USS Files Using BSAM, QSAM, or VSAM” on page 401.

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.

HFS data sets appear to the MVS system much as a PDSE does, but the internal structure is entirely different. DFSMS manages HFS data sets. It provides access to the files of data within the data sets as well as backup/recovery and migration/recall capability.

HFS data sets, not HFS files, have the following requirements and restrictions:

- They must reside on DASD volumes and be cataloged.
- They cannot be processed with standard open, end-of-volume, or access methods. You must use POSIX system calls to process them instead.
- They are supported by standard DADSM create, rename, and scratch routines.
- They are supported by DFSMShsm for dump/restore and migrate/recall if DFSMSdss is used as the data mover. DFSMShsm does not process individual files within an HFS data set.
- They are supported by the DFSMSdss DUMP, RESTORE, and COPY commands.
- They are not supported by IEBCOPY.

z/OS DFSMSdfp Advanced Services provides more information about managing HFS data sets.

© Copyright IBM Corp. 1987, 2002

Creating Hierarchical File System Data Sets

To allocate an HFS data set, specify HFS in the DSNTYPE parameter and the number of directory blocks in the SPACE parameter, in either the JCL or the data class. You must specify the size of the directory block, even if the size is not used, or the allocation fails.

You can define a data class for HFS data sets. Although you can create uncataloged HFS data sets, they must be cataloged when they are mounted. These data sets can expand to as many as 255 extents of 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.

You cannot use an HFS data set until someone with appropriate RACF authority uses the TSO MOUNT command to logically mount the file system. See *z/OS DFSMSdfp Advanced Services* and *z/OS UNIX System Services Planning*.

Creating Additional Directories

After someone allocates an HFS data set for the root file system, you can log on as a TSO/E user and define additional directories in the root file system by using the MKDIR command. For example, to create the /u/joe directory, enter:

```
MKDIR '/u'
MKDIR '/u/joe'
```

These directories can be used as mount points for additional mountable file systems. A storage administrator can use a program supplied by IBM 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 certain z/OS system security or UNIX system services authority. See *z/OS DFSMSdfp Advanced Services* and *z/OS UNIX System Services Planning*.

A user with write access authority to a directory, can use the MKDIR command to create subdirectories in that directory. Within the root directory, only superusers can create subdirectories. The MOUNT command is used by access authorized users (as an example, superusers) to mount file systems in a directory.

Part 2. VSAM Data Sets

© Copyright IBM Corp. 1987, 2002

Chapter 7. Organizing VSAM Data Sets

This chapter covers the following topics.

Topic	Page
VSAM Data Formats	75
VSAM Data Striping	80
Selection of VSAM Data Set Types	84
Extended-Format Data Sets	95
Access to Records in a VSAM Data Set	97
Access to Records through Alternate Indexes	100
Data Compression	103

VSAM Data Formats

The organization of data in all VSAM data sets, except linear data sets, is arranged in records, also called logical records. A logical record is the user record requested from, or given to, the VSAM record management function.

Logical records of VSAM data sets are stored differently from logical records in non-VSAM data sets. VSAM stores records in control intervals. A control interval is a continuous area of direct access storage that VSAM uses to store data records and control information that describes the records. Whenever a record is retrieved from direct access storage, the entire control interval containing the record is read into a VSAM I/O buffer in virtual storage. The desired record is transferred from the VSAM buffer to a user-defined buffer or work area. Figure 4 shows how a logical record is retrieved from direct access storage.

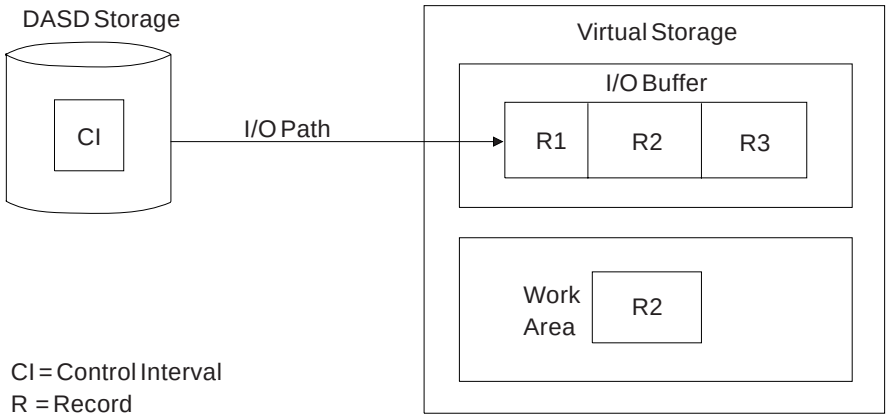


Figure 4. VSAM Logical Record Retrieval

Data Set Size

The maximum size of a VSAM data set is 4 GB (1 073 741 824 bytes) unless it is defined with a data class that specifies a DSNTYPE of EXT (extended format) with the extended addressability (also in the data class) set to Y (yes). A VSAM data set can be expanded to 123 extents per volume. The last four extents are reserved for extending a data set when the last extent cannot be allocated in one piece. (VSAM

© Copyright IBM Corp. 1987, 2002

attempts to extend a data set when the total number of extents is less than 250.) See “Allocating Space for VSAM Data Sets” on page 111 for information about space allocation for VSAM data sets.

Control Intervals

The size of control intervals can vary from one VSAM data set to another, but all the control intervals within the data portion of a particular data set must be the same length. Use the access method services DEFINE command and let VSAM select the size of a control interval for a data set, or request a particular control interval size. See “Optimizing Control Interval Size” on page 159 for information about selecting the best control interval size.

A control interval consists of:
- Logical records
- Free space
- Control information fields

In a linear data set all of the control interval bytes are data bytes. There is no imbedded control information.

Control Information Fields

Figure 5 contains control information consisting of two types of fields: one control interval definition field (CIDF), and one or more record definition fields (RDFs).

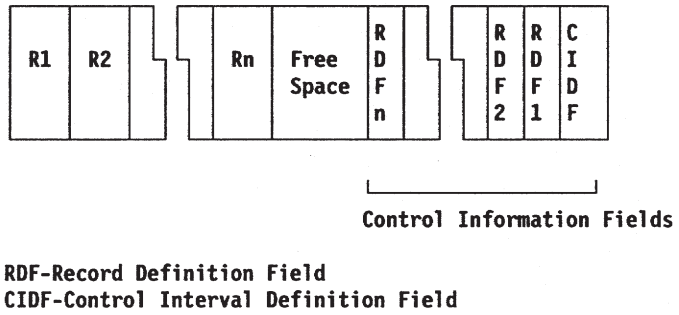


Figure 5. Control Interval Format

CIDFs are 4 bytes long, and contain the amount and location of free space. RDFs are 3 bytes long, and describe the length of records and how many adjacent records are of the same length.

If two or more adjacent records have the same length, only two RDFs are used for this group. One RDF gives the length of each record, and the other gives the number of consecutive records of the same length. Figure 6 shows RDFs for records of the same and different lengths:

Control Interval 1

Control interval size = 512 bytes
Record length = 160-byte records
Record definition fields-Only 2 RDFs are needed because all records are the same length.

	R1	R2	R3	FS	RDF 2	RDF 1	CIDF
Record length	160	160	160	22	3	3	4

Control Interval 2

Control interval size = 512 bytes
Record length-All records have different lengths
Record definition fields-One RDF is required for each logical record (RDF 1 for record 1, RDF 2 for record 2, and so forth).

R1	R2	R3	R4	FS	RDF 4	RDF 3	RDF 2	RDF 1	CIDF
130	70	110	140	46	3	3	3	3	4

Control Interval 3

Control interval size = 512 bytes
Record length-Records 1 through 3 are 80-byte records.
Records 4 and 5 have different lengths.
Record definition fields-Two RDFs are used for records 1 through 3.
Record 4 and 5 each have their own RDF.

R1	R2	R3	R4	R5	FS	RDF	RDF	RDF	RDF	CIDF
80	80	80	100	93	63	3	3	3	3	4

FS = Free Space

Figure 6. Record Definition Fields of Control Intervals

If a record exceeds the maximum record length, an error message is generated. If a record in an entry-sequenced or key-sequenced data set, or variable-length RRDS is smaller than the maximum record length, VSAM saves disk space by storing the actual record length in the RDF. However, in a fixed-length RRDS, records do not vary in length. The RDF reflects the length of the record slot, and cannot be adjusted.

Compressed Control Information Field

Compressed data records in an extended-format key-sequenced data set have a different format than noncompressed data records. This format includes a record prefix that contains internal compression information. When the record is a spanned record, each segment of the record contains a segment prefix with information similar to the record prefix for describing the segment. The length of the record prefix for nonspanned records is 3 bytes, and the length for spanned records is 5 bytes.

The stored record format has no affect on the data seen by the user as a result of a VSAM GET request. In addition, no special processing is required to place the record in the data set in a compressed format.

The presence of the record prefix does result in several incompatibilities that can affect the definition of the key-sequenced data set or access to the records in the key-sequenced data set. When a VSAM data set is in compressed format, VSAM must be used to extract and expand each record to obtain data that is usable. If a method other than VSAM is used to process a compressed data set and the method does not recognize the record prefix, the end result is unpredictable and could result in loss of data. See “Compressed Data” on page 96.

Control Areas

The control intervals in a VSAM data set are grouped together into fixed-length contiguous areas of direct access storage called control areas. A VSAM data set is actually composed of one or more control areas. The number of control intervals in a control area is fixed by VSAM.

The maximum size of a control area is one cylinder, and the minimum size is one track of DASD storage. When you specify the amount of space to be allocated to a data set, you implicitly define the control area size. See “Defining Alternate Indexes” on page 120 for information about defining an alternate index. See “Optimizing Control Area Size” on page 164 for information about finding the best control area size.

Spanned Records

Sometimes a record is larger than the control interval size used for a particular data set. In VSAM, you do not need to break apart or reformat such records, because you can specify spanned records when defining a data set. The SPANNED parameter permits a record to extend across or span control interval boundaries.

Spanned records might reduce the amount of DASD space required for a data set when data records vary significantly in length, or when the average record length is larger compared to the CI size. The following figures show the use of spanned records for more efficient use of space.

In Figure 7, each control interval is 10 240 bytes long.

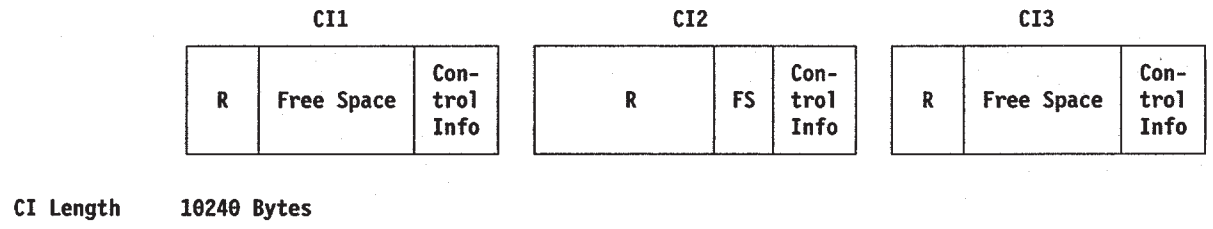


Figure 7. Data Set with Nonspanned Records

In Figure 7 control interval 1 contains a 2000-byte record. Control interval 2 contains a 10 000-byte record. Control interval 3 contains a 2000-byte record. All together, these three records use 30 720 bytes of storage.

Figure 8 contains a data set with the same space requirements as in Figure 7, but one that permits spanned records.

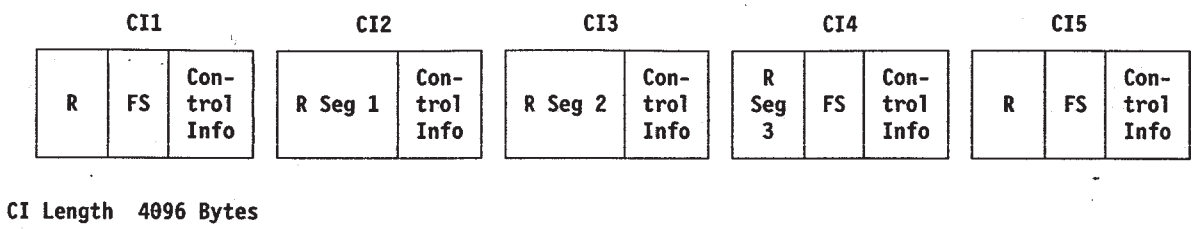


Figure 8. Data Set with Spanned Records

The control interval size is reduced to 4096 bytes. When the record to be stored is larger than the control interval size, the record is spanned between control intervals. In Figure 8, control interval 1 contains a 2000-byte record. Control intervals 2, 3, and 4 together contain one 10 000-byte record. Control interval 3 contains a 2000-byte record. By changing control interval size and permitting spanned records, you can store the three records in 20 480 bytes, reducing the amount of storage needed by 10 240 bytes.

Remember the following rules:

- A spanned record always begins on a control interval boundary and fills more than one control interval within a single control area.
- For compressed data sets with spanned records, the length of the record prefix is 5 bytes. Because of the additional 5 bytes, the key offset plus the key length (that is, relative key position) must be less than or equal to the CI size less 15.
- For key-sequenced data sets, the entire key field of a spanned record must be in the first control interval.
- The control interval containing the last segment of a spanned record might also contain unused space. Use the unused space only to extend the spanned record; it cannot contain all or part of any other record.
- Spanned records can only be used with key-sequenced data sets and entry-sequenced data sets.

- To span control intervals, you must specify the SPANNED parameter when you define your data set. VSAM decides whether a record is spanned or nonspanned, depending on the control interval length and the record length. Spanned/nonspanned can also be specified in the data class.
- Locate mode (OPTCD=LOC on the RPL) is not a valid processing mode for spanned records. A nonzero return code will be issued if locate mode is used.

VSAM Data Striping

To use striped data, a data set must be in extended format. All VSAM data set organizations are supported for striped data:

- Key-sequenced data set (KSDS)
- Entry-sequenced data set (ESDS)
- Relative-record data set (RRDS)
- Variable-length relative-record data set (VRRDS)
- Linear data set (LDS)

A striped data set has tracks that spread across multiple devices, as is the case for sequential access method or the CIs for VSAM. This format allows a single application request for records in multiple tracks or CIs to be satisfied by concurrent I/O requests to multiple volumes. The result is improved performance for sequential data access by achieving data transfer into the application at a rate greater than any single I/O path. The scheduling of I/O to multiple devices to satisfy a single application request is referred to as an I/O packet.

VSAM data striping applies only to data sets that are defined with more than one stripe. Any data set listed with one stripe is in the extended format and is not considered to be a striped data set.

Figure 9 illustrates primary and secondary space allocations on multiple volumes for a striped VSAM data set.

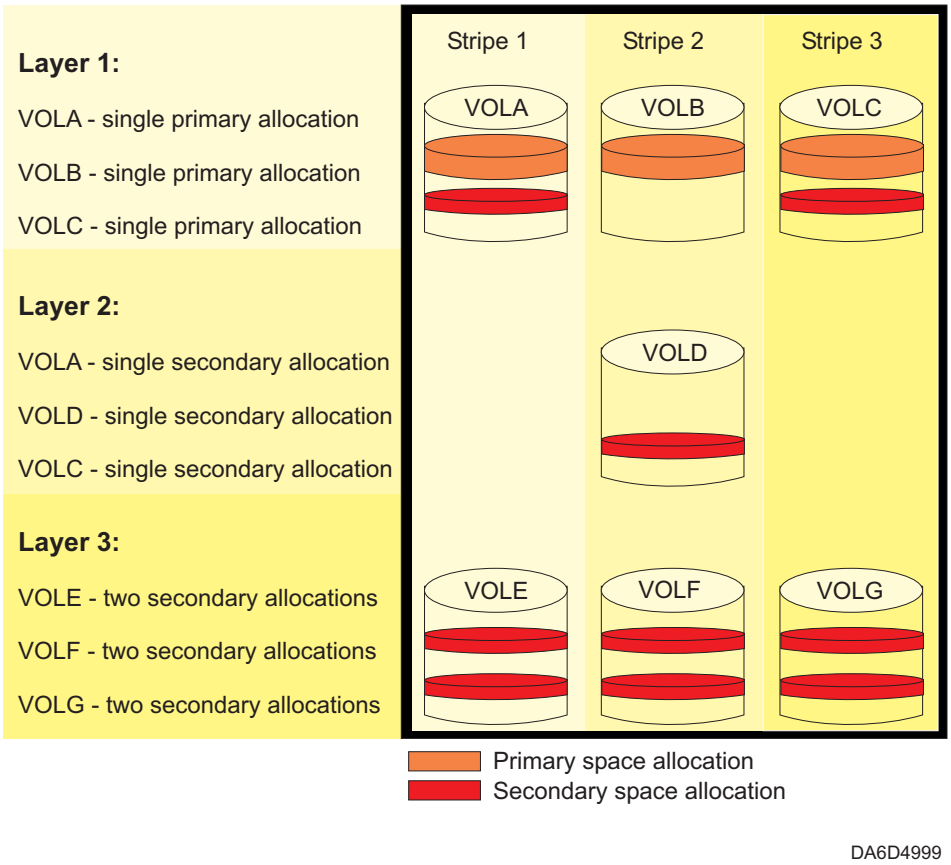


Figure 9. Primary and Secondary Space Allocations for Striped Data Sets

Figure 10 shows examples of the CIs within a control area (CA) on multiple volumes for a four-stripe VSAM data set.

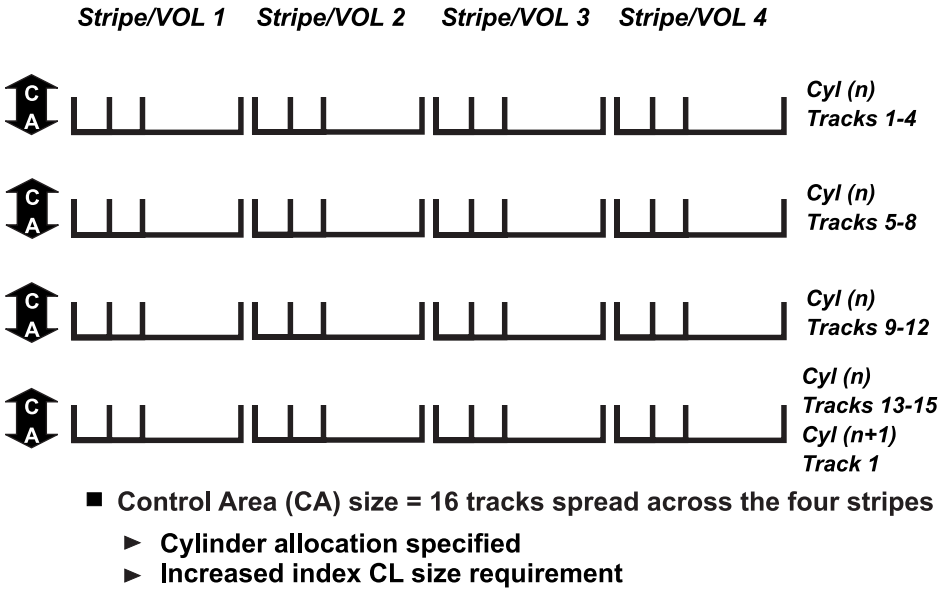


Figure 10. Control Interval / Control Area

Layering Concept for Data Striping

Layering is a concept generally associated with data that is striped. A layer in a striped environment is defined as the relationship of the volumes that make up the total number of stripes. That is, those volumes that will participate as part of the I/O packet. Once any volume or volumes, up to a maximum of stripe count, composing this I/O packet changes, this constitutes another layer. As relates to striped data, the volumes that constitute this I/O packet should be viewed in the same context as a single volume data set, as opposed to multivolume if the data were not striped. Once the data set extends to a second layer, this would be analogous to a multivolume nonstriped data set. Again, the definition of striped is a stripe count greater than 1. The sequential access method (SAM) does not support the concept of multilayering. VSAM supports multilayering. Figure 11 shows an example of the concept of layering with a four-stripe data set.

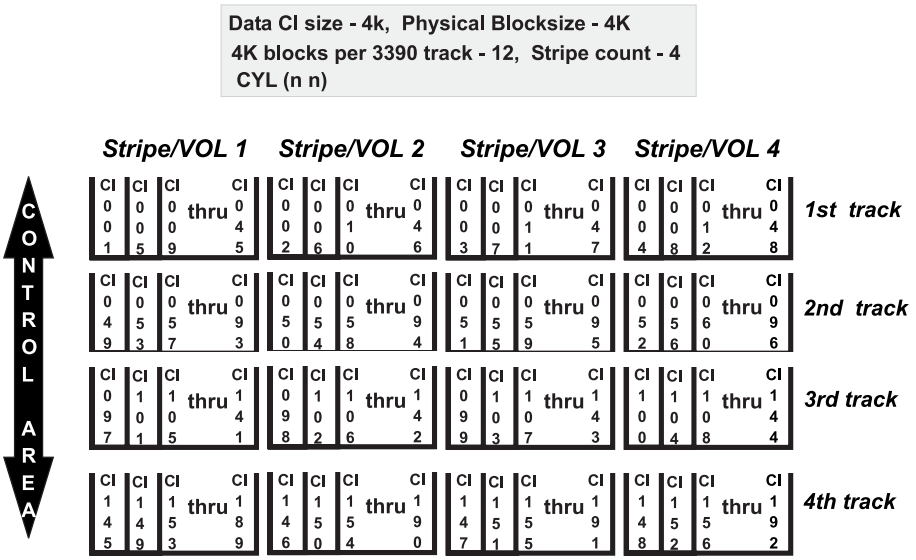


Figure 11. Layering (Four Stripe Data Set)

Other Considerations for Data Striping

To use data striping, you also need to consider space allocation, control area size, and processing.

Space Allocation for Striped VSAM Data Sets

The general rules discussed for striped extended format data sets apply to striped VSAM data sets. When the system allocates space for a striped extended-format data set, the system divides the primary amount among the volumes. If it does not divide evenly, the system rounds up the amount. For extended-format data sets, when the primary space on any volume is full, the system allocates space on that volume. The amount is the secondary amount divided by the number of stripes. If the secondary amount does not divide evenly, the system rounds up the amount.

Some additional considerations apply to the control area (CA) for VSAM. All allocations must be rounded to a CA boundary. The number of stripes influences the size of the control area, resulting in some differences in allocation quantity required to meet the stripe count and CA requirements. The following section on CA size considerations discusses this in more detail.

All data set extends are as described for striped data set extends. Basically, the system divides the secondary amount by the stripe count and allocates the result to each stripe. This occurs in all cases, including a data set with the guaranteed space attribute from the associated storage class (SC), as well as extending to a new layer.

Secondary Allocation Quantity of Zero: When you specify a secondary allocation quantity of zero for nonstriped VSAM data sets, an extend causes the allocation to occur on a new volume, using the primary-space amount. You can also use this feature to spread data over multiple volumes. For striped VSAM data sets, extensions occur by stripe and can occur on the same volume or on a new volume, using the primary-space amount when a secondary-space amount of zero is specified.

Increased Number of Extents: A striped VSAM data set can have 255 extents per stripe. Because a striped VSAM data set can have a maximum of 16 stripes, that means a striped component can have a maximum of 4080 extents per striped component. For a striped KSDS with 16 stripes, the VSAM KSDS data set could have a maximum of 4335 extents (4080 for the data component and 255 for the index component).

Allocation Restrictions: The Space Constraint Relief attribute will not be considered for striped data sets. The intended purposes for data striping follow:

- Spread the data across volumes (a basic implementation technique for any data that is striped).
- Provide >5 extent relief (completed for all allocations for VSAM striped data, regardless of the specification).

Control Area Size Calculation

The control area (CA) size for striped VSAM data is a factor of the stripe count stripes. A VSAM striped data set can be striped up to a count of 16. The minimum size for an allocation is a single track. The maximum CA size is a cylinder. Traditionally that would have meant that the maximum CA size, based on 3390 geometry, would be 15 tracks. That changes with striped VSAM data set in that the maximum CA size now has to accommodate the maximum stripe count (16), and the maximum CA now becomes 16 tracks.

The required allocation quantity now becomes a factor of both user specified amount and stripe count. As an example, take a specification for space of TRACKS(1 1) with the following results:

- For nonstriped, traditional VSAM, a control areas size of one track with a resulting primary and secondary allocation quantity of 1 track.
- For a striped data set with a striped count = maximum =16, the control area size is then 16 tracks with a resulting primary and secondary quantity of 16 tracks.

A larger CA results in a larger number of control intervals (CIs) in the CA, resulting in a larger number of entries to index in a data set organization containing an index (KSDS and VRRDS), with the end result being a requirement for a larger index CI size.

Processing Considerations for Striped Data Sets

The basic restrictions associated with data sets in the extended format also apply to striped data sets.

In addition, VSAM striped data sets do not support:

- RLS access

- Improved CI access

For the alternate index, neither the data nor the index will be striped.

Selection of VSAM Data Set Types

VSAM supports several data set types: entry-sequenced (ESDS), key-sequenced (KSDS), linear (LDS), fixed-length, and variable-length relative record (RRDS). Before you select a data set type, consider the following questions:

- Will you need to access the records in sequence, randomly, or both ways?
- Are all the records the same length?
- Will the record length change?
- How often will you need to move records?
- How often will you need to delete records?
- Do you want spanned records?
- Do you want to keep the data in order by the contents of the record?
- Do you want to access the data by an alternate index?
- Do you want to use access method services utilities with an IBM DB2 cluster?

Entry-sequenced data sets are best for the following kinds of applications:

- Applications that require sequential access only. It is better to use entry-sequenced data sets or variable-length RRDSs for sequential access, because they support variable-length records and can be expanded as records are added.
- Online applications that need to use an existing entry-sequenced data set. If you want to use an entry-sequenced data set in an online application, load the data set sequentially by a batch program and access the data set directly by the relative byte address (RBA).

Key-sequenced data sets are best for the following kinds of applications:

- Applications that require that each record have a key field.
- Applications that require both direct and sequential access.
- Applications that use high-level languages which do not support RBA use.
- Online applications usually use key-sequenced data sets.
- You want to access the data by an alternate index.
- The advantage of key-sequenced data sets over fixed-length RRDS using direct access is ease of programming.
- You want to have compressed data.

Linear data sets, although rarely used, are best for the following kinds of applications:

- Specialized applications that store data in linear data sets
- Data-in-virtual (DIV)

Relative-record data sets are best for the following kinds of applications:

- Applications that require direct access only.
- Applications in which there is a one-to-one correspondence between records and relative record numbers. For example, you could assign numeric keys to records sequentially, starting with the value 1. Then, you could access a RRDS both sequentially and directly by key.
- Fixed-length RRDSs use less storage and are usually faster at retrieving records than key-sequenced data sets or variable-length RRDSs.

- If the records vary in length, use a variable-length RRDS.
- Variable-length RRDSs can be used for COBOL applications.

Entry-Sequenced Data Sets

An entry-sequenced data set is comparable to a sequential (non-VSAM) data set. It contains records that can be either spanned or nonspanned. As Figure 12 shows, records are sequenced by the order of their entry in the data set, rather than by a key field in the logical record.

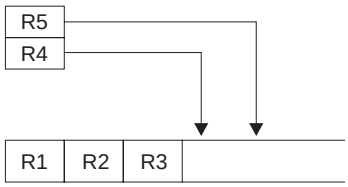


Figure 12. Entry-Sequenced Data Set

Records are added only at the end of the data set. Existing records cannot be deleted. If you want to delete a record, you must flag that record as inactive. As far as VSAM is concerned, the record is not deleted. Records can be updated, but they cannot be lengthened. To change the length of a record in an entry-sequenced data set, you must store it either at the end of the data set (as a new record) or in the place of a record of the same length that you have flagged as inactive or that is no longer required.

When a record is loaded or added, VSAM indicates its relative byte address (RBA). The RBA is the offset of this logical record from the beginning of the data set. The first record in a data set has an RBA of 0. The value of the RBA for the second and subsequent records depends on whether the file is spanned and on the control interval size chosen for the file, either manually or automatically. In general, it is not possible to predict the RBA of each record, except for the case of fixed-length records and a known control interval size. For a more detailed description of the internal format of VSAM files, see “VSAM Data Formats” on page 75.

You build an alternate index to keep track of these RBAs. Although an entry-sequenced data set does not contain an index component, alternate indexes are permitted. See “Defining Alternate Indexes” on page 120. Figure 13 shows the record lengths and corresponding RBAs for the data set shown in Figure 12.

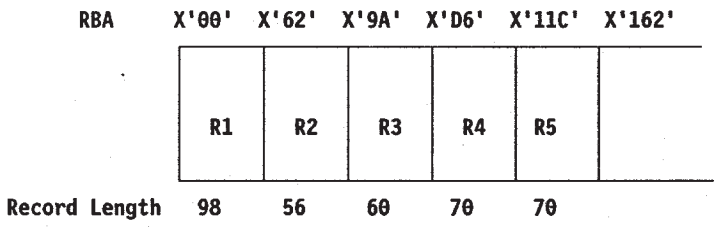


Figure 13. Example of RBAs of an Entry-Sequenced Data Set

Organizing VSAM Data Sets

Table 3 lists the operations and types of access for processing entry-sequenced data sets.

Table 3. Entry-Sequenced Data Set Processing

Operation	Sequential Access	Direct Access
Loading the data set	Yes	No
Adding records	Space after the last record is used for adding records	No
Retrieving records	Yes (returned in entry sequence)	Yes (by RBA)
Updating records	Yes, but you cannot change the record length	Yes (by RBA), but you cannot change the record length
Deleting records	Records cannot be deleted, but you can reuse its space for a record of the same length	Records cannot be deleted, but you can reuse its space for a record of the same length

Simulated VSAM Access to USS files

You can have simulated VSAM access to a USS file (simulated as an ESDS) by specifying PATH=pathname in the JCL DD statement, SVC 99, or TSO ALLOCATE command. For information about access using MVS access methods, see "Processing USS Files with an Access Method" on page 18.

When you use simulated VSAM, the system attempts to support a USS file compatibly, so that the application program sees the file as if it were an ESDS.

Because it does not actually store USS files as ESDSs, the system cannot simulate all the characteristics of an ESDS. Certain macros and services have incompatibilities or restrictions when dealing with USS files.

For information about VSAM interfaces and HFS files, see *z/OS DFSMS Macro Instructions for Data Sets*.

Record Processing for USS Files

Record boundaries are not maintained within binary files. Text files are presumed to be EBCDIC. Repositioning functions (such as POINT, CLOSE TYPE=T, GET DIRECT) are not permitted for FIFO or character special files.

When a file is accessed as binary, each record will be returned as length ACB RPLBUFL (move mode), except, possibly, the last one.

When a file is accessed as text, if any record in the file consists of zero bytes (that is, a text delimiter is followed by another text delimiter), the record returned will consist of one blank. If any record is longer than ACB RPLBUFL bytes, it will result in an error return code for GET (for an ACB).

Restrictions on USS Files

The following VSAM restrictions are associated with USS files:

- Only ESDS is simulated.
- No file sharing or buffer sharing is supported except for multiple readers and, of course, for a reader and a writer for FIFO.
- STRNO > 1 is not supported.
- Chained RPLs are not supported.
- Shared resources is not supported.
- Updated and backward processing is not supported.
- Direct processing or POINT for FIFO and character special files are not supported.
- There is no catalog support for HFS files.
- Alternate indexes are not supported.
- There is no support for JRNAD, UPAD, or the EXCEPTION exit.
- There is no cross-memory support.
- ERASE, VERIFY, and SHOWCAT are not supported.
- Certain SHOWCB requests return dummy values.
- Variable-length binary records do not retain record boundaries during conversion to a byte stream. During reading, each record, except the last, is assumed to be the maximum length.
- To specify the maximum record size, code the LRECL keyword on the JCL DD statement, SVC 99, or TSO ALLOCATE. If not specified, the default is 32 767.
- On return from a synchronous PUT or a CHECK associated with an asynchronous PUT, it is not guaranteed that data written has been synchronized to the output device. To ensure data synchronization, use ENDREQ, CLOSE, or CLOSE TYPE=T.
- There is no CI (control interval) access (MACRF=CNV).

Services and Utilities for USS Files

The following services and utilities support USS files:

- Access method services (AMS) REPRO—REPRO by DD name is supported and uses QSAM.
- IDCAMS PRINT—PRINT by DD name is supported and uses QSAM. Instead of displaying ’RBA OF RECORD’, PRINT displays ’RECORD NUMBER’.
- DEVTYPE macro—DEVTYPE provides information related to the USS file. If PATH is specified in the DD statement, DEVTYPE returns a return code of 0, a UCBTYP simulated value of X’00000103’, and a maximum block size of 32 760, instead of giving return code 8, as DEVTYPE did before DFSMS V1R3.

The following services and utilities do not support USS files. Unless stated otherwise, these services and utilities return an error or unpredictable value when issued for a USS file:

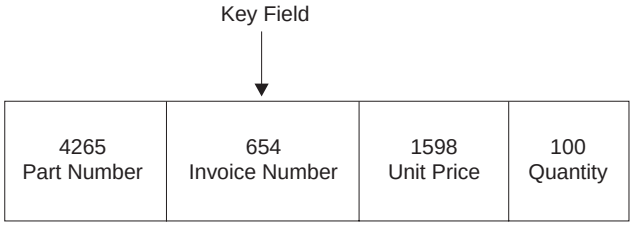
- Access method services (IDCAMS)—ALTER, DEFINE, DELETE, DIAGNOSE, EXAMINE, EXPORT, IMPORT, LISTCAT, and VERIFY
- OBTAIN, SCRATCH, RENAME, TRKCALC, and PARTREL macros

  These macros require a DSCB or UCB. HFS files do not have DSCBs or valid UCBs.

Guideline: ISPF Browse/Edit does not support USS files, but you can use the OBROWSE command.

Key-Sequenced Data Sets

In a key-sequenced data set, logical records are placed in the data set in ascending collating sequence by a field, called the key. Figure 14 shows that the key contains a unique value, such as an employee number or invoice number, that determines the record's collating position in the data set.



The key must be:

- Unique
- In the same position in each record
- In the first segment of a spanned record

Figure 14. Record of a Key-Sequenced Data Set

The key must be in the same position in each record, the key data must be contiguous, and each record's key must be unique. After it is specified, the value of the key cannot be altered, but the entire record can be erased or deleted. For compressed data sets, the key itself and any data before the key will not be compressed.

When a new record is added to the data set, it is inserted in its collating sequence by key, as shown in Figure 15.

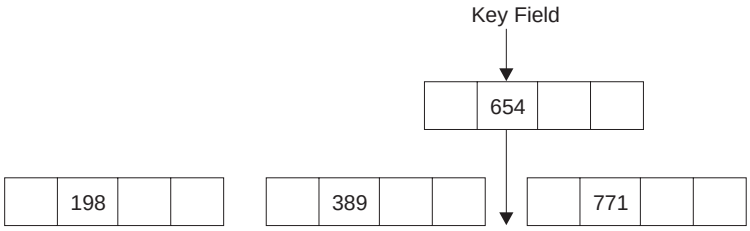


Figure 15. Inserting Records into a Key-Sequenced Data Set

Table 4 lists the operations and types of access for processing key-sequenced data sets.

Table 4. Key-Sequenced Data Set Processing

Operation	Sequential Access	Direct or Skip-Sequential Access
Loading the data set	Yes	No
Adding records	Yes (records must be written in key sequence)	Yes (records are added randomly by key)
Retrieving records	Yes (records are returned in key sequence)	Yes (by key)
Updating records	Yes	Yes
Deleting records	Yes	Yes

Free Space

When a key-sequenced data set is defined, unused space can be scattered throughout the data set to permit records to be inserted or lengthened. The unused space is called free space. When a new record is added to a control interval (CI) or an existing record is lengthened, subsequent records are moved into the following free space to make room for the new or lengthened record. Conversely, when a record is deleted or shortened, the space given up is reclaimed as free space for later use. When you define your data set, use the FREESPACE parameter to specify what percentage of each CI is to be set aside as free space when the data set is initially loaded.

Within each CA, reserve free space by using free CIs. If you have free space in your CA, it is easier to avoid splitting your control area when you want to insert additional records or lengthen existing records. When you define your data set, specify what percentage of the control area is to be set aside as free space, using the FREESPACE parameter.

See “Optimizing Free Space Distribution” on page 166 for information about specifying the optimal amount of CI and CA free space.

Considerations for Increasing Keys and Space

The structure of VSAM prime indexes for a KSDS and a VRRDS is built to create a single index record at the lowest level of the index, the sequence set, to provide pointers to each CI within a single CA. Each entry contains a compressed value representing the highest key that can be contained within that control interval. The value stored for the control interval containing records with the highest key in that control area represents the highest record-key value that can be contained in that control area. Once all the records are deleted from any single control interval, the current high-key value is no longer associated with that control interval’s entry in the sequence set record. It becomes a “free” control interval in which records containing any key within the range of keys for that control area can be inserted. This is called a CI reclaim.

However, this does not apply when it is the last empty control interval within the control area. In that case, the high-key value for that control interval is maintained and it becomes the highest key for any record that can be inserted into that control area. There is no reclaim capability for control areas that is comparable to that provided for control intervals. What can occasionally be observed as a normal result of not reclaiming control areas is data sets that just continue to grow in size. This will result when applications continually add records with keys that are in ascending sequence, followed by another or the same application that deletes old records after they have undergone some type of processing. During the deletion processing, the high-key value that was associated with that CA will be maintained, requiring that only records falling within that high-key range are eligible for insertion into that control area.

Since the record keys are always getting higher, no additional records will qualify for insertion into these empty control areas. The result is a data set in which a majority of the space is occupied by empty control intervals. When such a condition is detected, the only option a user has to reclaim this space is to rebuild the data set. This will require a logical copy of the data set, followed by a deletion of the old data set and a reload operation from the logical copy.

For more information see *z/OS DFSMS: Managing Catalogs*.

Insertion of a Logical Record in a CI

Figure 16 shows how CI free space is used to insert and delete a logical record in a KSDS or variable-length RRDS.

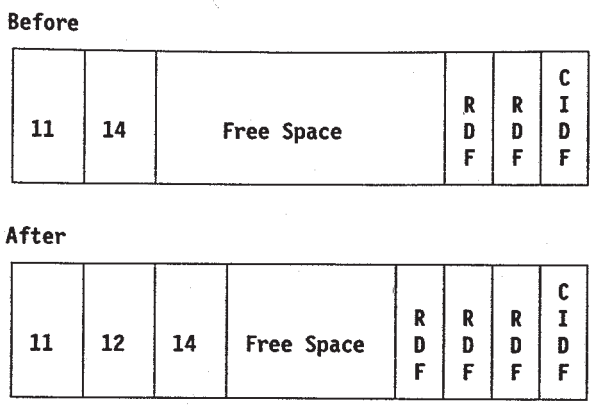


Figure 16. Inserting a Logical Record into a CI

Two logical records are stored in the first control interval shown in Figure 16. Each logical record has a key (11 and 14). The second control interval shows what happens when you insert a logical record with a key of 12.

1. Logical record 12 is inserted in its correct collating sequence in the CI.
2. The CI definition field (CIDF) is updated to show the reduction of available free space.
3. A corresponding record definition field (RDF) is inserted in the appropriate location to describe the length of the new record.

When a record is deleted, the procedure is reversed, and the space occupied by the logical record and corresponding RDF is reclaimed as free space.

Prime Index

A key-sequenced data set always has a prime index that relates key values to the relative locations of the logical records in a data set. The prime index, or simply index, has two uses in locating:

- The collating position when inserting records
- Records for retrieval

When initially loading a data set, records must be presented to VSAM in key sequence. The index for a key-sequenced data set is built automatically by VSAM as the data set is loaded with records.

When a data control interval is completely loaded with logical records, free space, and control information, VSAM makes an entry in the index. The entry consists of the highest possible key in the data control interval and a pointer to the beginning of that control interval.

Key Compression

The key in an index entry is stored by VSAM in a compressed form. Compressing the key eliminates from the front and back of a key those characters that are not necessary to distinguish it from the adjacent keys. Compression helps achieve a smaller index by reducing the size of keys in index entries. VSAM automatically does key compression in any key-sequenced data set. It is independent of whether the data set is in compressed format.

Control Interval Splits

When a data set is first loaded, the key sequence of data records and their physical order are the same. However, when data records are inserted, control interval splits can occur, causing the data control intervals to have a physical order that differs from the key sequence.

Linear Data Sets

A linear data set is a VSAM data set with a control interval size of 4096 bytes to 32 768 bytes in increments of 4096 bytes. A linear data set does not have imbedded control information. All linear data set bytes are data bytes.

A linear data set is processed as an entry-sequenced data set, with certain restrictions. Because a linear data set does not contain control information (CIDFs and RDFs), it cannot be accessed as if it contained individual records. Access a linear data set with the DIV macro. If you use DIV to access a data set, the control interval size must be 4096; otherwise, the data set will not be processed. See *z/OS MVS Programming: Assembler Services Guide* for information about using data-in-virtual (DIV).

Fixed-Length Relative-Record Data Sets

A fixed-length RRDS consists of several fixed-length slots. A fixed-length RRDS is defined using NUMBERED and a RECORDSIZE whose average and maximum lengths are the same.

Each slot has a unique relative record number, and the slots are sequenced by ascending relative record number. Each record occupies a slot and is stored and retrieved by the relative record number of that slot. The position of a data record is fixed; its relative record number cannot change. A fixed-length RRDS cannot have a prime index or an alternate index.

Because the slot can either contain data or be empty, a data record can be inserted or deleted without affecting the position of other data records in the fixed-length RRDS. The record definition field (RDF) shows whether the slot is occupied or empty. Free space is not provided in a fixed-length RRDS because the entire data set is divided into fixed-length slots.

In a fixed-length RRDS, each control interval contains the same number of slots. The number of slots is determined by the control interval size and the record length. Figure 17 shows the structure of a fixed-length RRDS after adding a few records. Each slot has a relative record number and an RDF. Table 5 on page 93 shows the access options available for RRDS processing.

1	2	3(E)	4(E)	RDF4	RDF3	RDF2	RDF1	CIDF
5	6(E)	7(E)	8(E)	RDF8	RDF7	RDF6	RDF5	CIDF
9	10(E)	11	12	RDF12	RDF11	RDF10	RDF9	CIDF

(E)-Empty Slot

Figure 17. Fixed-length Relative-Record Data Set

Table 5 lists the operations and types of access for processing fixed-length RRDSs.

Table 5. RRDS Processing

Operation	Sequential Access	Direct or Skip- Sequential Access
Loading the data set	Yes	Yes
Adding records	Yes (empty slots are used)	Yes (empty slots are used)
Retrieving records	Yes	Yes (by relative record number)
Updating records	Yes	Yes
Deleting records	Yes (a slot given up by a deleted record can be reused)	Yes (a slot given up by a deleted record can be reused)

Variable-Length Relative-Record Data Sets

A variable-length RRDS (VRRDS) is similar to a fixed-length RRDS, except that it contains variable-length records. Each record has a unique relative record number, and is placed in ascending relative record number order. Each record is stored and retrieved using its relative record number. Unlike a fixed-length RRDS, a variable-length RRDS does not have slots. The relative record number of a record cannot change. When that record is erased, the relative record number can be reused for a new record.

You must load the variable-length RRDS sequentially in ascending relative record number order. To define a variable-length RRDS, specify NUMBERED and RECORDSIZE. The average record length and maximum record length in RECORDSIZE must be different.

Free space is used for inserting and lengthening variable-length RRDS records. When a record is deleted or shortened, the space given up is reclaimed as free space for later use. When you define your data set, use the FREESPACE parameter to specify what percentage of each control interval and control area is to be set aside as free space when the data set is initially loaded. “Insertion of a Logical Record in a CI” on page 90 shows how free space is used to insert and delete a logical record.

A variable-length RRDSS cannot have spanned records and alternate indexes. VRRDS is a KSDS processed as an RRDS so a prime index is created.

Variable-length RRDS performance is similar to a key-sequenced data set, and is slower than for a fixed-length RRDS. Table 6 on page 94 shows the operations available for key-sequenced data sets and direct or skip-sequential access.

Table 6. Variable-Length RRDS Processing

Operation	Sequential Access	Direct or Skip- Sequential Access
Loading the data set	Yes, in ascending relative record number order	No
Adding records	Yes	Yes (free space is used)
Retrieving records	Yes	Yes (by relative record number)
Updating records	Yes	Yes
Deleting records	Yes (free space is reclaimed)	Yes (free space is reclaimed)

Summary of VSAM Data Set Types

Table 7 summarizes what each data set format offers.

Table 7. Comparison of ESDS, KSDS, Fixed-Length RRDS, Variable-Length RRDS, and Linear Data Sets

ESDS	KSDS	Fixed-Length RRDS	Variable-Length RRDS	Linear Data Sets
Records are in order as they are entered	Records are in collating sequence by key field	Records are in relative record number order	Records are in relative record number order	No processing at record level
Direct access by RBA	Direct access by key or by RBA	Direct access by relative record number	Direct access by relative record number	Access with data-in-virtual (DIV)
Alternate indexes permitted[1]	Alternate indexes permitted	No alternate indexes permitted	No alternate indexes permitted	No alternate indexes permitted
A record's RBA cannot change	A record's RBA can change	A record's relative record number cannot change	A record's relative record number cannot change	No processing at record level
Space at the end of the data set is used for adding records	Free space is used for inserting and lengthening records	Empty slots in the data set are used for adding records	Free space is used for inserting and lengthening records	No processing at record level
A record cannot be deleted, but you can reuse its space for a record of the same length[1]	Space given up by a deleted or shortened record becomes free space	A slot given up by a deleted record can be reused	Space given up by a deleted or shortened record becomes free space	No processing at record level
Spanned records permitted	Spanned records permitted	No spanned records	No spanned records	No spanned records
Extended format permitted	Extended format or compression permitted	Extended format permitted	Extended format permitted	Extended format permitted

Note:

1. Not supported for HFS files.

Extended-Format Data Sets

An extended-format data set for VSAM can be allocated for key-sequenced data sets, entry-sequenced data sets, variable-length or fixed-length relative-record data sets, and linear data sets.

Certain types of key-sequenced data set types are excluded. The following data sets cannot have an extended format:
- Catalogs
- Other system data sets
- Temporary data sets

When a data set is allocated as an extended format data set, the data and index are extended format. Any alternate indexes related to an extended format cluster are also extended format.

If a data set is allocated as an extended format data set, 32 bytes (x'20') are added to each physical block. Consequently, when the control interval size is calculated or explicitly specified, this physical block overhead may increase the amount of space actually needed for the data set. Figure 18 shows the percentage increase in space as indicated. Other control intervals do not result in an increase in needed space

```
- 3390 Direct Access Device

   Control Interval Size    Additional Space Required
             512                        2.1%
            1536                        4.5%
           18432                       12.5%

- 3380 Direct Access Device

   Control Interval Size    Additional Space Required
             512                        2.2%
            1024                        3.2%
```

Figure 18. Control Interval Size

Restrictions on Defining Extended-Format Data Sets

The following restrictions apply to defining extended-format data sets:
- An extended-format data set does not permit the indexes to be imbedded with the data (IMBED parameter) or the data to be split into key ranges (KEYRANGES parameter).
- Extended-format data sets must be SMS managed. The mechanism for requesting extended format for VSAM data sets is using the SMS data class DSNTYPE=EXT parameter and the subparameter R or P to indicate required or preferred.
- An open for improved control interval (ICI) processing is not permitted for extended format data sets.
- A checkpoint will not be permitted for an environment that has any extended format data sets open. The checkpoint fails with an IHJ000I message.
- Hiperbatch processing is not supported for extended format data sets.

Compressed Data

To use compression, a data set must be in extended format. Only extended-format key-sequenced data sets can be compressed. The compressed data records have a slightly different format than logical records in a data set that will not hold compressed data. This results in several incompatibilities that can affect the definition of the data set or access to records in the data set:

- The maximum record length for nonspanned data sets is three bytes less than the maximum record length of data sets that do not contain compressed data (this length is CISIZE-10).
- The relative byte address (RBA) of another record, or the address of the next record in a buffer, cannot be determined using the length of the current record or the length of the record provided to VSAM.
- The length of the stored record can change when updating a record without any length change.
- The key and any data in front of the key will not be compressed. Data sets with large key lengths and RKP data lengths might not be good candidates for compression.
- Only the data component of the base cluster is eligible for compression. Alternate indexes are not eligible for compression.
- The global shared resources (GSR) option is not permitted for compressed data sets.

In addition to these incompatibilities, the data set must meet certain requirements to permit compression at the time it is allocated:

- The data set must have a primary allocation of at least 5 MBs, or 8 MBs if no secondary allocation is specified.
- The maximum record length specified must be at least key offset plus key length plus forty bytes.
- Compressed data sets must be SMS managed. The mechanism for requesting compression for VSAM data sets is through the SMS data class COMPACTION=Y parameter.

Spanned record data sets require the key offset plus the key length to be less than or equal to the control interval size minus fifteen. These specifications regarding the key apply to alternate keys as well as primary keys.

Compressed data sets cannot be accessed using control interval (CI) processing except for VERIFY and VERIFY REFRESH processing and may not be opened for improved control interval (ICI) processing. A compressed data set can be created using the LIKE keyword and not just using a data class.

Access to Records in a VSAM Data Set

You can use addressed-sequential and addressed-direct access for the following types of data sets:
- Entry-sequenced data sets
- Key-sequenced data sets

You can use keyed-sequential, keyed-direct, and skip-sequential access for the following types of data sets:
- Key-sequenced data sets
- Fixed-length RRDSs
- Variable-length RRDS

All types of VSAM data sets, including linear, can be accessed by control interval access, but this is used only for very specific applications. CI mode processing is not permitted when accessing a compressed data set. The data set can be opened for CI mode processing to permit VERIFY and VERIFY REFRESH processing only. Control interval access is described in “Chapter 12. Processing Control Intervals” on page 183.

Access to Entry-Sequenced Data Sets

Entry-sequenced data sets are accessed by address, either sequentially or directly. When addressed sequential processing is used to process records in ascending relative byte address (RBA) sequence, VSAM automatically retrieves records in stored sequence.

To access a record directly from an entry-sequenced data set, you must supply the RBA for the record as a search argument. See “Entry-Sequenced Data Sets” on page 85 for information about obtaining the RBA.

Skip-sequential processing is not supported for entry-sequenced data sets.

Access to Key-Sequenced Data Sets

The most effective way to access records of a key-sequenced data set is by key, using the associated prime index.

Keyed-Sequential Access

Sequential access is used to load a key-sequenced data set and to retrieve, update, add, and delete records in an existing data set. When you specify sequential as the mode of access, VSAM uses the index to access data records in ascending or descending sequence by key. When retrieving records, you do not need to specify key values because VSAM automatically obtains the next logical record in sequence.

Sequential processing can be started anywhere within the data set. While positioning is not always required (for example, the first use of a data set starts with the first record), it is best to specify positioning using one of the following methods:

- Use the POINT macro.
- Issue a direct request with note string positioning (NSP), and change the request parameter list with the MODCB macro from “direct” to “sequential” or “skip sequential”.
- Use MODCB to change the request parameter list to last record (LRD), backward (BWD), and direct NSP; then change the RPL to SEQ, BWD, and SEQ.

Sequential access enables you to avoid searching the index more than once. Sequential is faster than direct for accessing multiple data records in ascending key order.

Keyed-Direct Access

Direct access is used to retrieve, update, delete and add records. When direct processing is used, VSAM searches the index from the highest level index-set record to the sequence-set for each record to be accessed. Searches for single records with random keys is usually done faster with direct processing. You need to supply a key value for each record to be processed.

For retrieval processing, either supply the full key or a generic key. The generic key is the high-order portion of the full key. For example, you might want to retrieve all records whose keys begin with the generic key AB, regardless of the full key value. Direct access lets you avoid retrieving the entire data set sequentially to process a small percentage of the total number of records.

Skip-Sequential Access

Skip-sequential access is used to retrieve, update, delete, and add records. When skip-sequential is specified as the mode of access, VSAM retrieves selected records, but in ascending sequence of key values. Skip-sequential processing lets you avoid retrieving a data set or records in the following inefficient ways:

- Entire data set sequentially to process a small percentage of the total number of records
- Desired records directly, which would cause the prime index to be searched from the top to the bottom level for each record

Addressed Access

Another way of accessing a key-sequenced data set is addressed access, using the RBA of a logical record as a search argument. If you use addressed access to process key-sequenced data, you should be aware that RBAs might change when a control interval split occurs or when records are added, deleted, or changed in size. With compressed data sets, the RBAs for compressed records are not predictable. Therefore, access by address is not suggested for normal use.

Access to Linear Data Sets

You can access a linear data set with the DIV macro.

Related reading. See *z/OS MVS Programming: Assembler Services Guide* for information about using data-in-virtual (DIV).

Access to Fixed-Length Relative-Record Data Sets

The relative record number is always used as a search argument for a fixed-length RRDS.

Keyed-Sequential Access

Sequential processing of a fixed-length RRDS is the same as sequential processing of an entry-sequenced data set. Empty slots are automatically skipped by VSAM.

Skip-Sequential Access

Skip-sequential processing is treated like direct requests, except that VSAM maintains a pointer to the record it just retrieved. When retrieving subsequent records, the search begins from the pointer, rather than from the beginning of the data set. Records must be retrieved in ascending sequence.

Keyed-Direct Access

A fixed-length RRDS can be processed directly by supplying the relative record number as a key. VSAM converts the relative record number to an RBA and determines the control interval containing the requested record. If a record in a slot flagged as empty is requested, a no-record-found condition is returned. You cannot use an RBA value to request a record in a fixed-length RRDS.

Access to Variable-Length Relative-Record Data Sets

The relative record number is used as a search argument for a variable-length RRDS.

Keyed-Sequential Access

Sequential processing of a variable-length RRDS is the same as for an entry-sequenced data set. On retrieval, relative record numbers that do not exist are skipped. On insert, if no relative record number is supplied, VSAM uses the next available relative record number.

Skip-Sequential Access

Skip-sequential processing is used to retrieve, update, delete, and add variable-length RRDS records. Records must be retrieved in ascending sequence.

Keyed-Direct Access

A variable-length RRDS can be processed directly by supplying the relative record number as a key. If you want to store a record in a specific relative record position, use direct processing and assign the desired relative record number. VSAM uses the relative record number to locate the control interval containing the requested record. You cannot use an RBA value to request a record in a variable-length RRDS.

Access to Records through Alternate Indexes

You can use access method services to define and build one or more alternate indexes over a key-sequenced or entry-sequenced data set, which is called the base cluster. An alternate index provides access to records by using more than one key. The alternate index accesses records in the same way as the prime index of a key-sequenced data set. An alternate index eliminates the need to store multiple copies of the same information for different applications. The alternate index is built from all the records in a base cluster. However, it is not possible to build an alternate index from only specific records in the base cluster.

Unlike a primary key, which must be unique, the key of an alternate index can refer to more than one record in the base cluster. An alternate-key value that points to more than one record is nonunique. If the alternate key points to only one record, the pointer is unique.

Restriction: The maximum number of nonunique pointers associated with an alternate index data record cannot exceed 32 767.

Alternate indexes are not supported for linear data sets, RRDS, or reusable data sets (data sets defined with the REUSE attribute). See “Defining Alternate Indexes” on page 120 for information about defining and building alternate indexes.

The alternate index is a key-sequenced data set; it consists of an index component and a data component. The records in the data component contain an alternate key and one or more pointers to data in the base cluster. For an entry-sequenced base cluster, the pointers are RBA values. For a key-sequenced base cluster, the pointers are primary-key values.

Each record in the data component of an alternate index is of variable length and contains header data, the alternate key, and at least one pointer to a base data record. Header data is fixed length and provides the following information:

- Whether the alternate index data record contains primary keys or RBA pointers
- Whether the alternate index data record contains unique or nonunique keys
- The length of each pointer
- The length of the alternate key
- The number of pointers

Figure 19 on page 101 shows the structure of an alternate index with nonunique keys connected to a key-sequenced data set. The salesman’s name is the alternate key in this example. The customer number is the primary key.

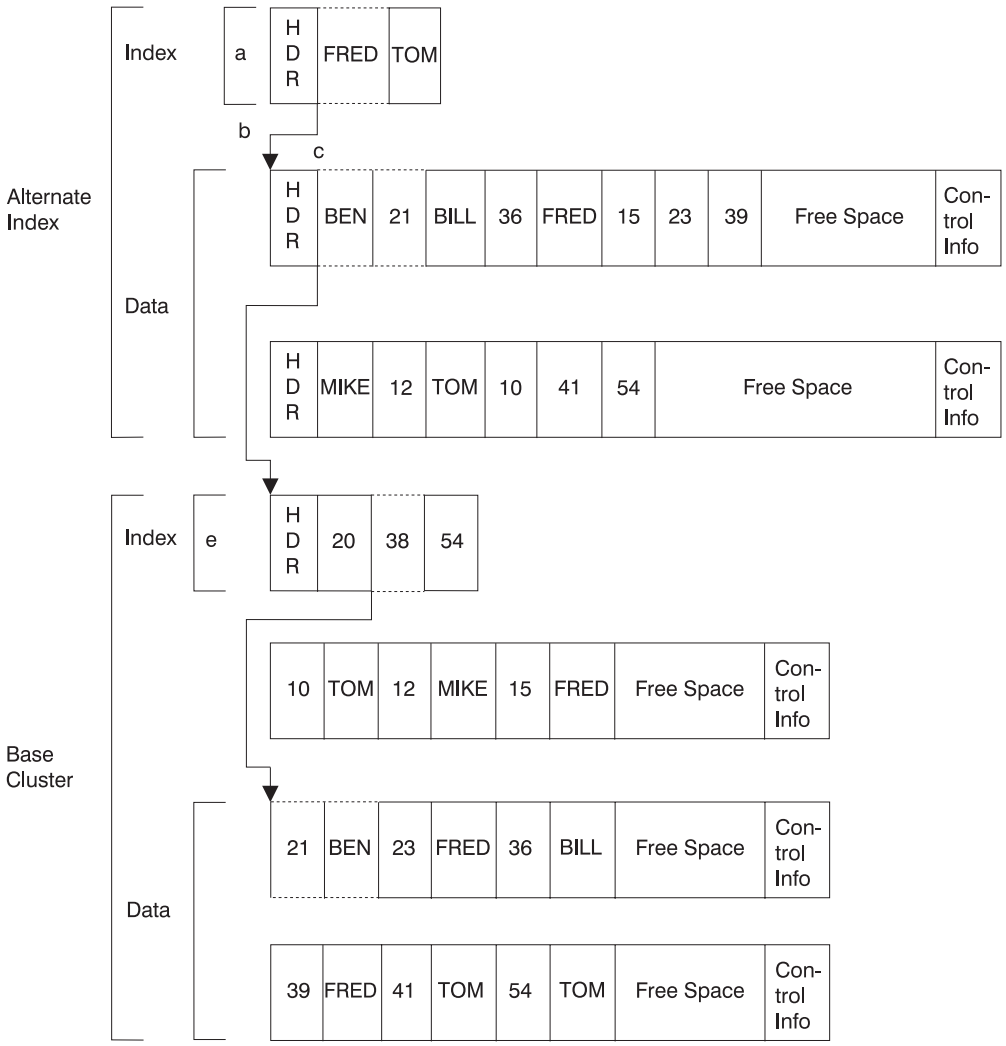


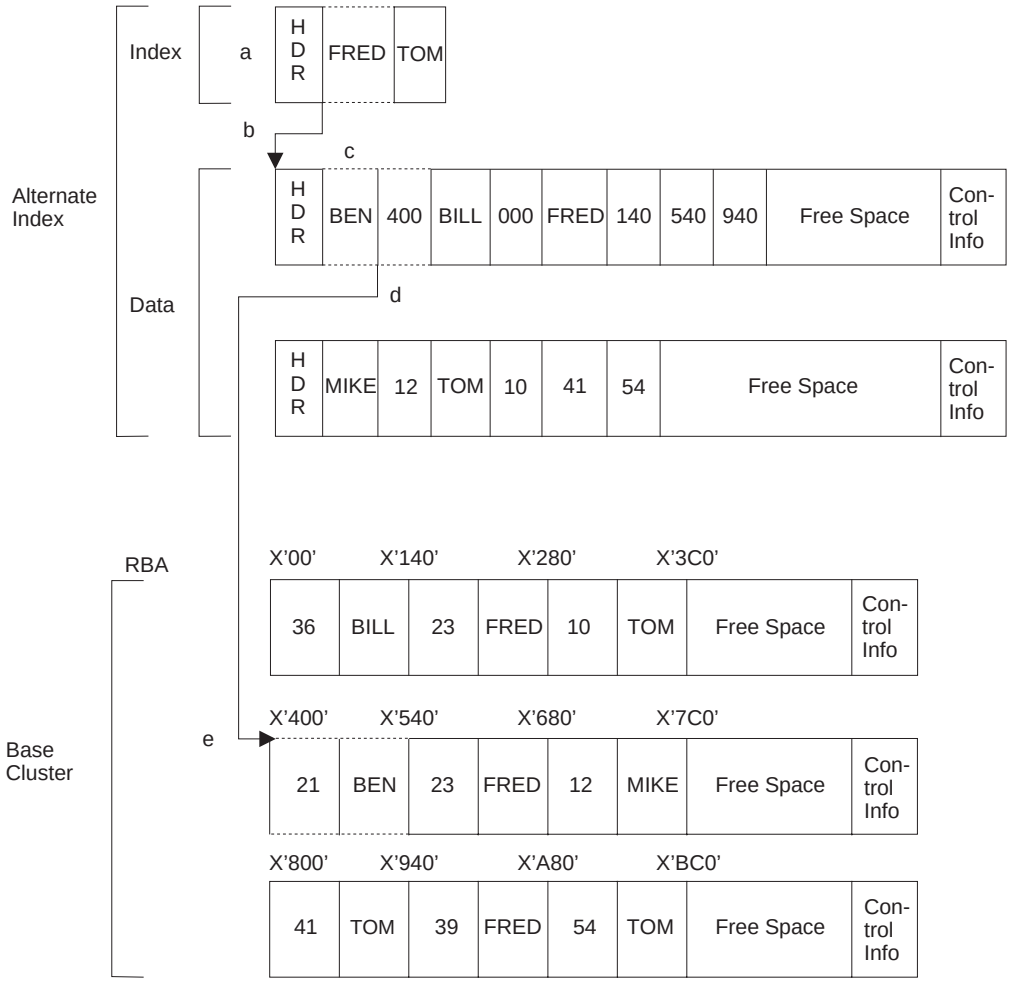
Figure 19. Alternate Index Structure for a Key-Sequenced Data Set

If you ask to access records with the alternate key of BEN, VSAM does the following:

1. VSAM scans the index component of the alternate index, looking for a value greater than or equal to BEN.
2. The entry FRED points VSAM to a data control interval in the alternate index.
3. VSAM scans the alternate index data control interval looking for an entry that matches the search argument, BEN.
4. When located, the entry BEN has an associated key, 21. The key, 21, points VSAM to the index component of the base cluster.
5. VSAM scans the index component for an entry greater than or equal to the search argument, 21.
6. The index entry, 38, points VSAM to a data control interval in the base cluster. The record with a key of 21 is passed to the application program.

RBAs are always written as fullword binary integers.

Figure 20 illustrates the structure of an alternate index connected to an entry-sequenced data set. The salesman's name is the alternate key in this example.



Note: RBAs are always written as fullword binary integers.

Figure 20. Alternate Index Structure for an Entry-Sequenced Data Set

If you ask to access records with the alternate key of BEN, VSAM does the following:

1. VSAM scans the index component of the alternate index, looking for a value greater than or equal to BEN.
2. The entry FRED points VSAM to a data control interval in the alternate index.
3. VSAM scans the alternate index data control interval looking for an entry that matches the search argument, BEN.
4. When located, the entry BEN has an associated pointer, 400, that points to an RBA in the base cluster.
5. VSAM retrieves the record with an RBA of X'400' from the base cluster.

When you build an alternate index, the alternate key can be any field in the base data set's records that has a fixed length and a fixed position in each record. The alternate-key field must be in the first segment of a spanned record. Keys in the data component of an alternate index are not compressed; the entire key is represented in the alternate-index data record.

A search for a given alternate key reads all the base cluster records containing this alternate key. For example, Figure 19 on page 101 and Figure 20 on page 102 show that one salesman has several customers. For the key-sequenced data set, several primary-key pointers (customer numbers) are in the alternate-index data record. There is one for each occurrence of the alternate key (salesman's name) in the base data set. For the entry-sequenced data set, several RBA pointers are in the alternate index data record. There is one for each occurrence of the alternate key (salesman's name) in the base data set. The pointers are ordered by arrival time.

Before a base cluster can be accessed through an alternate index, a path must be defined. A path provides a way to gain access to the base data through a specific alternate index. To define a path use the access method services command DEFINE PATH.

See "Defining Alternate Indexes" on page 120 for information about defining an alternate index. See "Defining a Path" on page 124 for information about defining a path.

Data Compression

When deciding whether to compress data, consider the following guidelines and rules:

- Compress when an existing data set is approaching the 4 gigabyte VSAM size limit or when you have capacity constraints
- Only SMS-managed data is eligible for compression
- The data set must be an extended format key-sequenced data set
- Control interval access is not permitted.
- Compression could require excessive amounts of storage when processed in locate mode (OPTCD=LOC).
- The GSR option is not permitted for compressed data sets.

You can convert an application to compression processing if the application uses data that can be highly compressible based on the structure or type of data. One consideration could be the length of the data records:

- The records can be large relative to the size of a control interval.
- Smaller control interval sizes can be desirable because of the random structure of the data.
- When a smaller control interval size is used without compressing data records, the length of the records can require a spanned data set. Records placed in a spanned data set are less likely to span control intervals when compression is used. The result could improve performance when VSAM processes the data because the amount of input/output required to GET or PUT the record is reduced.

Any program other than DFSMSdss, REPRO, and any other physical data copy/move program that does direct input/output to DASD for data sets which have data in compressed format can compromise data integrity. These programs must be modified to access the data using VSAM keyed access to permit expansion of compressed data.

Chapter 8. Defining VSAM Data Sets

This chapter covers the following topics.

Topic	Page
Using Cluster Names for Data and Index Components	106
Defining a Data Set with Access Method Services	106
Defining a Data Set with JCL	114
Loading a VSAM Data Set	115
Copying and Merging Data Sets	119
Defining Alternate Indexes	120
Defining a Page Space	124
Checking for Problems in Catalogs and Data Sets	126
Deleting Data Sets	127

This chapter explains how to define VSAM data sets. Other chapters provide examples and related information:

- For an example of defining a VSAM data set, see “Chapter 9. Defining and Manipulating VSAM Data Sets: Examples” on page 129.
- For examples that show various aspects of defining VSAM data sets, see *z/OS DFSMS Access Method Services*.
- For information about defining a data set using RLS, see “Locking” on page 229.
- For information about allocating an HFS file to be accessed through MVS access methods, see “Chapter 1. Introducing Data Sets” on page 3.

VSAM data sets are defined using either access method services commands or JCL dynamic allocation. A summary of defining a VSAM data sets follows:

1. VSAM data sets must be cataloged. If you want to use a new catalog, use access method services commands to create a catalog. The procedure for defining a catalog is described in *z/OS DFSMS: Managing Catalogs*.
2. Define a VSAM data set in a catalog using the TSO ALLOCATE command, the access method services ALLOCATE or DEFINE CLUSTER command, dynamic allocation, or JCL. Before you can define a VSAM data set with dynamic allocation or JCL, SMS must be active on your system. Dynamic allocation and JCL do not support most of the DEFINE options available with access method services.
3. Load the data set with either the access method services REPRO command or your own loading program.
4. Optionally, define any alternate indexes and relate them to the base cluster. Use the access method services DEFINE ALTERNATEINDEX, DEFINE PATH, and BLDINDEX commands to do this.

After any of these steps, you can use the access method services LISTCAT and PRINT commands to verify what has been defined, loaded, or processed. The LISTCAT and PRINT commands are useful for identifying and correcting problems.

© Copyright IBM Corp. 1987, 2002

Using Cluster Names for Data and Index Components

For a key-sequenced data set, a cluster is the combination of the data component and the index component. The cluster provides a way to treat the index and data components as a single component with its own name. You can also give each component a name. Fixed-length RRDSs, entry-sequenced data sets, and linear data sets are considered to be clusters without index components. To be consistent, cluster names are normally used for processing these data sets.

Defining a Data Set with Access Method Services

VSAM data sets can be defined with either the DEFINE CLUSTER command or the ALLOCATE command. When a cluster is defined, VSAM uses the following catalog entries to describe the cluster:

- A cluster entry describes the cluster as a single component.
- A data entry describes the cluster's data component.
- For a key-sequenced data set, an index entry describes the cluster's index component.

All of the cluster's attributes are recorded in the catalog. The information that is stored in the catalog provides the details needed to manage the data set and to access the VSAM cluster or the individual components.

If you use DEFINE CLUSTER, attributes of the data and index components can be specified separately from attributes of the cluster.

- If attributes are specified for the cluster and not the data and index components, the attributes of the cluster (except for password and USVR security attributes) apply to the components.
- If an attribute that applies to the data or index component is specified for both the cluster and the component, the component specification overrides the cluster's specification.

If you use ALLOCATE, attributes can be specified only at the cluster level.

Naming a Cluster

You specify a name for the cluster when defining it. Usually, the cluster name is given as the `dsname` in JCL. A cluster name that contains more than 8 characters must be segmented by periods; 1 to 8 characters can be specified between periods. A name with a single segment is called an unqualified name. A name with more than 1 segment is called a qualified name. Each segment of a qualified name is called a qualifier.

You can, optionally, name the components of a cluster. Naming the data component of an entry-sequenced cluster or a linear data set, or the data and index components of a key-sequenced cluster, makes it easier to process the components individually.

If you do not explicitly specify a data or index component name when defining a VSAM data set or alternate index, VSAM generates a name. Also, when you define a user catalog, VSAM generates only an index name for the user catalog (the name of the user catalog is also the data component name). VSAM uses the following format to generate names for both system-managed and non-system-managed data sets:

1. If the last qualifier of the name is `CLUSTER`, replace the last qualifier with `DATA` for the data component and `INDEX` for the index component.

   ```
   Cluster name:  SALES.REGION2.CLUSTER
   Generated data name = SALES.REGION2.DATA
   Generated index name = SALES.REGION2.INDEX
   ```

2. ELSE if the cluster name is less than or equal to 38 characters, then append `.DATA` to the end of the cluster name for the data component and a `.INDEX` for the index component.

   ```
   Cluster name:  DEPT64.ASSET.INFO
   Generated data name = DEPT64.ASSET.INFO.DATA
   Generated index name = DEPT64.ASSET.INFO.INDEX
   ```

3. ELSE if the cluster name is between 39 and 42 characters inclusive, then append a `.D` to the end of the cluster name for the data component and a `.I` for the index component.

   ```
   Cluster name:  DEPTABCD.RESOURCE.REGION66.DATA1234.STUFF
   Generated data name = DEPTABCD.RESOURCE.REGION66.DATA1234.STUFF.D
   Generated index name = DEPTABCD.RESOURCE.REGION66.DATA1234.STUFF.I
   ```

4. ELSE if the name is longer than 42 characters, and the last qualifier is not `CLUSTER`, use the first (N-1) qualifiers of the cluster, alternate index, or user catalog name up to the first four qualifiers, and append as many 8-character qualifiers as necessary to produce a 5-qualifier name.

   ```
   Cluster name:  DIV012.GROUP16.DEPT98.DAILYLOG.DEC1988.BACK
   Generated data name = DIV012.GROUP16.DEPT98.DAILYLOG.TY7RESNO
   Generated index name = DIV012.GROUP16.DEPT98.DAILYLOG.YIIQHNTR
   ```

After a name is generated, VSAM searches the catalog to ensure that the name is unique. If a duplicate name is found, VSAM continues generating new names using the format outlined in 4 until a unique one is produced.

Duplicate Data Set Names

VSAM prevents you from cataloging two objects with the same name in the same catalog. VSAM also prevents you from altering the name of an object so that its new name duplicates the name of another object in the same catalog. VSAM does not prevent duplication of names from one catalog to another however. **If you have multiple catalogs, you should ensure that a data set name in one catalog is not duplicated in another catalog.** The multilevel alias facility assigns a data set name to a unique catalog. However, if the number of alias levels searched is changed, it is possible that duplicate data set names could occur. If the number of alias search levels is changed, the former name cannot be located with the multilevel alias facility unless the levels are changed back. See *z/OS DFSMS: Managing Catalogs*.

z/OS DFSMS Access Method Services describes the order in which one of the catalogs available to the system is selected to contain the to-be-defined catalog entry. When you define an object, you should ensure that the catalog the system selects is the catalog you want the object entered.

Data set name duplication is not prevented when a user catalog is imported into a system. No check is made to determine if the imported catalog contains an entry name that already exists in another catalog in the system.

Temporary Data Set Names

You can use the access method services ALLOCATE or TSO ALLOCATE command to define a temporary system-managed VSAM data set. A temporary system-managed data set can also be allocated directly through JCL.

Temporary system-managed VSAM data sets do not require that you specify a data set name. If you specify a data set name it must begin with & or &&:

```
DSNAME(&CLUSTER)
```

See “Examples of Defining Temporary VSAM Data Sets” on page 132 for information about using the ALLOCATE command to define a temporary system-managed VSAM data set. See “Temporary VSAM Data Sets” on page 268 for information about restrictions on using temporary data sets.

Specifying Cluster Information

When you define a cluster, you can directly specify certain descriptive, performance, security and integrity information. Other sources provide information you omit or the system does not let you provide.

If the Storage Management Subsystem (SMS) is active, and you are defining a system-managed cluster, you can explicitly specify the data class, management class, and storage class parameters and take advantage of attributes defined by your storage administrator. You can also implicitly specify the SMS classes by taking the system determined defaults if such defaults have been established by your storage administrator. The SMS classes are assigned only at the cluster level. You cannot specify them at the data or index level.

If SMS is active and you are defining a non-system-managed cluster, you can also explicitly specify the data class or take the data class default if one is available. Management class and storage class are not supported for non-system-managed data sets.

If you are defining a non-system-managed data set and you do not specify the data class, you must explicitly specify all necessary descriptive, performance, security, and integrity information through other access method services parameters. Most of these parameters can be specified for the data component, the index component, or both. Specify information for the entire cluster with the CLUSTER parameter. Specify information for only the data component with the DATA parameter and for only the index component with the INDEX parameter. See “Using Access Method Services Parameters” on page 109 for an explanation of the types of descriptive, performance, security, and integrity information specified using these parameters.

Both the data class and some other access method services parameters can be used to specify values to the same parameter, for example, the control interval size. The system uses the following order of precedence, or filtering, to determine which parameter value to assign.

1. Explicitly specified DEFINE command parameters
2. Modeled attributes (assigned by specifying the MODEL parameter on the DEFINE command)
3. Data class attributes
4. DEFINE command parameter defaults

Using Access Method Services Parameters

If you do not use the SMS classes to specify the necessary descriptive, performance, security, and integrity information, you must use access method services parameters.

Descriptive Parameters

The following access method services parameters provide descriptive information:

- **INDEXED|NONINDEXED|NUMBERED|LINEAR parameter**—Specifies the type of data organization used (key sequenced, entry sequenced, relative record, or linear).
- **RECORDSIZE parameter**—Specifies the average and maximum lengths of data records. The RECORDSIZE parameter is not used for a linear data set.

  A variable-length RRDS is defined using NUMBERED and RECORDSIZE, where the average and maximum record length must be different.

  If the actual length of an entry-sequenced, key-sequenced, or variable-length RRDS record is less than the maximum record length, VSAM saves disk space by storing the actual record length in the record definition field (RDF). The RDF is not adjusted for fixed-length RRDSs.
- **KEYS parameter**—Specifies the length and position of the key field in the records of a key-sequenced data set.
- **CATALOG parameter**—Specifies the name and password of the catalog in which the cluster is to be defined.
- **VOLUMES parameter**—Specifies the volume serial numbers of the volumes on which space is allocated for the cluster. You can specify up to 59 DASD volumes.
- **RECORDS|KILOBYTES|MEGABYTES|TRACKS|CYLINDERS parameter**—Specifies the amount of space to allocate for the cluster. The CYLINDERS, TRACKS, MEGABYTES, KILOBYTES, and RECORDS parameters are permitted for a linear data set. If you specify the RECORDS parameter for a linear data set, the system allocates space with the number of control intervals equal to the number of records. (Linear data sets do not have records; they have objects that are contiguous strings of data.)
- **RECATALOG parameter**—Specifies if an entry is recreated from information in the VSAM volume data set (VVDS), or defined for the first time.
- **REUSE|NOREUSE parameter**—Specifies if the cluster is reusable for temporary storage of data. See “Reusing a VSAM Data Set as a Work File” on page 118.
- **BUFFERSPACE parameter**—Specifies the minimum amount of I/O buffer space that must be allocated to process the data set. See “Determining I/O Buffer Space for Nonshared Resource” on page 169.

Performance Parameters

The following access method services parameters provide performance information. All these performance options are discussed in “Chapter 11. Optimizing VSAM Performance” on page 159.

- **CONTROLINTERVALSIZE parameter**—Specifies the control interval size for VSAM to use (instead of letting VSAM calculate the size).

  The size of the control interval must be large enough to hold a data record of the maximum size specified in the RECORDSIZE parameter unless the data set was defined with the SPANNED parameter.

  Specify the CONTROLINTERVALSIZE parameter for data sets that use shared resource buffering, so you know what control interval size to code on the BLDVRP macro.
- **SPANNED parameter**—Specifies whether records can span control intervals. The SPANNED parameter is not permitted for fixed-length and variable-length RRDSs, and linear data sets.
- **SPEED|RECOVERY parameter**—Specifies whether to preformat control areas during initial loading of a data set. See “Using a Program to Load a Data Set” on page 116.
- **VOLUMES parameter for the index component**—Specifies whether to place the cluster’s index on a separate volume from data.
- **FREESPACE parameter**—Specifies the amount of free space to remain in the data component of a key-sequenced data set or variable-length RRDS’s control intervals and control areas when the data records are loaded.
- **KEYRANGES parameter**—Specifies how data should be spread over multiple volumes.

Security and Integrity Parameters

The following access method services parameters provide security and integrity information. See “Chapter 5. Protecting Data Sets” on page 55 for more information about the types of data protection available.

- **Passwords**—Because passwords are not supported for system-managed data sets, this information pertains to non-system-managed data sets only. See *z/OS DFSMS Access Method Services*.
- **AUTHORIZATION parameter**—Specifies your own authorization routine to verify that a requester has the right to gain access to data.
- **EXCEPTIONEXIT parameter**—Specifies an I/O error-handling routine (the exception exit routine) that is entered if the program does not specify a SYNAD exit. See “Chapter 17. Coding VSAM User-Written Exit Routines” on page 241 for information about VSAM user-written exit routines.
- **WRITECHECK parameter**—Specifies whether to verify that write operations have completed and that the data can be read.
- **SHAREOPTIONS parameter**—Specifies whether and to what extent data is to be shared among systems, and jobs.
- **ERASE parameter**—Specifies whether to erase the information a data set contains when you delete the data set.

  To control the erasure of data in a VSAM component whose cluster is RACF protected and cataloged, you can use an ERASE attribute in a generic or discrete profile. For information about specifying and using the ERASE option, see “Erasing DASD Data” on page 62 and “Generic and Discrete Profiles for VSAM Data Sets” on page 56.

Allocating Space for VSAM Data Sets

When you define a data set, you or SMS must specify the amount of space to allocate for the data set. If SMS is active, you can specify a data class and take advantage of the space allocation set by your storage administrator. If you want to specify space explicitly, you can specify it for VSAM data sets in units of records, kilobytes, megabytes, tracks, or cylinders. To maintain device independence, specify records, kilobytes or megabytes. The amount of space you allocate depends on the size of your data set and the index options you selected. “Using Index Options” on page 180 explains the index options that improve performance.

If a guaranteed space storage class (STORAGECLASS parameter) is assigned to the data set and volume serial numbers are specified, primary space is allocated on all specified volumes if the following conditions are met. If these conditions are not met, the command fails and IGDxxxxI messages are printed:

- All volumes specified belong to the same storage group.
- The storage group to which these volumes belong is in the list of storage groups selected by the ACS routines for this allocation.

When the primary amount on the first volume is used up, a secondary amount is allocated on that volume. Each time a new record does not fit in the allocated space, the system allocates more space in the secondary space amount. This can be repeated until the volume is out of space or the extent limit is reached. For nonguaranteed space allocations, when you allocate space for your data set, you can specify both a primary and a secondary allocation. A primary space allocation is the initial amount of space allocated. A secondary allocation is the amount of space allocated each time space is used up. It is possible for the storage administrator to specify whether to use primary or secondary allocation amounts when extending to a new volume (only valid for nonstriped VSAM data sets). This is done with an SMS data class parameter for VSAM attributes.

You can expand the space for a VSAM data set defined in a catalog to a maximum of 251 to 255 extents. The system reservez the last four extents for extending a data set when the system cannot allocate the last extent in one piece.

You can specify space allocation at the cluster or alternate-index level, at the data level only, or at both the data and index levels. It is best to allocate space at the cluster or data levels. VSAM allocates space if:

- Allocation is specified at the cluster or alternate index level only, the amount needed for the index is subtracted from the specified amount. The remainder of the specified amount is assigned to data.
- Allocation is specified at the data level only, the specified amount is assigned to data. The amount needed for the index is in addition to the specified amount.
- Allocation is specified at both the data and index levels, the specified data amount is assigned to data and the specified index amount is assigned to the index.
- Secondary allocation is specified at the data level, secondary allocation must be specified at the index level or the cluster level.

VSAM acquires space in increments of control areas. The control area size generally is based on primary and secondary space allocations. See “Optimizing Control Area Size” on page 164 for information about optimizing control area size.

Partial Release

Partial release is used to release unused space from the end of an extended format data set and is specified through SMS management class or by the JCL RLSE subparameter. All space after the high used RBA is released on a CA boundary up to the high allocated RBA. If the high used RBA is not on a CA boundary, the high used amount is rounded to the next CA boundary. Partial release restrictions include:

- Partial release processing is supported only for extended format data sets.
- Only the data component of the VSAM cluster is eligible for partial release.
- Alternate indexes opened for path or upgrade processing are not eligible for partial release. The data component of an alternate index when opened as cluster could be eligible for partial release.
- Partial release processing is not supported for temporary close.
- Partial release processing is not supported for data sets defined with guaranteed space.

VSAM CLOSE will request partial release processing only if:

- Partial release was specified through SMS management class or by the JCL SPACE=(,,RLSE) parameter on the DD statement.
- The data set is defined as extended format.
- The data set was opened for OUTPUT processing.
- This is the last ACB closing for the data set (this includes all closes in the current address space, other address spaces in the system, and other systems).

Small Data Sets

If you allocate space for a data set in a unit smaller than one cylinder, VSAM allocates space in tracks when defining the data set. For data sets less than 1 cylinder in size, it is best to specify the maximum number of tracks required in the primary allocation for the data component, 1 track for the sequence set index (which should not be imbedded), and no secondary allocation for either data or index.

VSAM checks the smaller of primary and secondary space values against the specified device's cylinder size. If the smaller quantity is greater than or equal to the device's cylinder size, the control area is set equal to the cylinder size. If the smaller quantity is less than the device's cylinder size, the size of the control area is set equal to the smaller space quantity. The minimum control area size is one track. See "Optimizing Control Area Size" on page 164 for information about creating small control areas.

Multiple Cylinder Data Sets

First, calculate the amount of space needed for the primary allocation. If the data set is larger than one cylinder, calculate and specify the number of cylinders needed for data in a newly defined data set for the primary allocation. Make the secondary allocation equal to or greater than one cylinder, but less than the primary allocation. "Calculating Space for the Data Component of a KSDS" on page 113 demonstrates how to calculate the size of a data set.

See "Using Index Options" on page 180 for information about index options.

Linear Data Sets

You must allocate space in tracks, cylinders or records. Specifying space in bytes is not permitted.

When a linear data set is defined, the catalog forces the block size to 4096 bytes. You can specify a control interval size of 4096 to 32 768 bytes in increments of 4096 bytes. If not an integer multiple of 4096, the control interval size is rounded up to the next 4096 increment. If the specified BUFFERSPACE is greater than 8192 bytes, it is decremented to a multiple of 4096. If BUFFERSPACE is less than 8192, access method services issues a message and fails the command.

Calculating Space for the Data Component of a KSDS

You can use the following formula for any DASD. The number of blocks per track and control intervals per track depends on the DASD you are using. The following example shows how to calculate the size of the data component for a key-sequenced data set. The following are assumed for the calculations:

Device type. 3390
Unit of space allocation. Cylinders
Data control interval size. 1024 bytes
Physical block size (calculated by VSAM). 1024 bytes
Record size. 200 bytes
Free space definition – control interval. 20%
Free space definition – control area. 10%
Number of records to be loaded. 3000

You can calculate space for the data component as follows:

1. Number of bytes of free space (20% × 1024) = 204 (round down)
2. Number of loaded records per control interval (1024–10–204)/200=4.
3. Number of physical blocks per track = 33.
4. Number of control intervals per track = 33.
5. Maximum number of control intervals per control area (33 × 15)=495.
6. Number of loaded control intervals per control area = 446 (495 – 10% × 495).
7. Number of loaded records per cylinder (4 × 446) = 1784.
8. Total space for data component (3000/1784) (rounded) = 2 cylinders.

The value (1024 – 10) is the control interval length minus 10 bytes for two RDFs and one CIDF. The record size is 200 bytes. On an IBM 3380, 31 physical blocks with 1024 bytes can be stored on one track. The value (33 × 15) is the number of physical blocks per track multiplied by the number of data tracks per cylinder (16 minus 1 for the imbedded sequence set control interval).

Calculating Space for the Index Component

There is no specific formula for calculating the space required for the index component. Use the access method services command LISTCAT to see how much space was allocated to the index component.

Using ALTER to Modify Attributes of a Component

After a data set has been defined, you can change some of its attributes using the access method services command, ALTER. You identify the component by name, and specify the new attributes. ALTER can also be used to change an entry-sequenced data set, with the proper attributes, to a linear data set. The contents of the data set are not modified. See *z/OS DFSMS Access Method Services* for an example of changing an entry-sequenced data set to a linear data set.

You cannot use ALTER to change a fixed-length RRDS into a variable-length RRDS, or vice versa.

Using ALTER to Rename Data Sets

You can use the ALTER command to rename VSAM data sets and members of PDSs and PDSEs, and to convert VSAM data sets from non-system managed to system managed. ALTER can also convert system-managed data sets to non-system managed. See *z/OS DFSMS Access Method Services* to determine which values or attributes you can alter for a particular data set type.

Defining a Data Set with JCL

SMS must be active on your system before you can use JCL to define a VSAM data set. Any VSAM data set can be defined using JCL, except for a variable-length RRDS. Defining a VSAM data set using JCL has certain advantages. It takes less time to input and it makes syntax for defining the VSAM data set very similar to that used for accessing it.

See “Chapter 19. Using Job Control Language for VSAM” on page 265 for information about the JCL keywords used to define a VSAM data set. See *z/OS MVS JCL Reference* and *z/OS MVS JCL User’s Guide* for information about JCL keywords and the use of JCL.

DB2 provides striping on partitioned table spaces. Each of the partitions is a separate linear data set. Striping is used to perform parallel I/O concurrently against more than one of these partitions. The benefit of striping is only achievable if multiple partitions do not get allocated on the same volume. You can achieve volume separation without resorting to the storage class guaranteed space allocations on system-managed volumes.

Allocate all of the partitions in a single IEFBR14 job step using JCL. As long as there are an adequate number of volumes in the storage groups, and the volumes are not above the allocation threshold, the SMS allocation algorithms with SRM will ensure each partition is allocated on a separate volume.

DB2 striping is unrelated to VSAM striping (see “Extended-Format Data Sets” on page 95). You can use both DB2 striping and VSAM striping for the same set of linear extended format data sets.

Loading a VSAM Data Set

After a data set is defined, you can load records into it from a source data set. Depending on the type of VSAM data set being loaded, the source data set records might or might not need to be in a particular order.

- Records being loaded into an entry-sequenced data set do not have to be submitted in any particular order. Entry-sequenced data set records are sequenced by their time of arrival rather than by any field in the logical record.
- Fixed-length RRDS records are placed into slots specified either by a user-supplied or a VSAM-supplied relative record number. The relative record number is not part of the logical record, so it is not necessary that the records be submitted in any particular order.
- Records being loaded into a key-sequenced data set must be in ascending order by key, with no duplicate keys in the input data set.
- Records being loaded into a variable-length RRDS must be in ascending order by key, with no duplicate keys in the input data set. If they are loaded in sequential mode, VSAM assigns the relative record number.

With entry-sequenced or key-sequenced data sets, or RRDSs, you can load all the records either in one job or in several jobs. If you use multiple jobs to load records into a data set, VSAM stores the records from subsequent jobs in the same manner that it stored records from preceding jobs, extending the data set as required.

Using REPRO to Load a Data Set

The REPRO command lets you retrieve records from a sequential, indexed-sequential, or VSAM data set and store them in VSAM format in a key-sequenced, entry-sequenced, relative-record, or a sequential data set. The REPRO command is also used to load data from one linear data set into another linear data set.

When records are to be stored in key sequence, index entries are created and loaded into an index component as data control intervals and control areas are filled. Free space is left as indicated in the cluster definition in the catalog, and, if indicated in the definition, records are stored on particular volumes according to key ranges.

VSAM data sets must be cataloged. Sequential and indexed sequential data sets need not be cataloged. Sequential data sets that are system managed must be cataloged.

Defining VSAM Data Sets

If a sequential or indexed-sequential data set is not cataloged, include the appropriate volume and unit parameters on your DD statement. Also, supply a minimum set of DCB parameters when the input data set is sequential or indexed sequential, and/or the output data set is sequential. The following table shows the four key parameters:

Parameters	User Can Supply	Default if Not Supplied
DSORG	IS	PS
RECFM	F, FB, V, VB, VS, VBS	U
BLKSIZE	Block size	None
LRECL	Logical record length	BLKSIZE for F or FB BLKSIZE-4 for V, VB, VS, VBS

The DCB parameter DSORG must be supplied using the DD statement. The DCB parameters RECFM, BLKSIZE, and LRECL can be supplied using the DSCB or header label of a standard labeled tape, or by the DD statement. The system can determine the optimum block size.

If you are loading a VSAM data set into a sequential data set, you must remember that the 3-byte VSAM record definition field (RDF) is not included in the VSAM record length. When REPRO attempts to copy a VSAM record whose length is within 4 bytes of LRECL, a recoverable error occurs and the record is not copied.

Access method services does not support records greater than 32 760 bytes for non-VSAM data sets (LRECL=X is not supported). If the logical record length of a non-VSAM input data set is greater than 32 760 bytes, or if a VSAM data set defined with a record length greater than 32 760 is to be copied to a sequential data set, the REPRO command terminates with an error message.

Records in an indexed sequential data set that have a fixed-length, unblocked format with a relative-key position of zero are preceded by the key string when used as input. The records in the output data set must have a record length defined that includes the extended length caused by the key string. To copy “dummy” indexed-sequential records (with X'FF' in the first byte), specify the DUMMY option in the ENVIRONMENT parameter.

The REPRO operation is terminated if:
- One physical I/O error is found while writing to the output data set
- A total of four errors is found in any combination of the following:
  - Logical error while writing to the output data set
  - Logical error while reading the input data set
  - Physical error while reading the input data set

For information about physical and logical errors, see *z/OS DFSMS Macro Instructions for Data Sets*.

Using a Program to Load a Data Set

To use your own program to load a key-sequenced data set, first sort the records (or build them) in key sequence, then store them sequentially (using the PUT macro). When you are initially loading a data set, direct access is not permitted. For more information about inserting records into a data set, see “Inserting and Adding Records” on page 144.

VSAM uses the high-used RBA field to determine whether or not a data set is empty. An implicit verify can update the high-used RBA. Immediately after definition of a data set, the high-used RBA value is zero. An empty data set cannot be verified.

The terms create mode, load mode, and initial data set load are synonyms for the process of inserting records into an empty VSAM data set. To start loading an empty VSAM data set, call the VSAM OPEN macro. Following a successful open, the load continues while records are added and concludes when the data set is closed.

Restriction: If an entry-sequenced data set fails to load, you cannot open it.

Certain restrictions apply during load mode processing:

- PUT and CHECK are the only macros you can use.
- Do not use improved control interval processing.
- You cannot do update or input processing until the data set has been loaded and closed.
- Specify only one string in the ACB (STRNO>1 is not permitted).
- Do not specify local shared resources (LSR) or global shared resources (GSR).
- You cannot share the data set.
- Direct processing is not permitted (except relative record keyed direct).

If the design of your application calls for direct processing during load mode, you can avoid this restriction by following these steps:

1. Open the empty data set for load mode processing.
2. Sequentially write one or more records, which could be dummy records.
3. Close the data set to terminate load mode processing.
4. Reopen the data set for normal processing. You can now resume loading or do direct processing. When using this method to load a VSAM data set, be cautious about specifying partial release. Once the data set is closed, partial release will attempt to release all space not used.

For information about using user-written exit routines when loading records into a data set, see “Chapter 17. Coding VSAM User-Written Exit Routines” on page 241.

During load mode, each control area can be preformatted as records are loaded into it. Preformatting is useful for recovery if an error occurs during loading. However, performance is better during initial data set load without preformatting. The RECOVERY parameter of the access method services DEFINE command is used to indicate that VSAM is to preformat control areas during load mode. In the case of a fixed-length RRDS and SPEED, a control area in which a record is inserted during load mode will always be preformatted. With RECOVERY, all control areas will be preformatted.

Preformatting clears all previous information from the direct access storage area and writes end-of-file indicators. For VSAM, an end-of-file indicator consists of a control interval with a CIDF equal to zeros.

- For an entry-sequenced data set, VSAM writes an end-of-file indicator in every control interval in the control area.
- For a key-sequenced data set, VSAM writes an end-of-file indicator in the first control interval in the control area following the preformatted control area. (The preformatted control area contains free control intervals.)

- For a fixed-length RRDS, VSAM writes an end-of-file indicator in the first control interval in the control area following the preformatted control area. All RDFs in an empty preformatted control interval are marked “slot empty”.

As records are loaded into a preformatted control area of an entry-sequenced data set, an end-of-file indicator following the records indicates how far loading has progressed. You can then resume loading at that point, after verifying the data set. (You cannot open the data set unless you first verify it.) If an error occurs that prevents loading from continuing, you can identify the last successfully loaded record by reading to end of file.

The SPEED parameter does not preform at the data control areas. It writes an end-of-file indicator only after the last record is loaded. Performance is better if you use the SPEED parameter and if using extended format data sets. Extended format data sets may use system-managed buffering. This permits the number of data buffers to be optimized for load mode processing. This can be used with the REPRO parameter for a new data set for reorganization or recovery. If an error occurs that prevents loading from continuing, you cannot identify the last successfully loaded record and you might have to reload the records from the beginning. For a key-sequenced data set, the SPEED parameter only affects the data component.

Rule: Remember that, if you specify SPEED, it will be in effect for load mode processing. After load mode processing, RECOVERY will be in effect, regardless of the DEFINE specification.

Reusing a VSAM Data Set as a Work File

VSAM enables you to define reusable data sets to use as work files. Define the data set as reusable and specify that it be reset when you open it.

A data set that is not reusable can be loaded only once. After the data set is loaded, it can be read and written to, and the data in it can be modified. However, the only way to remove the set of data is to use the access method services command DELETE, which deletes the entire data set. If the data set is to be used again, it must be redefined with the access method services command DEFINE or by JCL or dynamic allocation.

Instead of using the DELETE - DEFINE sequence, you can specify the REUSE parameter in the DEFINE CLUSTER|ALTERNATEINDEX command. The REUSE parameter lets you treat a filled data set as if it were empty and load it again and again regardless of its previous contents.

A reusable data set can be a key-sequenced data set, an entry-sequenced data set, a linear data set, or a RRDS that resides on one or more volumes. A reusable base cluster cannot have an alternate index, and it cannot be associated with key ranges. When a reusable data set is opened with the reset option, it cannot be shared with other jobs.

VSAM uses a high-used RBA (relative byte address) field to determine if a data set is empty or not. Immediately after a data set is defined, the high-used RBA value is zero. After loading and closing the data set, the high-used RBA is equal to the offset of the last byte in the data set. In a reusable data set, this high-used RBA field can be reset to zero at OPEN by specifying MACRF=RST in the ACB at OPEN. VSAM can use this reusable data set like a newly defined data set.

For compressed format data sets, in addition to the high-used RBA field being reset to zero for MACRF=RST, OPEN resets the compressed and uncompressed data set sizes to zero. The compression dictionary token will not be reset and will be reused to compress the new data. Because the dictionary token was derived from previous data, this could affect the compression ratio depending on the nature of the new data.

Copying and Merging Data Sets

You might want to copy a data set or merge two data sets for a variety of reasons. For example, you might want to create a test copy, you might want two copies to use for two different purposes, or you might want to keep a copy of back records before updating a data set. You can use the access method services REPRO command to copy data sets.

For information about accessing a data set using RLS, see “Chapter 15. Using VSAM Record-Level Sharing” on page 223.

You can use the REPRO command to do any of the following:

- Copy or merge a VSAM data set into another VSAM data set.
- Copy or merge a sequential data set into another sequential data set.
- Copy an alternate index as a key-sequenced VSAM data set.
- Copy a VSAM data set whose records are fixed length into an empty fixed-length RRDS.
- Convert a sequential or indexed sequential data set into a VSAM data set.
- Copy a VSAM data set into a sequential data set.
- Copy a data set (other than a catalog) to reorganize it. Data sets are reorganized automatically.
- Copy individual members of a PDS or PDSE. A PDS or PDSE cannot be copied, but individual members can be copied.

When copying to a key-sequenced data set, the records to be copied must be in ascending order, with no duplicates in the input data set. All the keys must be unique. With an entry-sequenced data set, the records to be copied can be in any order.

Because data is copied as single logical records in either key order or physical order, automatic reorganization can take place as follows:

- Physical relocation of logical records
- Alteration of a record’s physical position within the data set
- Redistribution of free space throughout the data set
- Reconstruction of the VSAM indexes

If you are copying to or from a sequential data set that is not cataloged, you must include the appropriate volume and unit parameters on your DD statements. For more information about these parameters see “Using REPRO to Load a Data Set” on page 115.

Table 8 describes how the data from the input data set is added to the output data set when the output data set is an empty or nonempty entry-sequenced, sequential, key-sequenced, or linear data set, or fixed-length or variable-length RRDS.

Table 8. Adding Data to Various Types of Output Data Sets

Type of Data Set	Empty	Nonempty
Entry sequenced	Loads new data set in sequential order.	Adds records in sequential order to the end of the data set.
Sequential	Loads new data set in sequential order.	Adds records in sequential order to the end of the data set.
Key sequenced	Loads new data set in key sequence and builds an index.	Merges records by key and updates the index. Unless the REPLACE option is specified, records whose key duplicates a key in the output data set are lost.
Linear	Loads new linear data set in relative byte order.	Adds data to control intervals in sequential order to the end of the data set.
Fixed-length RRDS	Loads a new data set in relative record sequence, beginning with relative record number 1.	Records from another fixed-length or variable-length RRDS are merged, keeping their old record numbers. Unless the REPLACE option is specified, a new record whose number duplicates an existing record number is lost. Records from any other type of organization cannot be copied into a nonempty fixed-length RRDS.
Variable-length RRDS	Loads a new data set in relative record sequence, beginning with relative record number 1.	Records from another fixed-length or variable-length RRDS are merged, keeping their old record numbers. Unless the REPLACE option is specified, a new record whose number duplicates an existing record number is lost. Records from any other type of organization cannot be copied into a nonempty fixed-length RRDS.

The REPRO operation is terminated if:

- One physical I/O error is found while writing to the output data set
- A total of four errors is found in any combination of the following:
  - Logical error while writing to the output data set
  - Logical error while reading the input data set
  - Physical error while reading the input data set.

Defining Alternate Indexes

An alternate index is a key-sequenced data set containing index entries organized by the alternate keys of its associated base data records. It provides another way of locating records in the data component of a cluster.

An alternate index can be defined over a key-sequenced or entry-sequenced cluster. An alternate index cannot be defined for a reusable cluster, a fixed- or variable-length RRDS, a catalog, another alternate index, a linear data set, or a non-VSAM data set. The data class parameter can be specified for a system-managed alternate index. Access method services DEFINE will assign the same management class and storage class as the alternate index's base cluster. If a base cluster is defined as extended format, then the alternate index it relates to must be able to be defined as extended format. Alternate indexes cannot be compressed. See "Access to Records through Alternate Indexes" on page 100 for information about the structure of an alternate index.

The sequence for building an alternate index is as follows:

1. Define the base cluster, using either the ALLOCATE command, the DEFINE CLUSTER command, or JCL.
2. Load the base cluster either by using the REPRO command or by writing your own program to load the data set.

3. Define the alternate index, using the DEFINE ALTERNATEINDEX command.
4. Relate the alternate index to the base cluster, using the DEFINE PATH command. The base cluster and alternate index are described by entries in the same catalog.
5. Build the alternate index, using the BLDINDEX command.

VSAM uses three catalog entries to describe an alternate index:

- An alternate index entry describes the alternate index as a key-sequenced cluster.
- A data entry describes the alternate index's data component.
- An index entry describes the alternate index's index component.

Except for data class, attributes of the alternate index's data and index components can be specified separately from the attributes of the whole alternate index. If attributes are specified for the whole alternate index and not for the data and index components, these attributes (except for password and USVR security attributes) apply to the components as well. If the attributes are specified for the components, they override any attributes specified for the entire alternate index.

Naming an Alternate Index

You specify an entry name for an alternate index when you define it. You can specify the entry name as the *dsname* in a JCL DD statement. For details on how VSAM can generate component names for you, see "Naming a Cluster" on page 106.

Specifying Alternate Index Information

When you define an alternate index, you specify descriptive information and performance, security, and data integrity options. The information can apply to the alternate index's data component, its index component, or the whole alternate index. Information for the entire alternate index is specified with the ALTERNATEINDEX parameter and its subparameters. Information for the data component or the index component is specified with the parameter DATA or INDEX.

Passwords are not supported for system-managed alternate indexes. To define an alternate index, you must have RACF alter authority for the base cluster.

Specifying Descriptive Information for an Alternate Index

You need to specify the following descriptive information for an alternate index:

- The name and password of the base cluster related to the alternate index, as specified in the RELATE parameter. The RELATE entry name must be selected so that the multilevel alias facility selects the correct catalog. See *z/OS DFSMS: Managing Catalogs* for information about the multilevel alias facility and *z/OS DFSMS Access Method Services* for information about the order of catalog search.
- Amount of space to allocate for the alternate index, as specified in the CYLINDERS|KILOBYTES|MEGABYTES|RECORDS|TRACKS parameter.
- Volume serial numbers of the volumes on which space is allocated for the alternate index, as designated in the VOLUMES parameter. If you specify the VOLUMES parameter for system-managed data sets, however, the volumes designated might or might not be used, and sometimes can result in a failure. You can indicate nonspecific volumes for a system-managed data set by designating an asterisk (*) for each volume serial. SMS then determines the

volume serial. The default is one volume. Note that if both specific and nonspecific volumes are designated, the specified volume serials must be named first.

- The minimum amount of I/O buffer space that OPEN must provide when the program processes the alternate index's data, as designated in the BUFFERSPACE parameter.
- Name and password of the catalog containing the alternate index's entries, as designated in the CATALOG parameter. This must be the same catalog that contains the base cluster's entries.
- Data class, for alternate indexes, to take advantage of the attributes assigned by the storage administrator.
- Length and position of the alternate key field in data records of the base cluster, as specified in the KEYS parameter.
- Average and maximum lengths of alternate index records, as specified in the RECORDSIZE parameter.
- Whether the alternate index is reusable, as specified in the REUSE parameter.
- Whether the data set is extended format and whether it has extended addressability. These characteristics for the alternate index are the same as those for the cluster.

The performance options and the security and integrity information for the alternate index are the same as that for the cluster. See "Using Access Method Services Parameters" on page 109.

Specifying RECORDSIZE for an Alternate Index with Nonunique Keys

When you define an alternate index with many nonunique keys, specify a RECORDSIZE value that is large enough to handle all the nonunique keys. All occurrences of primary keys for a particular alternate key must be within a single alternate index logical record. If the maximum RECORDSIZE value is 1000, for example, you would not be able to support as many nonunique keys as you would if the maximum RECORDSIZE value were 5000.

Building an Alternate Index

When an alternate index is built by BLDINDEX processing, the alternate index's volume and the base cluster's volume must be mounted. Any volumes identified with the WORKFILES parameter must also be mounted. If one of the data sets identified by the WORKFILES *ddname* is system managed, the other data set must be either a system-managed data set or a non-system-managed data set cataloged in the catalog determined by the catalog search order. See *z/OS DFSMS Access Method Services* for information about the catalog search order. The base cluster cannot be empty (that is, its high-used RBA value cannot be zero). Each record's alternate key value must be unique, unless the alternate index was defined with the NONUNIQUEKEY attribute.

Access method services opens the base cluster to read the data records sequentially, sorts the information obtained from the data records, and builds the alternate index data records.

The base cluster's data records are read and information is extracted to form the key-pointer pair:

- When the base cluster is entry sequenced, the alternate-key value and the data record's RBA form the key-pointer pair.
- When the base cluster is key sequenced, the alternate-key value and the primary-key value of the data set record form the key-pointer pair.

The key-pointer pairs are sorted in ascending alternate-key order. See *z/OS DFSMS Access Method Services* for information about calculating the amount of virtual storage required to sort records, and using the BLDINDEX internal sort (see the BLDINDEX command).

After the key-pointer pairs are sorted into ascending alternate key order, access method services builds alternate index records for key-pointer pairs. When all alternate index records are built and loaded into the alternate index, the alternate index and its base cluster are closed. You cannot build an alternate index over an extended addressable entry-sequenced data set.

Maintaining Alternate Indexes

VSAM assumes alternate indexes are always synchronized with the base cluster and does not check synchronization during open processing. Therefore, all structural changes made to a base cluster must be reflected in its alternate index or indexes. This is called index upgrade.

You can maintain your own alternate indexes or have VSAM maintain them. When the alternate index is defined with the UPGRADE attribute of the DEFINE command, VSAM updates the alternate index whenever there is a change to the associated base cluster. VSAM opens all upgrade alternate indexes for a base cluster whenever the base cluster is opened for output. If you are using control interval processing, you cannot use UPGRADE. See "Chapter 12. Processing Control Intervals" on page 183.

You can define a maximum of 125 alternate indexes in a base cluster with the UPGRADE attribute.

All the alternate indexes of a given base cluster that have the UPGRADE attribute belong to the upgrade set. The upgrade set is updated whenever a base data record is inserted, erased, or updated. The upgrading is part of a request and VSAM completes it before returning control to your program. If upgrade processing is interrupted because of a machine or program error so that a record is missing from the base cluster but its pointer still exists in the alternate index, record management will synchronize the alternate index with the base cluster by letting you reinsert the missing base record. However, if the pointer is missing from the alternate index, that is, the alternate index does not reflect all the base cluster data records, you must rebuild your alternate index to resolve this discrepancy.

Note that when you use SHAREOPTIONS 2, 3, and 4, you must continue to ensure read/write integrity when issuing concurrent requests (such as GETs and PUTs) on the base cluster and its associated alternate indexes. Failure to ensure read/write integrity might temporarily cause "No Record Found" or "No Associated Base Record" errors for a GET request. You can bypass such errors by reissuing the GET request, but it is best to prevent the errors by ensuring read/write integrity.

If you specify NOUPGRADE in the DEFINE command when the alternate index is defined, insertions, deletions, and changes made to the base cluster will not be reflected in the associated alternate index.

When a path is opened for update, the base cluster and all the alternate indexes in the upgrade set are allocated. If updating the alternate indexes is unnecessary, you can specify NOUPDATE in the DEFINE PATH command and only the base cluster is allocated. In that case, VSAM does not automatically upgrade the alternate index. If two paths are opened with MACRF=DSN specified in the ACB macro, the NOUPDATE specification of one can be nullified if the other path is opened with UPDATE specified.

How Reorganization Affects Alternate Indexes

If you reorganize a base cluster, you do not need to rebuild the alternate indexes, because the relationships between the base cluster and the alternate indexes have not changed. If you unload a VSAM data set, delete the existing cluster on DASD, redefine the CLUSTER on DASD, then load the new data set from the unloaded copy, then you do not need to rebuild the alternate index.

Alternate Index Backups

You can use DFSMShsm to back up a base cluster and its associate alternate indexes. For more information see *z/OS DFSMShsm Managing Your Own Data*.

Defining a Path

After an alternate index is defined, you need to establish the relationship between an alternate index and its base cluster, using the access method services command, DEFINE PATH. You must name the path and can also give it a password. The path name refers to the base cluster/alternate index pair. When you access the data set through the path, you must specify the path name and the DSNAME parameter in the JCL.

When your program opens a path for processing, both the alternate index and its base cluster are opened. When data in a key-sequenced base cluster is read or written using the path's alternate index, keyed processing is used. RBA processing is permitted only for reading or writing an entry-sequenced data set's base cluster.

See *z/OS DFSMS Access Method Services* for information about using the DEFINE PATH command.

Defining a Page Space

A page space is a system data set that contains pages of virtual storage. The pages are stored into and retrieved from the page space by the auxiliary storage manager. A page space is an entry-sequenced cluster that is entirely preformatted (unlike other data sets), and is contained on a single volume. You cannot open a page space as a user data set.

The considerations for defining a page space are much like those for defining a cluster. The DEFINE PAGESPACE command has many of the same parameters as the DEFINE CLUSTER command, so the information you must supply for a page space is similar to what you would specify for a cluster. A page space data set cannot be in extended format. For details on specifying information for a data set, especially for system-managed data sets, see "Specifying Cluster Information" on page 108 and "Using Access Method Services Parameters" on page 109.

A page space has a maximum usable size equal to 16 MB paging slots (records). See *z/OS DFSMS Access Method Services* for information about the description for space size declarations.

You can define a page space in a user catalog, then move the catalog to a new system, and establish it as the system's master catalog. For page spaces to be system managed, they must be cataloged, and you must let the system determine which catalog to use. Page spaces also cannot be duplicate data sets. See "Duplicate Data Set Names" on page 107 for information about how VSAM handles duplicate data sets. The system cannot use a page space if its entry is in a user catalog.

JOBCAT and STEPCAT DD statements are not supported for system-managed data sets. The JOBCAT or STEPCAT DD statement will fail if it references a system-managed catalog, or if the data set being searched is system managed. Also, all referenced catalogs must be connected to the system master catalog.

When you issue a DEFINE PAGESPACE command, the system creates an entry in the catalog for the page space, then preformats the page space. If an error occurs during the preformatting process (for example, an I/O error or an allocation error), the page space's entry remains in the catalog even though no space for it exists. Issue a DELETE command to remove the page space's catalog entry before you redefine the page space.

Each page space is represented by two entries in the catalog: a cluster entry and a data entry. (A page space is conceptually an entry-sequenced cluster.) Both of these entries should be password protected if the page space is password protected.

The system recognizes a page space if it is defined as a system data set at system initialization time or if it is named in SYS1.PARMLIB. To be used as a page space, it must be defined in a master catalog.

Recommendations:

1. When you define page spaces during system initialization, use the ALTER command to add passwords to each entry because passwords cannot be specified during system initialization. The passwords you specify with the DEFINE PAGESPACE command are put in both the page space's cluster entry and its data entry. Unless you ensure that the catalog containing the page space entry is either password protected or RACF protected, a user can list the catalog's contents and find out each entry's passwords.
2. Passwords are ignored for system-managed data sets. For these, you must have RACF alter authority.

Checking for Problems in Catalogs and Data Sets

VSAM provides you with several means of locating problems in your catalogs and data sets. This section describes procedures for listing catalog entries and printing the contents of data sets.

You can also use the access method services REPRO command to copy a data set to an output device. For more information about REPRO see “Copying and Merging Data Sets” on page 119.

The access method services VERIFY command provides a means of checking and restoring end-of-data-set values after system failure. For more information about VERIFY see “Using VERIFY to Process Improperly Closed Data Sets” on page 53. See *z/OS DFSMS: Managing Catalogs* for information about using the DIAGNOSE command to indicate the presence of invalid data or relationships in the BCS and VVDS.

The access method services EXAMINE command lets the user analyze and report on the structural inconsistencies of key-sequenced data set clusters. The EXAMINE command is described in “Chapter 16. Checking VSAM Key-Sequenced Data Set Clusters for Structural Errors” on page 235.

Listing Catalog Entries

After you define a catalog or data set, use the access method services command LISTCAT to list all or part of a catalog’s entries. LISTCAT shows information about objects defined in the catalog, such as:

- Attributes of the object, including SMS attributes
- Creation and expiration dates
- Protection specification
- Statistics on dynamic usage or data set accessing represented by the entry
- Space allocation
- Volume information
- Structure of the data set

The listing can be customized by limiting the number of entries, and the information about each entry, that is printed.

You can obtain the same list while using the interactive storage management facility (ISMF) by issuing the CATLIST line operator on the Data Set List panel. The list is placed into a data set, which you can view immediately after issuing the request. See *z/OS DFSMS: Using the Interactive Storage Management Facility* for information about the CATLIST line operator.

Printing the Contents of Data Sets

If a problem occurs, you can use the access method services command PRINT to print part or all of the contents of a fixed-length or variable-length RRDS; a key-sequenced, linear, or entry-sequenced VSAM data set; an alternate index; or a catalog. If you use the relative byte address, you can print part of a linear data set. Partial printing is rounded up to 4096 byte boundaries. The components of a key-sequenced data set or an alternate index can be printed individually by specifying the component name as the data set name. An alternate index is printed as though it were a key-sequenced cluster.

Entry-sequenced and linear data sets are printed in physical sequential order. Key-sequenced data sets can be printed in key order or in physical-sequential order. Fixed-length or variable-length RRDSs are printed in relative record number sequence. A base cluster can be printed in alternate key sequence by specifying a path name as the data set name for the cluster.

Only the data content of logical records is printed. System-defined control fields are not printed. Each record printed is identified by one of the following:

- The relative byte address (RBA) for entry-sequenced data sets.
- The key for indexed-sequential and key-sequenced data sets, and for alternate indexes
- The record number for fixed-length or variable-length RRDSs.

See *z/OS MVS Programming: Authorized Assembler Services Guide* for information about program authorization. See “Authorized Program Facility and Access Method Services” on page 64 for information about using the PRINT command to print a catalog.

Restriction: If the system finds four logical and/or physical errors while attempting to read the input, printing ends abnormally.

Deleting Data Sets

Use the access method services DELETE command, described in *z/OS DFSMS Access Method Services*, to delete data sets, catalogs, and objects. DELETE *entry name* removes the data set from the volume on which it resides, and the catalog entry for the data set. You can delete the entire cluster, or just the alternate index, path, or alias, for example. Use DELETE *entry name* VVR FILE (*ddname*) to delete an uncataloged VSAM data set. DELETE *entry name* VVR FILE (*ddname*) removes the VSAM volume record (VVR) from the VSAM volume data set (VVDS), and the data set control block from the volume table of contents (VTOC).

Use the ERASE parameter if you want to erase the components of a cluster or alternate index when deleting it. ERASE overwrites the data set. Use the NOSCRATCH parameter if you do not want the data set entry (DSCB) removed from the VTOC. NOSCRATCH nullifies an ERASE parameter on the same DELETE command.

Chapter 9. Defining and Manipulating VSAM Data Sets: Examples

This chapter covers the following topics.

Topic	Page
Example of Defining a VSAM Data Set	130
Examples of Defining Temporary VSAM Data Sets	132
Examples of Defining Alternate Indexes and Paths	133

The following set of examples contain a wide range of functions available through access method services commands that let you define:

- VSAM data sets
- Temporary VSAM data sets
- Alternate indexes and paths

See *z/OS DFSMS Access Method Services* for examples of the other functions available through access method services.

An existing system catalog is assumed to be security protected at the update-password, control-password, and master-password levels. Because passwords are not supported for system-managed data sets and catalogs, assume that you have RACF authority for the operation being performed in the examples that define or manipulate system-managed data sets and catalogs.

© Copyright IBM Corp. 1987, 2002

Example of Defining a VSAM Data Set

The following example shows a typical sequence of commands to create a catalog, define a data set (that is cataloged in the newly created catalog), load the data set with data, list the data set's catalog entry, and print the data set:

```
//DEFINE   JOB ...
//STEP1    EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *

           DEFINE USERCATALOG (NAME (USERCATX) ICFCATALOG CYLINDERS(15 5) -
                  VOLUMES(VSER05)) DATA (CYLINDERS(3 1))

           IF LASTCC = 0 THEN -
                DEFINE CLUSTER(NAME (EXAMPL1.KSDS) VOLUMES(VSER05)) -
                  DATA              (KILOBYTES (50 5))
/*
//STEP2    EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSABEND DD SYSOUT=*
//AMSDUMP  DD SYSOUT=*
//INDSET4  DD DSNAME=SOURCE.DATA,DISP=OLD,
//       VOL=SER=VSER02,UNIT=3380
//SYSIN    DD *

     REPRO INFILE(INDSET4) OUTDATASET(EXAMPL1.KSDS)

     IF LASTCC = 0 THEN -
         LISTCAT ENTRIES(EXAMPL1.KSDS)

    IF LASTCC = 0 THEN -
         PRINT INDATASET(EXAMPL1.KSDS)
/*
```

The following access method services commands are used in this example:

DEFINE USERCATALOG	Create the catalog
DEFINE CLUSTER	Define the data set
REPRO	Load the data set
LISTCAT	List the catalog entry
PRINT	Print the data set

See "Chapter 19. Using Job Control Language for VSAM" on page 265 for examples of creating VSAM data sets through JCL. See *z/OS DFSMS Access Method Services* for more details and examples of these or other access method services commands.

The first DEFINE command defines a user catalog named USERCATX. The USERCATALOG keyword specifies that a user catalog is to be defined. The command's parameters follow.

NAME	NAME is required and names the catalog being defined.
ICFCATALOG	Specifies the catalog format.
CYLINDERS	Specifies the amount of space to be allocated from the volume's available space. If it is specified for a system-managed catalog, it overrides the DATACLAS space specification. A space parameter is required.
VOLUMES	Specifies the volume to contain the catalog. You should not specify the VOLUMES parameter if the catalog is system managed.
DATA	DATA is required when attributes are to be explicitly specified for the data component of the cluster.
CYLINDERS	Specifies the amount of space allocated for the data component. A space parameter is required.

The second DEFINE command defines a key-sequenced data set named EXAMPL1.KSDS. The command's parameters are:

CLUSTER	The CLUSTER keyword is required, and specifies that a cluster is to be defined. The CLUSTER keyword is followed by the parameters specified for the whole clusters, and, optionally, by the parameters specified separately for the data and index components.
NAME	NAME is required and specifies the cluster being defined.
VOLUMES	Specifies the volumes that a cluster's components are allocated space. You can specify up to 59 volumes per cluster for system-managed clusters.
DATA	DATA is required when parameters are explicitly specified for the data component of the cluster.
KILOBYTES	Specifies the amount of space allocated for the data component.

The REPRO command here loads the VSAM key-sequenced data set named EXAMPL1.KSDS from an existing data set called SOURCE.DATA (that is described by the INDSET4 DD statement). The command's parameters are:

INFILE	Identifies the data set containing the source data. The ddname of the DD statement for this data set must match the name specified on this parameter.
OUTDATASET	Identifies the name of the data set to be loaded. Access method services dynamically allocates the data set. The data set is cataloged in the master catalog.

Because the cluster component is not password protected, a password is not required.

If the REPRO operation is successful, the data set's catalog entry is listed, and the contents of the data set just loaded are printed.

LISTCAT	Lists catalog entries. The ENTRIES parameter identifies the names of the entries to be listed.
PRINT	Prints the contents of a data set. The INDATASET parameter is required and identifies the name of the data set to be printed. Access method services dynamically allocates the data set. The data set is cataloged in the master catalog. No password is required because the cluster component is not password protected.

Examples of Defining Temporary VSAM Data Sets

The following examples uses the ALLOCATE command to define a new temporary VSAM data set. See "Temporary VSAM Data Sets" on page 268 for examples of creating temporary VSAM data sets through JCL.

Example 1: Defining a Temporary VSAM Data Set Using ALLOCATE

```
//ALLOC    JOB  ...
//STEP1    EXEC PGM=IDCAMS
//SYSPRINT DD   SYSOUT=A
//SYSIN    DD   *

           ALLOC -
              DSNAME(&CLUSTER) -
              NEW PASS -
              RECORG(ES) -
              SPACE(1,10) -
              AVGREC(M) -
              LRECL(256) -
              STORCLAS(TEMP)
/*
```

The command's parameters are:

DSNAME	Specifies the data set name. If you specify a data set name for a system-managed temporary data set, it must begin with & or &&. The DSNAME parameter is optional for temporary data sets only. If you do not specify a DSNAME, the system generates a qualified data set name for the temporary data set.
NEW	Specifies that a new data set is created in this job step.
RECORG	Specifies a VSAM entry-sequenced data set.
SPACE	Specifies an average record length of 1 and a primary quantity of 10.
AVGREC	Specifies that the primary quantity specified on the SPACE keyword represent the number of records in megabytes (multiplier of 1,048,576).
LRECL	Specifies a record length of 256 bytes.
STORCLAS	Specifies a storage class for the temporary data set. The STORCLAS keyword is optional. If you do not specify STORCLAS for the new data set and your storage administrator has provided an ACS routine, the ACS routine can select a storage class.

Example 2: Creating a Temporary Data Set with Default Parameter Values

The following example shows the minimum number of parameters required to create a temporary non-VSAM sequential data set. You must specify the RECORG parameter to create a temporary VSAM data set.

```
//ALLOC    JOB  ...
//STEP1    EXEC PGM=IDCAMS
//SYSPRINT DD   SYSOUT=A
//SYSIN    DD   *
           ALLOC -
              NEW PASS
```

If no DSNAME name is specified, the system generates one. If no STORCLAS name is specified, and your storage administrator has provided an ACS routine, the ACS routine can select a storage class.

Examples of Defining Alternate Indexes and Paths

In this section, the access method services DEFINE ALTERNATEINDEX and DEFINE PATH commands are used to define alternate indexes and a path.

The following example defines an alternate index on a previously loaded VSAM key-sequenced base cluster, defines a path in the alternate index to provide a means for processing the base cluster through the alternate index, and builds the alternate index. The alternate index, path, and base cluster must all be defined in the same catalog, in this case, the master catalog. Because the master catalog is recoverable, access method services dynamically allocate the volume that contains the base cluster's index component to access its recovery area.

JCL Statements

The IDCUT1 and IDCUT2 DD statements describe the DSNAMES and a volume containing data space made available to BLDINDEX for defining and using two sort work data sets in the event an external sort is performed. The data space is not used by BLDINDEX if enough virtual storage is available to perform an internal sort.

Commands

The first DEFINE command defines a VSAM alternate index over the base cluster EXAMPL1.KSDS.

NAME	NAME is required and names the object being defined.
RELATE	RELATE is required and specifies the name of the base cluster on which the alternate index is defined.
MASTERPW and UPDATEPW	Specifies the master and update passwords, respectively, for the alternate index.
KEYS	Specifies the length of the alternate key and its offset in the base-cluster record.
RECORDSIZE	Specifies the length of the alternate-index record. The average and maximum record lengths are 40 and 50 bytes, respectively. Because the alternate index is being defined with the NONUNIQUEKEY attribute, the index must be large enough to contain the primary keys for all occurrences of any one alternate key.
VOLUMES	VOLUMES is required only for non-system-managed data sets and specifies the volume that contains the alternate index (EXAMPL1.AIX).
CYLINDERS	Specifies the amount of space allocated to the alternate index. A space parameter is required.
NONUNIQUEKEY	Specifies that the base cluster can contain multiple occurrences of any one alternate key.
UPGRADE	Specifies that the alternate index is to reflect all changes made to the base-cluster records, such as additions or deletions of records.
CATALOG	Because the master catalog is password protected, the CATALOG parameter is required. It specifies the name of the master catalog and its update or master password, which is required for defining in a protected catalog.

The second DEFINE command defines a path over the alternate index. After the alternate index is built, opening with the path name causes processing of the base cluster through the alternate index.

NAME	The NAME parameter is required and names the object being defined.
PATHENTRY	The PATHENTRY parameter is required and specifies the name of the alternate index over which the path is defined and its master password.
READPW	Specifies a read password for the path; it is propagated to the master-password level.
CATALOG	The CATALOG parameter is required, because the master catalog is password protected. It specifies the name of the master catalog and its update or master password that is required for defining in a protected catalog.

The BLDINDEX command builds an alternate index. Assume that enough virtual storage is available to perform an internal sort. However, DD statements with the default ddnames of IDCUT1 and IDCUT2 are provided for two external sort work data sets if the assumption is incorrect and an external sort must be performed.

INDATASET	The INDATASET parameter identifies the base cluster. Access method services dynamically allocates the base cluster. The base cluster's cluster entry is not password protected even though its data and index components are.
OUTDATASET	The OUTDATASET parameter identifies the alternate index. Access method services dynamically allocates the alternate index. The update- or higher-level password of the alternate index is required.
CATALOG	The CATALOG parameter specifies the name of the master catalog. If it is necessary for BLDINDEX to use external sort work data sets, they will be defined in and deleted from the master catalog. The master password permits these actions.

The PRINT command causes the base cluster to be printed using the alternate key, using the path defined to create this relationship. The INDATASET parameter identifies the path object. Access method services dynamically allocates the path. The read password of the path is required.

Chapter 10. Processing VSAM Data Sets

This chapter covers the following topics.

Topic	Page
Creating an Access Method Control Block	138
Creating an Exit List	138
Opening a Data Set	139
Creating a Request Parameter List	140
Manipulating the Contents of Control Blocks	142
Requesting Access to a Data Set	143
Closing Data Sets	154
Operating in SRB or Cross-Memory Mode	155
Using VSAM Macros in Programs	156

To process VSAM data sets, you use VSAM macros. You can use the following procedure for processing a VSAM data set to read, update, add, or delete data:

1. Create an access method control block to identify the data set to be opened using the ACB or GENCB macro.
2. Create an exit list to specify the optional exit routines that you supply, using the EXLST or GENCB macro.
3. Optionally, create a resource pool, using the BLDVRP macro. (See “Chapter 14. Sharing Resources Among VSAM Data Sets” on page 211.)
4. Connect your program to the data set you want to process, using the OPEN macro.
5. Create a request parameter list to define your request for access, using the RPL or GENCB macro.
6. Manipulate the control block contents using the GENCB, TESTCB, MODCB and SHOWCB macros.
7. Request access to the data set, using one or more of the VSAM request macros (GET, PUT, POINT, ERASE, CHECK, and ENDREQ).
8. Disconnect your program from the data set, using the CLOSE macro.

The virtual resource pool for all components of the clusters or alternate indexes must be successfully built before any open is issued to use the resource pool; otherwise, the results might be unpredictable or performance problems might occur.

For information about the syntax of each macro, and for coded examples of the macros, see *z/OS DFSMS Macro Instructions for Data Sets*.

The ACB, RPL, and EXLST are created by the caller of VSAM. When storage is obtained for these blocks, virtual storage management assigns the PSW key of the requestor to the subpool storage. An authorized task can change its PSW key. Since VSAM record management runs in the protect key of its caller, such a change might make previously acquired control blocks unusable because the storage key of the subpool containing these control blocks no longer matches the VSAM caller’s key.

© Copyright IBM Corp. 1987, 2002

Creating an Access Method Control Block

Before opening a data set for processing, you must create an access method control block (ACB) that:

- Identifies the data set to be opened
- Specifies the type of processing
- Specifies the basic options
- Indicates if a user exit routine is to be used while the data set is being processed

Include the following information in your ACB for OPEN to prepare the kind of processing your program requires:

- The address of an exit list for your exit routines. Use the EXLST macro to construct the list.
- If you are processing concurrent requests, the number of requests (STRNO) defined for processing the data set. For more information about concurrent requests see “Making Concurrent Requests” on page 151.
- The size of the I/O buffer virtual storage space and/or the number of I/O buffers that you are supplying for VSAM to process data and index records.
- The password required for the type of processing desired. Passwords are not supported for system-managed data sets. You must have RACF authorization for the type of operation to be performed.
- The processing options that you plan to use:
  - Keyed, addressed, or control interval, or a combination
  - Sequential, direct, or skip sequential access, or a combination
  - Retrieval, storage, or update (including deletion), or a combination
  - Shared or nonshared resources.
- The address and length of an area for error messages from VSAM.
- If using RLS, see “Chapter 15. Using VSAM Record-Level Sharing” on page 223.

You can use the ACB macro to build an access method control block when the program is assembled, or the GENCB macro to build a control block when the program is executed. See “Manipulating the Contents of Control Blocks” on page 142 for information about the advantages and disadvantages of using GENCB.

Creating an Exit List

To access exit routines during data set processing, you must specify the addresses of your exit routines using the EXLST macro. Any number of ACB macros in a program can indicate the same exit list for the same exit routines to do all the special processing for them, or they can indicate different exit lists. Use exit routines for the following tasks:

- **Analyzing physical errors.** When VSAM finds an error in an I/O operation that the operating system’s error routine cannot correct, the error routine formats a message for your physical error analysis routine (the SYNAD user exit) to act on.
- **Analyzing logical errors.** Errors not directly associated with an I/O operation, such as an invalid request, cause VSAM to exit to your logical error analysis routine (the LERAD user exit).
- **End-of-data-set processing.** When your program requests a record beyond the last record in the data set, your end-of-data-set routine (the EODAD user exit) is given control. The end of the data set is beyond either the highest addressed or the highest keyed record, if your program is using addressed or keyed access.

- **Journalizing transactions.** To journalize the transactions against a data set, you might specify a journal routine (the JRNAD user exit). To process a key-sequenced data set using addressed access, you need to know if any RBAs changed during keyed processing. When you are processing by key, VSAM exits to your routine for noting RBA changes before writing a control interval in which there is an RBA change. When journalizing transactions for compressed data sets, the RBAs and data lengths represent compressed data. VSAM does not exit to the JRNAD routine for RBA change if the data set is extended addressable.
- **User processing.** User processing exits (UPAD) are available to assist subsystems that need to dispatch new units of work. The UPAD wait exit is given control before VSAM issues any WAIT SVCs. Use the UPAD post exit to make it easier to use cross-memory processing. See Table 21 on page 259.

The EXLST macro is coordinated with the EXLST parameter of an ACB or GENCB macro used to generate an ACB. To use the exit list, you must code the EXLST parameter in the ACB.

You can use the EXLST macro to build an exit list when the program is assembled, or the GENCB macro to build an exit list when the program is executed. For information about the advantages and disadvantages of using GENCB see “Manipulating the Contents of Control Blocks” on page 142.

Opening a Data Set

Before accessing a data set, your program must issue the OPEN macro to open the data set for processing. Opening a data set causes VSAM to take the following actions:

- Verify that the data set matches the description specified in the ACB or GENCB macro (for example, MACRF=KEY implies that the data set is a key-sequenced data set).
- Construct the internal control blocks that VSAM needs to process your requests for access to the data set.

  To determine which processing options to use, VSAM merges information from the data definition (DD) statement and catalog definition of the data set with information in the access method control block and exit list. The order of precedence follows:

  1. The DD statement AMP parameters
  2. The ACB, EXLST, or GENCB parameters
  3. The catalog entry for the data set

  For example, if both an ACB or GENCB macro and the DD statement have values for buffer space, the values in the DD statement override those in the macro. The catalog entry is the minimum buffer space when it is not specified in the DD statement or macro or when it is less than the amount specified in the data set definition.
- Check for consistency of updates to the prime index and data components if you are opening a key-sequenced data set, an alternate index, or a path. If separate updates occur to data set and its index, VSAM issues a warning message to indicate a time stamp discrepancy.
- Check the password your program specified in the ACB PASSWD parameter against the appropriate password (if any) in the catalog definition of the data. The system does not support passwords for system-managed data sets. You must

have RACF authorization for the operation. The password requirement depends on the kind of access that is specified in the access method control block:

- Full access lets you perform all operations (retrieve, update, insert, and delete) on a data set on any associated index or catalog record. The master password lets you delete or alter the catalog entry for the data set or catalog it protects.
- Control-interval update access requires the control password or RACF control authority. The control lets you use control-interval access to retrieve, update, insert, or delete records in the data set it protects. For information about the use of control-interval access, see “Chapter 12. Processing Control Intervals” on page 183.

  Control-interval read access requires only the read password or RACF read authority, that lets you examine control intervals in the data set it protects. The read password or RACF read authority does not let you add, change, or delete records.
- Update access requires the update password, which lets you retrieve, update, insert, or delete records in the data set it protects.
- Read access requires the read password, that lets you examine records in the data set it protects. The read password does not permit you to add, change, or delete records.

A password of one level authorizes you to do everything that a password of a lower level authorizes.

Besides passwords, you can have protection provided by RACF. When RACF protection and password protection are both applied to a data set, password protection is bypassed, and use is authorized solely through the RACF checking system. RACF checking is bypassed for a caller that is in supervisor state or Key 0. For more information on password and RACF protection, see “Chapter 5. Protecting Data Sets” on page 55.

- An error during OPEN can cause a component that is open for update processing to close improperly, leaving on the open-for-output indicator. When VSAM detects an open-for-output indicator, it issues an implicit VERIFY command and a message that indicates whether the VERIFY command was successful.

  If a subsequent OPEN is issued for update, VSAM turns off the open-for-output indicator at CLOSE. If the data set was open for input, however, VSAM leaves on the open-for-output indicator.

Creating a Request Parameter List

After you have connected your program to a data set, you can issue requests for access. A request parameter list defines a request. This list identifies the data set to which the request is directed by naming the ACB macro that defines the data set. Each request macro (GET, PUT, ERASE, POINT, CHECK, and ENDREQ) gives the address of the request parameter list that defines the request.

You can use the RPL macro to generate a request parameter list (RPL) when your program is assembled, or the GENCB macro to build a request parameter list when your program is executed. For information about the advantages and disadvantages of using GENCB, see “Manipulating the Contents of Control Blocks” on page 142.

When you define your request, specify only the processing options appropriate for that particular request. Parameters not required for a request are ignored. For

example, if you switch from direct to sequential retrieval with a request parameter list, you do not have to zero out the address of the field containing the search argument (ARG=address).

The following information defines your request:

- Access by address (RBA), key, or relative record number. Address access can be sequential or direct. Key or relative record number access can be sequential, skip sequential, or direct. Access can be forward (next sequential record) or backward (previous sequential record). Access can be for updating or not updating. A nonupdate direct request to retrieve a record causes VSAM to position to the following record for subsequent sequential access. For more information about VSAM positioning, see “POINT Macro for Positioning” on page 147.
- RPLs (including RPLs defined by a chain), either synchronous, so that VSAM does not give control back to your program until the request completes, or asynchronous, so that your program can continue to process or issue other requests while the request is active. With asynchronous requests, your program must use the CHECK macro to suspend its processing until the request completes. For more information about synchronous and asynchronous processing, see “Making Asynchronous Requests” on page 153.
- For a keyed request, either a generic key (a leading portion of the key field), or a full key to which the key field of the record is to be compared.
- For retrieval, either a data record to be placed in a work area in your program or the address of the record within VSAM’s buffer to be passed to your program. For requests that involve updating or inserting, the work area in your program contains the data record.
- For a request to directly access a control interval, specify the RBA of the control interval. With control interval access, you are responsible for maintaining the control information in the control interval. If VSAM’s buffers are used, VSAM permits control interval and stored record operations simultaneously. If your program provides its own buffers, only control interval processing is permitted. For information about control interval access, see “Chapter 12. Processing Control Intervals” on page 183.

You can chain request parameter lists together to define a series of actions for a single GET or PUT. For example, each parameter list in the chain could contain a unique search argument and point to a unique work area. A single GET macro would retrieve a record for each request parameter list in the chain. All RPLs in a chain must refer to the same ACB.

A chain of request parameter lists is processed serially as a single request. (Chaining request parameter lists is not the same as processing concurrent requests in parallel. Processing in parallel requires that VSAM keep track of many positions in a data set.)

Each request parameter list in a chain should have the same OPTCD subparameters. Having different subparameters can cause logical errors. You cannot chain request parameter lists for updating or deleting records—only for retrieving records or storing new records. You cannot process records in the I/O buffer with chained request parameter lists. (RPL OPTCD=UPD and RPL OPTCD=LOC are invalid for a chained request parameter list.)

With chained request parameter lists, a POINT, a sequential or skip-sequential GET, or a direct GET with positioning requested (RPL OPTCD=NSP) causes VSAM to position itself at the record following the record identified by the last request parameter list in the chain.

When you are using chained RPLs, if an error occurs anywhere in the chain, the RPLs following the one in error are made available without being processed and are posted complete with a feedback code of zero.

Manipulating the Contents of Control Blocks

VSAM provides a set of macros, GENCB, TESTCB, MODCB, and SHOWCB, to let you manipulate the contents of control blocks at execution time. Use these macros to generate, test, modify, and display the contents of fields in the access method control block, the exit list, and the request parameter list. You do not have to know the format of the control block when you use these macros.

The GENCB, MODCB, TESTCB, and SHOWCB macros build a parameter list that describes, in codes, the actions indicated by the parameters you specify. The parameter list is passed to VSAM to take the indicated actions. An error can occur if you specify the parameters incorrectly.

If you issue a MODCB, SHOWCB, or TESTCB for a non-VSAM ACB, unpredictable results occur.

Generating a Control Block

The GENCB macro can be used to generate an access method control block, an exit list, or a request parameter list when your program is executed. Generating the control block at execution time with GENCB has the advantage of requiring no reassembly of the program when you adopt a new version of VSAM in which control block formats might have changed. If you use the ACB, EXLST, and RPL macros to build control blocks, and then adopt a subsequent release of VSAM in which the control block format has changed, you have to reassemble your program. GENCB also gives you the ability to generate multiple copies of the ACB, EXLST, or RPL to be used for concurrent requests. The disadvantage of using GENCB is that the path length is longer. It takes more instructions to build a control block using GENCB than to code the control block directly.

You can use the WAREA parameter to provide an area of storage in which to generate the control block. This work area has a 64K (X'FFFF') size limit. If you do not provide storage when you generate control blocks, the ACB, RPL, and EXLST reside below 16 MB unless LOC=ANY is specified.

Testing the Contents of ACB, EXLST, and RPL Fields

With the TESTCB macro, VSAM compares the contents of a field you specify with a value that you specify. To show the result of this comparison, VSAM sets the condition code in the PSW (program status word). Only one keyword can be specified each time TESTCB is issued. Use TESTCB to find out:

- If an action has been done by VSAM or your program (for example, opening a data set or activating an exit).
- What kind of a data set is being processed to alter your program logic as a result of the test.

After issuing a TESTCB macro, examine the PSW condition code. If the TESTCB is not successful, register 15 contains an error code and VSAM passes control to an error routine, if one has been specified. For a keyword specified as an option or a name, you test for an equal or unequal comparison; for a keyword specified as an address or a number, you test for an equal, unequal, high, low, not-high, or not-low condition.

VSAM compares A to B, where A is the contents of the field and B is the value to compare. A low condition means, for example, A is lower than B — that is, the value in the control block is lower than the value you specified. If you specify a list of option codes for a keyword (for example, MACRF=(ADR,DIR)), each of them must equal the corresponding value in the control block for you to get an equal condition.

Some of the fields can be tested at any time; others, only after a data set is opened. The ones that can be tested only after a data set is opened can, for a key-sequenced data set, pertain either to the data or to the index, as specified in the OBJECT parameter.

You can display fields using the SHOWCB macro at the same time you test the fields.

Modifying the Contents of an ACB, EXLST, or RPL

The MODCB macro lets you customize the control blocks generated with the GENCB macro. The MODCB macro can be used to modify the contents of an access method control block, an exit list, or a request parameter list. Typical reasons to modify a request parameter list are to change the length of a record (RECLEN) when you are processing a data set whose records are not all the same length, and to change the type of request (OPTCD), such as from direct to sequential access or from full-key search argument to generic key search argument.

Displaying the Contents of ACB, EXLST, and RPL Fields

The SHOWCB macro causes VSAM to move the contents of various fields in an access method control block, an exit list, or a request parameter list into your work area. You might want to learn the reason for an error or to collect statistics about a data set to permit your program to print a message or keep records of transactions.

Requesting Access to a Data Set

After your program is opened and a request parameter list is built, use the action request macros GET, PUT, ERASE, POINT, CHECK, and ENDREQ. Each request macro uses a request parameter list that defines the action to be taken. For example, when a GET macro points to a request parameter list that specifies synchronous, sequential retrieval, the next record in sequence is retrieved. When an ENDREQ macro points to a request parameter list, any current request (for example, a PUT) for that request parameter list finishes, and the resources held by the request parameter list are released.

The action request macros lets you do the following tasks:
- Insert new records
- Retrieve existing records
- Point to existing records
- Update existing records
- Delete existing records
- Write buffers
- Retain buffers

- Perform multistring processing
- Perform concurrent requests
- Access records using a path
- Check for completion of asynchronous requests
- End request processing

Inserting and Adding Records

Record insertions in VSAM data sets occur in several ways:

- **PUT RPL OPTCD=DIR,NSP**—Inserting records directly. VSAM remembers its position for subsequent sequential access.
- **PUT RPL OPTCD=DIR,NUP**—Inserting a record directly. VSAM does not remember its position.
- **PUT RPL OPTCD=SEQ,NUP or NSP**—Inserting records sequentially. VSAM remembers its position for subsequent sequential access.
- **PUT RPL OPTCD=SKP,NUP or NSP**—Inserting records in skip sequential order. VSAM remembers its position for subsequent sequential access.

Insertions into an Entry-Sequenced Data Set

VSAM does not insert new records into an entry-sequenced data set. All records are added at the end of the data set.

Insertions into a Key-Sequenced Data Set

Insertions into a key-sequenced data set use the free space provided during the definition of the data set or the free space that develops because of control interval and control area splits. To create a data set or make mass insertions, use RPL OPTCD=SEQ,NUP or NSP. RPL OPTCD=SEQ,NUP or NSP inserts the records sequentially and maintains free space during load mode and during mass insertions. All the other types use the direct insert strategy. If MACRF=SIS is specified in the ACB, all inserts use sequential insert strategy.

With addressed access of a key-sequenced data set, VSAM does not insert or add new records.

Sequential Insertion. If the new record belongs after the last record of the control interval and the free space limit has not been reached, the new record is inserted into the existing control interval. If the control interval does not contain sufficient free space, the new record is inserted into a new control interval without a true split.

If the new record does not belong at the end of the control interval and there is free space in the control interval, it is placed in sequence into the existing control interval. If adequate free space does not exist in the control interval, a control interval split occurs at the point of insertion. The new record is inserted into the original control interval and the following records are inserted into a new control interval.

Mass Sequential Insertion. When VSAM detects two or more records to be inserted in sequence into a collating position (between two records) in a data set, VSAM uses a technique called mass sequential insertion to buffer the records being inserted, and to reduce I/O operations. Using sequential instead of direct access takes advantage of this technique. Also extend your data set (resume loading) by using sequential insertion to add records beyond the highest key or relative record number. There are possible restrictions to extending a data set into a new control area depending on the specified share options. See “Chapter 13. Sharing VSAM Data Sets” on page 195.

Mass sequential insertion observes control interval and control area free space specifications when the new records are a logical extension of the control interval or control area (that is, when the new records are added beyond the highest key or relative record number used in the control interval or control area).

When several groups of records in sequence are to be mass inserted, each group can be preceded by a POINT with RPL OPTCD=KGE to establish positioning. KGE specifies that the key you provide for a search argument must be equal to the key or relative record number of a record.

Direct Insertion—CI Split. If the control interval has enough available space, the record is inserted. If the control interval does not have enough space to hold the record, the entire CI is split, unless the record is the last key in the file. The last record is always placed in a new, empty CI and does not show up as a CI split.

Direct Insertion—CA Split. If no additional CI is available to allow a CI split, the CA is split. For the last record in the file, however, the new record is inserted as the first record in a new, empty CA. This does not show up as a CA split. If the new record belongs after the last record of the control interval and there is still space, the new record is added to the end of the existing control interval. If the control interval does not contain sufficient free space, the new record is inserted into an unused control interval.

Insertions into a Fixed-Length Relative-Record Data Set

You can insert records into a fixed-length RRDS either sequentially or directly.

Sequential Insertion. Insertions into a fixed-length RRDS go into empty slots. When a record is inserted sequentially into a fixed-length RRDS it is assigned the next relative record number in sequence. If the slot is not empty, VSAM sets an error return code, indicating a duplicate record. The assigned number is returned in the argument field of the RPL.

Direct Insertion. Direct or skip-sequential insertion of a record into a fixed-length RRDS places the record as specified by the relative record number in the argument field of the RPL. You must insert the record into a slot that does not contain a record. If the slot specified does contain a record, VSAM sets an error return code in the RPL and rejects the request.

If the insertion is to the end of the control interval, the record is placed in a new control interval.

Insertions into a Variable-Length Relative-Record Data Set

A variable-length RRDS is processed in the same way as a fixed-length RRDS, with the following exceptions:

- You must specify the record length in the RECLEN field of the RPL macro.
- Insertions into a variable-length RRDS use the free space provided during the definition of the data set or the free space that develops because of control interval and control area splits.

As for a fixed-length RRDS, you can insert records into a variable-length RRDS either sequentially or directly.

Sequential Insertion. When a record is inserted sequentially into a variable-length RRDS, it is assigned the next available relative record number in sequence. The assigned number is returned in the argument field of the RPL. Use mass sequential insertion with a variable-length RRDS.

Direct Insertion. Direct or skip-sequential insertion of a record into a variable-length RRDS places the record as specified by the relative record number in the argument field of the RPL. If you specify a duplicate relative record number, VSAM sets an error return code in the RPL and rejects the request.

Insertions into a Linear Data Set

Linear data sets cannot be processed at the record level. Use of the GET, PUT and POINT macros is not permitted at the record level. You must use the DIV macro to process a linear data set. See *z/OS MVS Programming: Assembler Services Guide* for information about using DIV.

Retrieving Records

The GET macro is used to retrieve records. To retrieve records for update, use the GET macro with the PUT macro. When you retrieve records either sequentially or directly, VSAM returns the length of the retrieved record to the RECLEN field of the RPL.

Sequential Retrieval

Records can be retrieved sequentially using keyed access or addressed access.

Keyed Sequential Retrieval. The first time your program accesses a data set for keyed sequential access (RPL OPTCD=(KEY,SEQ)), VSAM is positioned at the first record in the data set in key sequence if and only if the following is true:

1. Nonshared resources are being used.
2. There have not been any previous requests against the file.

If VSAM picks a string that has been used previously this implicit positioning does not occur. Therefore, with concurrent or multiple RPL's, it is best to initiate your own POINTs and positioning to prevent logic errors.

With shared resources, you must always use a POINT macro to establish position. A GET macro can then retrieve the record. Certain direct requests can also hold position. See Table 9 on page 150 for details on when positioning is retained or released. VSAM checks positioning when processing modes are changed between requests.

For keyed sequential retrieval of a fixed-length or variable-length RRDS, the relative record number is treated as a full key. If a deleted record is found during sequential retrieval, it is skipped and the next record is retrieved. The relative record number of the retrieved record is returned in the argument field of the RPL.

Addressed Sequential Retrieval. Retrieval by address is identical to retrieval by key, except the search argument is a RBA, which must be matched to the RBA of a record in the data set. When a processing program opens a data set with nonshared resources for addressed access, VSAM is positioned at the record with RBA of zero to begin addressed sequential processing. A sequential GET request causes VSAM to retrieve the data record at which it is positioned and then positions VSAM at the next record. The address specified for a GET or a POINT must correspond to the beginning of a data record; otherwise the request is not valid. Spanned records stored in a key-sequenced data set cannot be retrieved using addressed retrieval.

You cannot predict the RBAs of compressed records.

GET-previous (backward-sequential) processing is a variation of normal keyed or addressed-sequential processing. Instead of retrieving the next record in ascending sequence (relative to current positioning in the data set), GET-previous processing retrieves the next record in descending sequence. To process records in descending sequence, specify BWD in the RPL OPTCD parameter. Select GET-previous processing for POINT, GET, PUT (update only), and ERASE operations. The initial positioning by POINT, other than POINT LRD, requires that you specify a key. The following GET-previous processing does not need any specified key to retrieve the next record in descending sequence.

GET-previous processing is not permitted with control interval or skip-sequential processing.

POINT Macro for Positioning

You can use the POINT macro to begin retrieving records sequentially at a place other than the beginning of the data set. The POINT macro places VSAM at the record with the specified key or relative byte address. However, it does not provide data access to the record. If you specify a generic key (a leading portion of the key field), the record pointed to is the first of the records having the same generic key. The POINT macro can position VSAM for either forward or backward processing, if FWD or BWD was specified in the RPL OPTCD parameter.

If, after positioning, you issue a direct request through the same request parameter list, VSAM drops positioning unless NSP or UPD was specified in the RPL OPTCD parameter.

When a POINT is followed by a VSAM GET/PUT request, both the POINT and the subsequent request must be in the same processing mode. For example, a POINT with RPL OPTCD=(KEY,SEQ,FWD) must be followed by GET/PUT with RPL OPTCD=(KEY,SEQ,FWD); otherwise, the GET/PUT request is rejected.

For skip-sequential retrieval, you must indicate the key of the next record to be retrieved. VSAM skips to the next record's index entry by using horizontal pointers in the sequence set to find the appropriate sequence-set index record and scan its entries. The key of the next record to be retrieved must always be higher in sequence than the key of the preceding record retrieved.

If your request fails, with an error code, positioning cannot be maintained. To determine if positioning is maintained when a logical error occurs, see *z/OS DFSMS Macro Instructions for Data Sets*. Positioning is always released when you specify the ENDREQ macro.

Direct Retrieval

Records can also be retrieved directly using keyed access or addressed access.

Keyed Direct Retrieval. For a key-sequenced data set does not depend on prior positioning. VSAM searches the index from the highest level down to the sequence set to retrieve a record. Specify the record to be retrieved by supplying one of the following:

- The exact key of the record
- An approximate key, less than or equal to the key field of the record
- A generic key

You can use an approximate specification when you do not know the exact key. If a record actually has the key specified, VSAM retrieves it. Otherwise, it retrieves the record with the next higher key. Generic key specification for direct processing causes VSAM to retrieve the first record having that generic key. If you want to retrieve all the records with the generic key, specify RPL OPTCD=NSP in your direct request. That causes VSAM to position itself at the next record in key sequence. Then retrieve the remaining records sequentially.

To use direct or skip-sequential access to process a fixed-length or variable-length RRDS, you must supply the relative record number of the record you want in the argument field of the RPL macro. For a variable-length RRDS, you also must supply the record length in the RECLEN field of the RPL macro. If you request a deleted record, the request causes a no-record-found logical error.

A fixed-length RRDS has no index. VSAM takes the number of the record to be retrieved and calculates the control interval that contains it and its position within the control interval.

Addressed Direct Retrieval. Requires the RBA of each individual record is specified; previous positioning is not applicable.

With direct processing, optionally specify RPL OPTCD=NSP to indicate the position is maintained following the GET. Your program can then process the following records sequentially in either a forward or backward direction.

Updating Records

The GET and PUT macros are used to update records. A GET for update retrieves the record and the following PUT for update stores the record the GET retrieved.

When you update a record in a key-sequenced data set, you cannot alter the primary-key field.

Changing Record Length

You can update the contents of a record with addressed access, but you cannot alter the record's length. To change the length of a record in an entry-sequenced data set, you must store it either at the end of the data set (as a new record) or in the place of an inactive record of the same length. You are responsible for marking the old version of the record as inactive.

Processing the Data Component of a Key-Sequenced Data Set

You can process the data component separately from the index component. Processing the data component separately lets you print or dump the data component and the index component of a key-sequenced data set individually. *However, do not process only the data component if you plan to update the data set.* Always open the cluster when updating a key-sequenced data set.

Deleting Records

After a GET for update retrieves a record, an ERASE macro can delete the record. The ERASE macro can be used only with a key-sequenced data set or a fixed-length or variable-length RRDS. When you delete a record in a key-sequenced data set or variable-length RRDS, the record is physically erased. The space the record occupied is then available as free space.

You can erase a record from the base cluster of a path only if the base cluster is a key-sequenced data set. If the alternate index is in the upgrade set in which UPGRADE was specified when the alternate index was defined, it is modified automatically when you erase a record. If the alternate key of the erased record is unique, the alternate index data record with that alternate key is also deleted.

When you erase a record from a fixed-length RRDS, the record is set to binary zeros and the control information for the record is updated to indicate an empty slot. Reuse the slot by inserting another record of the same length into it.

With an entry-sequenced data set, you are responsible for marking a record you consider to be deleted. As far as VSAM is concerned, the record is not deleted. Reuse the space occupied by a record marked as deleted by retrieving the record for update and storing in its place a new record of the same length.

Deferring and Forcing Buffer Writing

For integrity reasons, it is sometimes desirable to force the data buffer to be written after a PUT operation. At other times, it is desirable to defer the writing of a buffer as long as possible to improve performance. At the time the PUT is issued, if the RPL OPTCD specifies direct processing (DIR), and NSP is not specified, forced writing of the buffer occurs. Otherwise, writing is deferred. An ERASE request follows the same buffer writing rules as the PUT request. If LSR and GSR deferred writes are not specified, an ENDREQ macro always forces the current modified data buffer to be written.

Retaining and Positioning Data Buffers

Some operations retain positioning while others release it. In a similar way, some operations hold onto a buffer and others release it with its contents. Table 9 shows which RPL options result in the retention of data buffers and positioning, and which options result in the release of data buffers and positioning.

Table 9. Effect of RPL Options on Data Buffers and Positioning

RPL Options	Retained	Released
SEQ	*	
SKP	*	
DIR NSP	*	
DIR (no NSP)		*
DIR LOC	*	
UPD (with Get)	*	

Notes:

1. A sequential GET request for new control intervals releases the previous buffer.
2. The ENDREQ macro and the ERASE macro with RPL OPTCD=DIR release data buffers and positioning.
3. Certain options that retain positioning and buffers on normal completion cannot do so if the request fails with an error code. See *z/OS DFSMS Macro Instructions for Data Sets* to determine if positioning is maintained if a logical error occurs.

The following operation uses but immediately releases a buffer and does not retain positioning:

```
GET  RPL OPTCD=(DIR,NUP,MVE)
```

Processing Multiple Strings

In multiple string processing, there can be multiple independent RPLs within an address space for the same data set. The data set can have multiple tasks that share a common control block structure. There are several ACB and RPL arrangements to indicate that multiple string processing occurs:

- In the first ACB opened, STRNO or BSTRNO is greater than 1.
- Multiple ACBs are opened for the same data set within the same address space and are connected to the same control block structure.
- Multiple concurrent RPLs are active against the same ACB using asynchronous requests.
- Multiple RPLs are active against the same ACB using synchronous processing with each requiring positioning to be held.

If you are doing multiple string update processing, you must consider VSAM look-aside processing and the rules surrounding exclusive use. Look-aside means VSAM checks its buffers to see if the control interval is already present when requesting an index or data control interval.

For GET nonupdate requests, an attempt is made to locate a buffer already in storage. As a result, a down-level copy of the data can be obtained either from buffers attached to this string or from secondary storage.

For GET to update requests, the buffer is obtained in exclusive control, and then read from the device for the latest copy of the data. If the buffer is already in exclusive control of another string, the request fails with an exclusive control feedback code. If you are using shared resources, the request can be queued, or can return an exclusive control error.

The exclusive use rules follow:

1. If a given string obtains a record with a GET for update request, the control interval is not available for update or insert processing by another string.
2. If a given string is in the process of a control area split caused by an update with length change or an insert, that string obtains exclusive control of the entire control area being split. Other strings cannot process insert or update requests against this control area until the split is complete.

If you are using nonshared resources, VSAM does not queue requests that have exclusive control conflicts, and you are required to clear the conflict. If a conflict is found, VSAM returns a logical error return code, and you must stop activity and clear the conflict. If the RPL that caused the conflict had exclusive control of a control interval from a previous request, you issue an ENDREQ before you attempt to clear the problem. Clear the conflict in one of three ways:

- Queue until the RPL holding exclusive control of the control interval releases that control and then reissue the request
- Issue an ENDREQ against the RPL holding exclusive control to force it to release control immediately
- Use shared resources and issue MRKBFR MARK=RLS.

Note: If the RPL includes a correctly specified MSGAREA and MSGLEN, the address of the RPL holding exclusive control is provided in the first word of the MSGAREA. The RPL field, RPLDDDD, contains the RBA of the requested control interval.

Making Concurrent Requests

With VSAM, you can maintain concurrent positioning for many requests to a data set. Thus, processing many portions of a data set concurrently, even within the same program, providing many RPLs and strings for that data set.

Strings (sometimes called *place holders*) are like cursors, each represents a position in the data set and are like holding your finger in a book to keep the place. The same ACB is used for all requests, and the data set needs to be opened only once. This means, for example, you could be processing a data set sequentially using one RPL, and at the same time, using another RPL, directly access selected records from the same data set.

Keep in mind, though, that strings are not “owned” by the RPL any longer than the request holds position. Once a request gives up position (for example, with an ENDREQ), that string is free to be used by another request and must be repositioned in the data set by the user.

For each request, a string defines the set of control blocks for the exclusive use of one request. For example, if you use three RPLs, you should specify three strings. If the number of strings you specify is not sufficient, and you are using NSR, the operating system dynamically extends the number of strings as needed by the concurrent requests for the ACB. Strings allocated by dynamic string addition are not necessarily in contiguous storage.

Dynamic string addition does not occur with LSR and GSR. Instead, you get a logic error if you have more requests than available strings.

The maximum number of strings that can be defined or added by the system is 255. Therefore, the maximum number of concurrent requests holding position in one data set at any one time is 255.

Using a Path to Access Records

When you are processing records sequentially using a path, records from the base cluster are returned according to ascending or, if you are retrieving the previous record, descending alternate key values. If there are several records with a nonunique alternate key, those records are returned in the order they were entered into the alternate index. READNEXT and READPREV returns these nonunique alternate index records in the same sequence. VSAM sets a return code in the RPL when there is at least one more record with the same alternate key to be processed. For example, if there are three data records with the alternate key 1234, the return code would be set during the retrieval of records one and two and would be reset during retrieval of the third record.

When you use direct or skip-sequential access to process a path, a record from the base data set is returned according to the alternate key you specified in the argument field of the RPL macro. If the alternate key is not unique, the record first entered with that alternate key is returned and a feedback code (duplicate key) is set in the RPL. To retrieve the remaining records with the same alternate key, specify RPL OPTCD=NSP when retrieving the first record with a direct request, and then switch to sequential processing.

You can insert and update data records in the base cluster using a path if:

- The PUT request does not result in nonunique alternate keys in an alternate index (defined with the UNIQUEKEY attribute). However, if a nonunique alternate key is generated and the NONUNIQUEKEY attribute is specified, updating can occur.
- You do not change the key of reference between the time the record was retrieved for update and the PUT is issued.
- You do not change the primary key.

When the alternate index is in the upgrade set, the alternate index is modified automatically by inserting or updating a data record in the base cluster. If the updating of the alternate index results in an alternate index record with no pointers to the base cluster, the alternate-index record is erased.

Rule: When you use SHAREOPTIONS 2, 3, and 4, you must continue to ensure read/write integrity when issuing concurrent requests (such as GETs and PUTs) on the base cluster and its associated alternate indexes. Failure to ensure read/write integrity might temporarily cause “No Record Found” or “No Associated Base Record” errors for a GET request. Bypass such errors by reissuing the GET request, but it is best to prevent the errors by ensuring read/write integrity.

Making Asynchronous Requests

In synchronous mode, VSAM does not return to your program from a PUT or GET operation until it has completed the operation. In asynchronous mode, VSAM returns control to your program before completing a PUT or a GET. A program in asynchronous mode can perform other useful work while a VSAM PUT or GET is completed.

Asynchronous mode can improve throughput with direct processing because it permits processing to overlap with accesses from and to the direct access device. When reading records directly, each read often involves a seek on the direct access device, a slow operation. In synchronous mode, this seek time does not overlap with other processing.

Specifying Asynchronous Mode

To specify asynchronous mode, you must specify OPTCD=ASY rather than OPTCD=SYN in the RPL.

Checking for Completion of Asynchronous Requests

Suppose your program is ready to process the next record, but VSAM is still trying to obtain that record. (The next record is not yet read in from the direct access device.) You might need to stop execution of the program and wait for VSAM to complete reading in the record. The CHECK macro stops executing the program until the operation in progress is complete. You must issue a CHECK macro after each request for an RPL. If you attempt another request without an intervening CHECK, that request is rejected.

Once the request is completed, CHECK releases control to the next instruction in your program, and frees up the RPL for use by another request.

Ending a Request

Suppose you determine that you do not want to complete a request that you initiated. For example, suppose you determine during the processing immediately following a GET that you do not want the record you just requested. You can use the ENDREQ macro to cancel the request. Using the ENDREQ macro has the following advantages:

- Avoids checking an unwanted asynchronous request.
- Writes any unwritten data or index buffers in use by the string.
- Cancels the VSAM positioning on the data set for the RPL.

Recommendation: If you issue the ENDREQ macro, it is important that you check the ENDREQ return code to make sure it completes successfully. If an asynchronous request does not complete ENDREQ successfully, you must issue the CHECK macro. The data set cannot be closed until all asynchronous requests successfully complete either ENDREQ or CHECK. ENDREQ waits for the target RPL to post, so it should not be issued in an attempt to end a hung request.

Closing Data Sets

The CLOSE macro disconnects your program from a data set. It causes VSAM to take the following actions:

- Write any unwritten data or index records whose contents have changed.
- Update the catalog entry for the data set if necessary (if the location of the end-of-file indicator has changed, for example).
- Write SMF records if SMF is being used.
- Restore control blocks to the status they had before the data set was opened.
- Release virtual storage obtained during OPEN processing for additional VSAM control blocks and VSAM routines.
- Release (only for extended format key-sequenced data sets) all space after the high-used RBA (on a CA boundary) up to the high-allocated RBA if partial release was specified at open time.

For data sets that did not go through a normal CLOSE processing (for example, due to power failure), VSAM OPEN performs an implicit VERIFY. This performs the same function as when the VERIFY is explicitly performed with the access method services command. If the VERIFY is successful, the application is passed a zero return code and a unique reason code, indicating that the VERIFY was performed on its behalf. If the VERIFY is unsuccessful, VSAM OPEN passes a unique return code and reason code. This is a warning. It is the responsibility of the application to determine if processing should continue. Implicit VERIFYs are not performed for linear data sets. VSAM OPEN completes with return and reason codes as described previously.

Note: The success of a VERIFY command does not mean that the data set does not have any errors. It means that VSAM successfully located the end of the data set.

If a record management error occurs while CLOSE is flushing buffers, the data set's catalog information is not updated. The catalog cannot properly reflect the data set's status and the index cannot accurately reflect some of the data records. If the program enters an abnormal termination routine (ABEND), all open data sets are closed. The VSAM CLOSE invoked by ABEND does not update the data set's catalog information, it does not complete outstanding I/O requests, and buffers are not flushed. The catalog cannot properly reflect the cluster's status, and the index cannot accurately reference some of the data records. Use the access method services VERIFY command to correct catalog information. The use of VERIFY is described in "Using VERIFY to Process Improperly Closed Data Sets" on page 53.

When processing asynchronous VSAM requests, all strings must be quiesced by issuing the CHECK macro or the ENDREQ macro before issuing CLOSE or CLOSE TYPE=T (temporary CLOSE).

CLOSE TYPE=T causes VSAM to complete any outstanding I/O operations, update the catalog if necessary, and write any required SMF records. Processing can continue after a temporary CLOSE without issuing an OPEN macro.

If a VSAM data set is closed and CLOSE TYPE=T is not specified, you must reopen the data set before performing any additional processing on it.

When you issue a temporary or a permanent CLOSE macro, VSAM updates the data set's catalog records. If your program ends with an abnormal end (ABEND) without closing a VSAM data set the data set's catalog records are not updated, and contain inaccurate statistics.

It is the user's responsibility to ensure that shared DD statements are not dynamically deallocated until all ACBs that share these DD statements are closed. For more information about dynamic allocation, see *z/OS MVS JCL User's Guide*.

Operating in SRB or Cross-Memory Mode

Operating in service request block (SRB) or cross-memory mode enables you to use structures in other address spaces to increase the amount of space available. SRB and cross-memory modes are supervisor modes of operation reserved for authorized users. Cross-memory is a complex concept, and there are several warnings and restrictions associated with it. See *z/OS MVS Programming: Authorized Assembler Services Guide*.

In VSAM operate only in cross-memory or SRB mode for synchronous, supervisor state requests with shared resources or improved control interval (ICI) access. For data sets not processed with ICI, an attempt to invoke VSAM asynchronously, in problem state, or with nonshared resources in either cross-memory or SRB mode results in a logical error. This error is not generated for ICI.

VSAM makes no attempt to synchronize cross-memory mode requests. For non-ICI processing, the RPL must specify WAITX, and a UPAD exit (user processing exit routine) must be provided in an exit list to handle the wait and post processing for cross-memory requests; otherwise a VSAM error code is returned.

For cross-memory mode requests, VSAM does not do wait processing when a UPAD for wait returns to VSAM. For non-cross-memory task mode, however, if the UPAD taken for wait returns with ECB not posted VSAM issues a WAIT supervisor call instruction (SVC). For either mode when a UPAD is taken for post processing returns, VSAM assumes the ECB has been marked complete and does not do post processing.

ICI in cross-memory mode assumes (without checking) the request is synchronous. UPAD is not required. If UPAD routine is not provided, I/O wait and post processing is done by suspend and resume. There is no resource wait/post processing for ICI. See "Improved Control Interval Access" on page 191 for information about ICI.

SRB mode does not require UPAD. If a UPAD is provided for an SRB mode request, it is taken only for I/O wait and resource wait processing.

In cross-memory or SRB mode, record management cannot issue any supervisor call instructions (SVCs). Whenever VSAM cannot avoid issuing an SVC, it sets an RPL return code to indicate that you must change processing mode so that you are running under a TCB (task control block) in the address space in which the data set was opened. You cannot be in cross-memory mode. Then reissue the request to permit the SVC to be issued by VSAM. The requirement for VSAM to issue an SVC is kept to a minimum. Areas identified as requiring a TCB not in cross-memory mode are EXCEPTIONEXIT, loaded exits, EOV (end-of-volume), dynamic string addition, and alternate index processing.

If a logical error or an end-of-data condition occurs during cross-memory or SRB processing, VSAM attempts to enter the LERAD (logical error) or EODAD (end-of-data-set) exit routine. If the routine must be loaded, it cannot be taken because loading involves an SVC; VSAM sets the RPL feedback to indicate “invalid TCB”. If an I/O error occurs during cross-memory or SRB processing and an EXCEPTIONEXIT or loaded SYNAD (physical error exit) routine is specified, these routines cannot be taken; the RPL feedback indicates an I/O error condition.

See “Chapter 17. Coding VSAM User-Written Exit Routines” on page 241 for more information.

Using VSAM Macros in Programs

At this point it is important to see how all of these macros work together in a program. Figure 21 shows the relationship between JCL and the VSAM macros in a program.

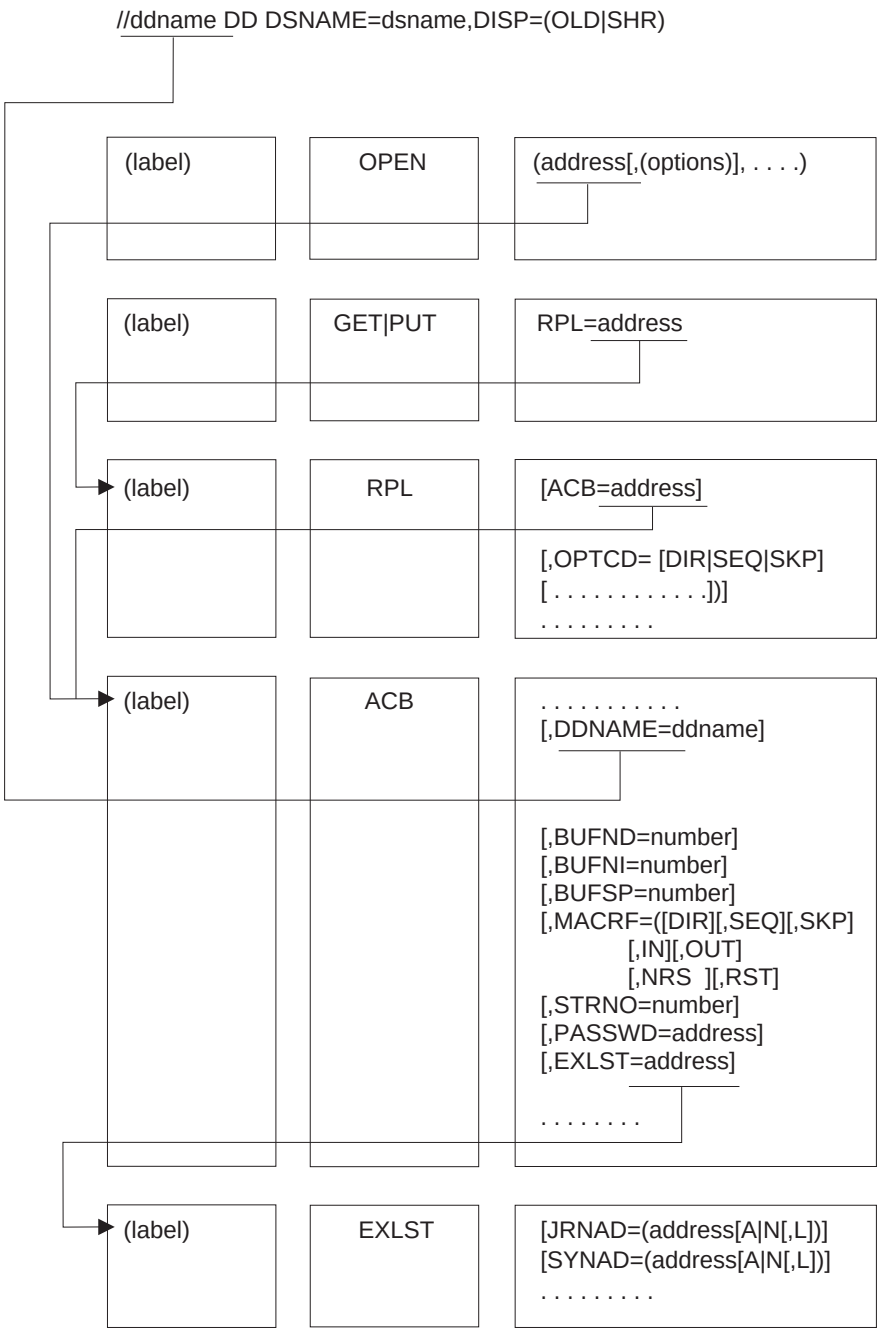


Figure 21. VSAM Macro Relationships

Figure 22 on page 158 is a skeleton program that shows the relationship of VSAM macros to each other and to the rest of the program.

```
START  CSECT
         SAVE(14,12)                      Standard entry code
         .
         B      INIT                      Branch around file specs

MASACB   ACB    DDNAME=MASDS,AM=VSAM,     File specs                X
                MACRF=(KEY,SEQ,OUT),                                X
                EXLST=EXITS,                                        X
                RMODE31=ALL

MASRPL   RPL    ACB=MASACB,                                         X
                OPTCD=(KEY,SEQ,NUP,MVE,SYN),                        X
                AREA=WA,                                            X
                AREALEN=80,                                         X
                RECLEN=80

EXITS    EXLST  LERAD=LOGER,                                        X
                JRNAD=JOURN

TRANDCB  DCB    DDNAME=TRANDS,                                      X
                DSORG=PS,                                           X
                MACRF=GM,                                           X
                EODAD=EOTRF,                                        X
                LRECL=80,                                           X
                BLKSIZE=80,                                         X
                RECFM=F
INIT     .                                Program initialization
         .
         OPEN   (MASACB,,TRANDCB)        Connect data sets
         .
         GET    TRANDCB,WA               Processing loop
         .
         PUT    RPL=MASRPL
         .
EOTRF    CLOSE  (MASACB,,TRANDCB)        Disconnect data sets
         .
         RETURN (14,12)                  Return to calling routine
LOGER    Exit routines
         .
JOURN    .
         .
WA       DS     CL80                     Work area
         END
```

Figure 22. Skeleton VSAM Program

Chapter 11. Optimizing VSAM Performance

This chapter covers the following topics, describing many of the options and factors that influence or determine the performance of both VSAM and the operating system.

Topic	Page
Optimizing Control Interval Size	159
Optimizing Control Area Size	164
Optimizing Free Space Distribution	166
Using Index Options	180
Obtaining Diagnostic Information	181
Migrating from the Mass Storage System	181
Using Hiperbatch	181

Most of the options are specified in the access method services DEFINE command when a data set is defined. Sometimes options can be specified in the ACB and GENCB macros and in the DD AMP parameter.

Optimizing Control Interval Size

You can let VSAM select the size of a control interval for a data set, you can request a particular control interval size in the DEFINE command, or you can specify data class in DEFINE and use the CISIZE attribute assigned by your storage administrator. You can improve VSAM's performance by specifying a control interval size in the DEFINE command, depending on the particular storage and access requirements for your data set. See "Control Intervals" on page 76 for information about the structure and contents of control intervals.

Control interval size affects record processing speed and storage requirements in the following ways:

- **Buffer space.** Data sets with large control interval sizes require more buffer space in virtual storage. For information about how much buffer space is required, see .
- **I/O operations.** Data sets with large control interval sizes require fewer I/O operations to bring a given number of records into virtual storage; fewer index records must be read. It is best to use large control interval sizes for sequential and skip-sequential access. Large control intervals are not beneficial for keyed direct processing of a key-sequenced data set or variable-length RRDS.
- **Free space.** Free space is used more efficiently (fewer control interval splits and less wasted space) as control interval size increases relative to data record size. For more information about efficient use of free space, see "Optimizing Free Space Distribution" on page 166.

Control Interval Size Limitations

When you request a control interval size, you must consider the length of your records and whether the SPANNED parameter has been specified.

© Copyright IBM Corp. 1987, 2002

The valid control interval sizes and block sizes are from 512 to 8192 bytes in increments of 512 bytes, and from 8 KB to 32 KB in increments of 2 KB for objects cataloged in catalogs and for the catalogs themselves. Control interval size must be a whole number of physical blocks. If you specify a control interval that is not a proper multiple for the supported block size, VSAM increases it to the next multiple. For example, 2050 is increased to 2560.

The valid control interval and block sizes for index components are 512, 1024, 2048, and 4096 bytes.

The block size of the index component is always equal to the control interval size. However, the block size for the data component and index component might differ.

Unless the data set was defined with the SPANNED attribute, the control interval must be large enough to hold a data record of the maximum size specified in the RECORDSIZE parameter. Because the minimum amount of control information in a control interval is 7 bytes, a control interval is normally at least 7 bytes larger than the largest record in the component. For compressed data sets, a control interval is at least 10 bytes larger than the largest record after it is compressed. This allows for the control information and record prefix. Since the length of a particular record is hard to predict and since the records might not compress, it is best to assume that the largest record is not compressed. If the control interval size you specify is not large enough to hold the maximum size record, VSAM increases the control interval size to a multiple of the minimum physical block size. The control interval size VSAM provides is large enough to contain the record plus the overhead.

For a variable-length RRDS, a control interval is at least 11 bytes larger than the largest record.

The use of the SPANNED parameter removes this constraint by permitting data records to be continued across control intervals. The maximum record size is then equal to the number of control intervals per control area multiplied by control interval size minus 10. The use of the SPANNED parameter places certain restrictions on the processing options that can be used with a data set. For example, records of a data set with the SPANNED parameter cannot be read or written in locate mode. For more information about spanned records see “Spanned Records” on page 78.

Physical Block Size and Track Capacity

Figure 23 shows the relationship between control interval size, physical block size, and track capacity that is not in extended format.

<table>
<tr><td colspan="2">CI1</td><td colspan="2">CI2</td><td colspan="2">CI3</td><td colspan="2">CI4</td><td colspan="2">CI5</td><td colspan="2">CI6</td></tr>
<tr><td>PB</td><td>PB</td><td>PB</td><td>PB</td><td>PB</td><td>PB</td><td>PB</td><td>PB</td><td>PB</td><td>PB</td><td>PB</td><td>PB</td></tr>
<tr><td colspan="4">Track1</td><td colspan="2">Track2</td><td colspan="4">Track3</td><td colspan="2">Track4</td></tr>
</table>

PB - Physical Block

DA6D4437

Figure 23. Control Interval Size, Physical Track Size, and Track Capacity

The information about a track is divided into physical blocks. Control interval size must be a whole number of physical blocks. Control intervals can span tracks.

However, poor performance results if a control interval spans a cylinder boundary, because the read/write head must move between cylinders.

The physical block size is always selected by VSAM. VSAM chooses the largest physical block size that exactly divides into the control interval size. The block size is also based on device characteristics.

Track Allocations versus Cylinder Allocations

All space specifications except cylinders are converted to the appropriate number of tracks. Usually, cylinder allocations provide better performance than track allocations. Performance also depends on the control interval size, buffer size, the number of buffers, and index options.

Data Control Interval Size

You can either specify a data control interval size or default to a system-calculated control-interval size. If you do not specify a size, the system calculates a default value that best uses the space on the track for the average record size of spanned records or the maximum record size of nonspanned records.

You can cache all or some of the VSAM record level sharing (RLS) data in a coupling facility (CF) cache structure defined to DFSMS. The CF cache structure is faster than the DASD caching structure, so performance with VSAM RLS data improves. All active systems in a sysplex must have the greater than 4K CF caching code installed before the function is enabled.

To set up RLS CF caching, use the following values:

- ALL or UPDATESONLY or NONE for the SMS DATACLAS RLS CF Cache Value keyword

  To allow greater than 4K caching of DFSMS VSAM data sets open for RLS processing, you need to make the following changes:.

  – You can change the value of the SMS DATACLAS RLS CF Cache Value keyword if you do not want caching of all VSAM RLS data:

    ALL Indicates that RLS is to cache VSAM index and data components. **ALL** is the default.

    NONE Indicates that RLS is to cache only the VSAM index data. The data components are not to be placed in the cache structure.

    UPDATESONLY
    Indicates that RLS is to place only WRITE requests in the cache structure.

  – VSAM honors the RLS CF Cache Value keyword only when you specify RLS_MaxCfFeatureLevel(A) and all systems in the sysplex can run the greater than 4K caching code.

    To determine the code level on each system in the sysplex and whether the RLS CF Cache Value keyword is honored, use the 'D SMS,SMSVSAM', 'D SMS,SMSVSAM,ALL', D SMS,CFCACHE() operator commands. When DFSMS cache structures connect to the system, VSAM RLS issues an IGW500I message to indicate that greater than 4K caching is active. DFSMS cache structures connect to the system through the first instance of a sphere opened on each system.

- You can specify the following values for the RLS_MaxCfFeatureLevel keyword:

  – A—This value allows greater than 4K caching if all active VSAM RLS instances in the sysplex have the correct level of code.

  – Z—This is the default value if you do not specify RLS_MaxCfFeatureLevel in the active SMS configuration. Greater than 4K caching is not allowed.

- RLS_MaxCfFeatureLevel keyword in the SETSMS command
- RLS_MaxCfFeatureLevel keyword in the SET SMSxx command
- RLS_MaxCfCacheFeatureLevel in the D SMS,OPTIONS command

If a CONTROLINTERVALSIZE value is specified on the cluster level, this value propagates to the component level at which no CONTROLINTERVALSIZE value has been specified.

Normally, a 4096-byte data control interval is reasonably good regardless of the DASD device used, processing patterns, or the processor. A linear data set requires a control interval size of 4096 to 32768 bytes in increments of 4096 bytes. However, there are some special considerations that might affect this choice:

- If you have very large control intervals, more pages are required to be fixed during I/O operations. This could adversely affect the operation of the system.
- Small records in a data control interval can result in a large amount of control information. Often free space cannot be used.
- The type of processing you use can also affect your choice of control interval size:
  - ***Direct processing.*** When direct processing is predominant, a small control interval is preferable, because you are only retrieving one record at a time. Select the smallest data control interval that uses a reasonable amount of space.
  - ***Sequential processing.*** When sequential processing is predominant, larger data control intervals can be good choices. For example, given a 16 KB data buffer space, it is better to read two 8 KB control intervals with one I/O operation than four 4 KB control intervals with two I/O operations.
  - ***Mixed processing.*** If the processing is a mixture of direct and sequential, a small data control interval with multiple buffers for sequential processing can be a good choice.

If you specify free space for a key-sequenced data set or variable-length RRDS, the system determines the number of bytes to be reserved for free space. For example, if control interval size is 4096, and the percentage of free space in a control interval has been defined as 20%, 819 bytes are reserved. Free space calculations drop the fractional value and use only the whole number.

To find out what values are actually set in a defined data set, issue the access method services LISTCAT command.

Index Control Interval Size

For a key-sequenced data set, either specify an index control interval size or default to a system-calculated size. If you do not specify a size, VSAM calculates the CISIZE value based on the expected key-compression ratio of 3:1 and the number of data CIs per control area in the data set. After VSAM determines the number of CIs in a control area, it estimates whether the user-specified size is large enough for all the CIs in a control area. (See “How VSAM Adjusts Control Interval Size” on page 164.) If the size is too small, the system increases the size of the index control interval to VSAM’s minimum acceptable size. If the specified size is larger than the minimum size that VSAM calculated, the system uses specified size.

You might need a larger CI than the size that VSAM calculated, depending on the allocation unit, the data CI size, the key length, and the key content as it affects compression. (It is rare to have the entire key represented in the index, because of key compression.) If the keys for the data set do not compress according to the estimated ratio (3:1), the index CI size that VSAM calculated might be too small, resulting in the inability to address CIs in one or more CAs. This results in allocated space that is unusable in the data set. After the first define (DEFINE), a catalog listing (LISTCAT) shows the number of control intervals in a control area and the key length of the data set.

Optimizing VSAM Performance

You can use the number of control intervals and the key length to estimate the size of index record necessary to avoid a control area split, which occurs when the index control interval size is too small. To make a general estimate of the index control interval size needed, multiply one half of the key length (KEYLEN) by the number of data control intervals per control area (DATA CI/CA):

```
(KEYLEN/2) * DATA CI/CA ≤ INDEX CISIZE
```

The use of a 2:1 ratio rather than 3:1, which VSAM uses, allows for some of the additional overhead factors in the actual algorithm for determining the CI size.

How VSAM Adjusts Control Interval Size

The control interval sizes you specify when the data set is defined are not necessarily the ones that appear in the catalog. VSAM makes adjustments, if possible, so that control interval size conforms to proper size limits, minimum buffer space, adequate index-to-data size, and record size. For example, this happens when your data set is defined.

1. Specifies data and index control interval size. After VSAM determines the number of control intervals in a control area, it estimates whether one index record is large enough to handle all control intervals in the control area. If not, the size of the index control interval is increased, if possible. If the size cannot be increased, VSAM decreases the number of control intervals in the control area.
2. Specifies maximum record size as 2560 and data control interval size as 2560, and have no spanned records. VSAM adjusts the data control interval size to 3072 to permit space for control information in the data control interval.
3. Specifies buffer space as 4K, index control interval size as 512, and data control interval size as 2K. VSAM decreases the data control interval to 1536. Buffer space must include space for two data control intervals and one index control interval at DEFINE time. For more information about buffer space requirements see .

Optimizing Control Area Size

You cannot explicitly specify control-area size. Generally, the primary and secondary space allocation amounts determine the control-area size:

- If either the primary or secondary allocation is smaller than one cylinder, the smaller value is used as the control-area size. If RECORDS is specified, the allocation is rounded up to full tracks.
- If both primary and secondary allocations are equal to or larger than one cylinder, the control-area size is one cylinder, the maximum size for a control area.

The following examples show how the control-area size is generally determined by the primary and secondary allocation amount. The index control-interval size and buffer space can also affect the control-area size. The following examples are based on the assumption that the index CI size is large enough to handle all the data CIs in the CA and the buffer space is large enough not to affect the CI sizes:

- **CYLINDERS(5,10)**—Results in a 1-cylinder control-area size.
- **KILOBYTES(100,50)**—The system determines the control area based on 50 KB, resulting in a 1-track control-area size.
- **RECORDS(2000,5)**—Assuming 10 records would fit on a track, results in a 1-track control-area size.
- **TRACKS(100,3)**—Results in a 3-track control-area size.
- **TRACKS(3,100)**—Results in a 3-track control-area size.

A spanned record cannot be larger than the size of a control area minus the size of the control information (10 bytes per control interval). Therefore, do not specify a primary or secondary allocation that is not large enough to contain the largest spanned record.

Note: If space is allocated in kilobytes, megabytes, or records, the system sets the control area size equal to multiples of the minimum number of tracks or cylinders required to contain the specified kilobytes, megabytes, or records. Space is not assigned in units of bytes or records.

If the control area is smaller than a cylinder, its size will be an integral multiple of tracks, and it can span cylinders. However, a control area can never span an extent of a data set, which is always composed of a whole number of control areas. For more information about allocating space for a data set, see “Allocating Space for VSAM Data Sets” on page 111.

Advantages of a Large Control Area Size

Control area size has significant performance implications. One-cylinder control areas have the following advantages:

- There is a smaller probability of control area splits.
- The index is more consolidated. One index record addresses all the control intervals in a control area. If the control area is large, fewer index records and index levels are required. For sequential access, a large control area decreases the number of reads of index records.
- There are fewer sequence set records. The sequence set record for a control area is always read for you. Fewer records means less time spent reading them.
- If the sequence set of the index is imbedded on the first track of the control area replicated to reduce the rotational delay inherent when reading from the device.
- If you have allocated enough buffers, a large control area lets you read more buffers into storage at one time. A large control area is useful if you are accessing records sequentially.

Disadvantages of a Large Control Area Size

The following disadvantages of a one-cylinder control area must also be considered:

- If there is a control area split, more data is moved.
- During sequential I/O, a large control area ties up more real storage and more buffers.

Optimizing Free Space Distribution

With the DEFINE command, either specify the percentage of free space in each control interval and the percentage of free control intervals per control area or specify data class and use the FREESPACE attribute assigned through the ACS routines established by your storage administrator.

Free space improves performance by reducing the likelihood of control interval and control area splits. This, in turn, reduces the likelihood of VSAM moving a set of records to a different cylinder away from other records in the key sequence. When there is a direct insert or a mass sequential insert that does not result in a split, VSAM inserts the records into available free space.

The amount of free space you need depends on the number and location of records to be inserted, lengthened, or deleted. Too much free space can result in:

- Increased number of index levels, that affects run times for direct processing.
- More direct access storage required to contain the data set.
- More I/O operations required to sequentially process the same number of records.

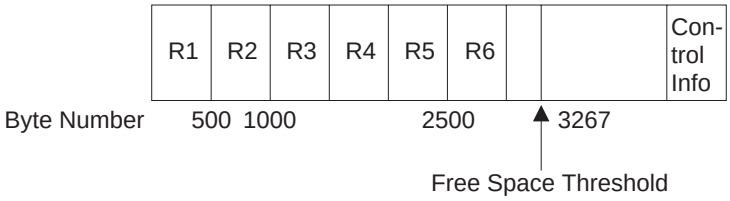
Too little free space can result in an excessive number of control interval and control area splits. These splits are time consuming, and have the following additional effects:

- More time is required for sequential processing because the data set is not in physical sequence.
- More seek time is required during processing because of control area splits.

Use LISTCAT or the ACB JRNAD exit to monitor control area splits. See “JRNAD Exit Routine to Journalize Transactions” on page 245.

When splits become frequent, reorganize the data set using REPRO or EXPORT. Reorganization creates a smaller, more efficient data set with fewer control intervals. However, reorganizing a data set is time consuming.

Figure 24 shows how free space is determined for a control interval.



FREESPACE(20 10)
CONTROLINTERVALSIZE(4096)
RECORDSIZE(500 500)

Figure 24. Determining Free Space

For this data set, each control interval is 4096 bytes. In each control interval, 10 bytes are reserved for control information. Because control interval free space is specified as 20%, 819 bytes are reserved as free space. (4096 × .20 = 819). Round down. The free space threshold is 3267 bytes. The space between the threshold and the control information is reserved as free space.

Because the records loaded in the data set are 500-byte records, there is not enough space for another record between byte 3000 and the free space threshold at byte 3267. These 267 bytes of unused space are also used as free space. This leaves 1086 bytes of free space; enough to insert two 500-byte records. Only 86 bytes are left unusable.

When you specify free space, ensure that the percentages of free space you specify yield full records and full control intervals with a minimum amount of unusable space.

Selecting the Optimal Percentage of Free Space

Determine the amount of control interval free space based on the percentage of record additions expected, and their distribution:

No additions. If no records will be added and if record sizes will not be changed, there is no need for free space.

Few additions. If few records will be added to the data set, consider a free space specification of (0 0). When records are added, new control areas are created to provide room for additional insertions.

If the few records to be added are fairly evenly distributed, control interval free space should be equal to the percentage of records to be added. (FSPC (nn 0), where nn equals the percentage of records to be added.)

Evenly distributed additions. If new records will be evenly distributed throughout the data set, control area free space should equal the percentage of records to be added to the data set after the data set is loaded. (FSPC (0 nn), where nn equals the percentage of records to be added.)

Unevenly distributed additions. If new records will be unevenly distributed throughout the data set, specify a small amount of free space. Additional splits, after the first, in that part of the data set with the most growth will produce control intervals with only a small amount of unneeded free space.

Mass insertion. If you are inserting a group of sequential records, take full advantage of mass insertion by using the ALTER command to change free space to (0 0) after the data set is loaded. For more information about mass insertion see “Inserting and Adding Records” on page 144.

Additions to a specific part of the data set. If new records will be added to only a specific part of the data set, load those parts where additions will not occur with a free space of (0 0). Then, alter the specification to (n n) and load those parts of the data set that will receive additions. The example in “Altering the Free Space Specification When Loading a Data Set” demonstrates this.

Altering the Free Space Specification When Loading a Data Set

The following example uses the ALTER command to change the FREESPACE specification when loading a data set.

Assume that a large key-sequenced data set is to contain records with keys from 1 through 300000. It is expected to have no inserts in key range 1 through 100000, some inserts in key range 100001 through 200000, and heavy inserts in key range 200001 through 300000.

An ideal data structure at loading time would be:

Key Range	Free Space
1 through 100000	None
100001 through 200000	5% control area
200001 through 300000	5% control interval and 20% control area

You can build this data structure as follows:

1. DEFINE CLUSTER and do one of the following:
   - Omit the FREESPACE parameter
   - Specify FREESPACE (0 0)
   - Specify DATACLAS and use the FREESPACE attribute assigned through the automatic class selection routines established by your storage administrator.
2. Load records 1 through 100000 with REPRO or any user program using a sequential insertion technique.
3. CLOSE the data set.
4. Change the FREESPACE value of the cluster with the access method services command ALTER *clustername* FREESPACE (0 5). Explicit specification of FREESPACE overrides the data class attribute assigned by your storage administrator.
5. Load records 100001 through 200000 with REPRO or any user program using a sequential insertion technique.
6. CLOSE the data set.
7. Change the FREESPACE value of the cluster with the access method services command ALTER *clustername* FREESPACE (5 20).
8. Load records 200001 through 300000 with REPRO or any user program using a sequential insertion technique.

This procedure has the following advantages:

- It prevents wasting space. For example, if FREESPACE (0 10) were defined for the whole data set, the free space in the first key range would all be wasted.
- It minimizes control interval and control area splits. If FREESPACE (0 0) were defined for the whole data set, there would be a very large number of control interval and control area splits for the first inserts.

Determining I/O Buffer Space for Nonshared Resource

I/O buffers are used by VSAM to read and write control intervals from DASD to virtual storage. For a key-sequenced data set or variable-length RRDS, VSAM requires a minimum of three buffers, two for data control intervals and one for an index control interval. (One of the data buffers is used only for formatting control areas and splitting control intervals and control areas.) The VSAM default is enough space for these three buffers. Only data buffers are needed for entry-sequenced, fixed-length RRDSs or for linear data sets.

To increase performance, there are parameters to override the VSAM default values. There are five places where these parameters can be specified:

- BUFFERSPACE, specified in the access method services DEFINE command. This is the least amount of storage ever provided for I/O buffers.
- BUFSP, BUFNI, and BUFND, specified in the VSAM ACB macro. This is the maximum amount of storage to be used for a data set's I/O buffers. If the value specified in the ACB macro is greater than the value specified in DEFINE, the ACB value overrides the DEFINE value.
- BUFSP, BUFNI, and BUFND, specified in the JCL DD AMP parameter. This is the maximum amount of storage to be used for a data set's I/O buffers. A value specified in JCL overrides DEFINE and ACB values if it is greater than the value specified in DEFINE.
- ACCBIAS specified in the JCL DD AMP parameter. Record access bias has six specifications:

SYSTEM	Force system-managed buffering and let the system determine the buffering technique based on the ACB MACRF and storage-class specification.
USER	Bypass system-managed buffering.
SO	System-managed buffering with sequential optimization.
SW	System-managed buffering weighted for sequential processing.
DO	System-managed buffering with direct optimization.
DW	System-managed buffering weighted for direct optimization.

- Data class Record Access Bias = SYSTEM or USER

VSAM must always have sufficient space available to process the data set as directed by the specified processing options.

Obtaining Buffers Above 16 MB

To increase the storage area available below 16 MB for your application program, request VSAM data buffers and VSAM control blocks from virtual storage above 16 MB. To do this, specify the RMODE31 parameter on the ACB macro. See "Chapter 18. Using 31-Bit Addressing Mode with VSAM" on page 263.

Optionally, specify RMODE31 in your JCL DD AMP parameter to let the user override any RMODE31 values specified when the ACB was created. If you did not specify RMODE31 the JCL AMP parameter and ACCBIAS=SYSTEM, the default value, RMODE31=BUFF, is in effect. If you attempt to reference the VSAM buffers directly (as in LOCATE mode), your program must run in 31-bit addressing mode. If your program runs in 24-bit addressing mode and you need to access VSAM buffers directly, code RMODE31=CB or RMODE31=NONE in the JCL AMP parameter.

Tuning for System-Managed Buffering

VSAM can use a technique called system-managed buffering (SMB) to determine the number of buffers and the type of buffer management to use for VSAM data sets. VSAM also determines the number of buffers to locate in Hiperspace for use in direct optimization.

To indicate that VSAM is to use SMB, specify either of the following options:

- Specify the ACCBIAS subparameter of the JCL DD statement AMP parameter and an appropriate value for record access bias.
- Specify Record Access Bias in the data class and an application processing option in the ACB.

For system-managed buffering (SMB), the data set must use both of the following options:

- System Management Subsystem (SMS) storage
- Extended format (DSNTYPE=ext in the data class)

JCL takes precedence over the specification in the data class. You must specify NSR. SMB either weights or optimizes buffer handling toward sequential or direct processing.

To optimize your extended format data sets, use the ACCBIAS subparameter of the AMP parameter along with related subparameters SMBVSP, SMBDFR, and SMBHWT. You can also use these subparameters in conjunction with Record Access Bias=SYSTEM in the data class. These subparameters are only for Direct Optimized processing.

Processing Techniques

The information in this section is for planning purposes only. It is not absolute or exact regarding storage requirements. You should use it only as a guideline for estimating storage requirements. Individual observations might vary depending on specific implementations and processing.

System-managed buffering (SMB), a feature of DFSMSdfp™, supports batch application processing. SMB takes the following actions:

1. It changes the defaults for processing VSAM data sets. This enables the system to take better advantage of current and future hardware technology.
2. It initiates a buffering technique to improve application performance. The technique is one that the application program does not specify.

You can choose or specify any of the four processing techniques that SMB implements:

- Direct Optimized (DO)
- Sequential Optimized (SO)
- Direct Weighted (DW)
- Sequential Weighted (SW)

Direct Optimized (DO). The DO processing technique optimizes for totally random record access. This is appropriate for applications that access records in a data set in totally random order. This technique overrides the user specification for nonshared resources (NSR) buffering with a local shared resources (LSR) implementation of buffering.

The following three options, SMBVSP, SMBDFR, and SMBHWT, are only for processing with the Direct Optimized technique.

- **SMBVSP.** This option specifies the amount of virtual storage to obtain for buffers when opening the data set. You can specify virtual buffer size in kilobytes, from 1K to 2048000K, or in megabytes, from 1M to 2048M. This value is the total amount of virtual storage that you can address in a single address space. This value does not specify storage that the system or the access method requires.
- **SMBDFR.** This option lets you defer writing buffers to the medium either until the buffer is required for a different request or until the data set is closed. CLOSE TYPE=T does not write the buffers to the medium when the system uses LSR processing for direct optimization. Defaults for deferred write processing depend upon the SHAREOPTIONS values, which you specify when you define the data set. The default for SHAREOPTIONS (1,3) and (2,3) is deferred write. The default for SHAREOPTIONS (3,3), (4,3), and (x, 4) is nondeferred write. If the user specifies a value for SMBDFR, this value always takes precedence over any defaults.
- **SMBHWT.** This option permits the specification of a whole decimal value from 1-99 for allocating Hiperspace buffers. The allocation is based on a multiple of the number of virtual buffers that have been allocated.

Sequential Optimized (SO). The SO technique optimizes processing for record access that is in sequential order. This is appropriate for backup and for applications that read the entire data set or a large percentage of the records in sequential order.

Direct Weighted (DW). The majority is direct processing; some is sequential. DW processing provides the minimum read-ahead buffers for sequential retrieval and the maximum index buffers for direct requests.

Sequential Weighted (SW). The majority is sequential processing; some is direct. This technique uses read-ahead buffers for sequential requests and provides additional index buffers for direct requests. The read-ahead will not be as large as the amount of data transferred with SO.

To implement SMB, an application program must specify nonshared resources (NSR) buffering, ACB MACRF=(NSR). The system does not apply SMB when any VSAM data set is opened with a request for any other buffering option, MACRF=(LSR|GSR|UBF|RLS).

The basis for the default technique is the application specification for ACB MACRF=(DIR,SEQ,SKP) Also, specification of the following values in the associated storage class (SC) influence the default technique:

- Direct millisecond response
- Direct bias
- Sequential millisecond response
- Sequential bias

You can specify the technique externally by using the ACCBIAS subparameter of the AMP= parameter. The system invokes the function only during data set OPEN processing. After SMB makes the initial decisions during that process, it has no further involvement.

Internal Processing Techniques

In addition, two internal techniques support data set creation and load-mode processing. A user cannot specify these techniques, which the system invokes internally if the data set is in load mode (HURBA=0) and if the following items specify the SYSTEM keyword:

- RECORD ACCESS BIAS for the related data class
- ACCBIAS in the AMP= parameter of the data set JCL

The two techniques are Create Optimized (CO) and Create Recovery Optimized (CR).

Create Optimized (CO): Maximum buffers to optimize load performance if you specify SPEED in the data definition.

Create Recovery Optimized (CR): Maximum buffers to optimize load performance if you specify RECOVERY in the data definition.

Processing Guidelines and Restrictions

The following guidelines and restrictions relate to processing with each technique.

Direct Optimized (DO) Guidelines. DO could result in a requirement for the most additional processor virtual storage. This results in the creation of a local shared resources (LSR) pool for each data set opened with this technique in a single application program. The size of the data set is a major factor in the processor virtual storage requirement for buffering. The size of the pool is based on the actual data set size at the time the pool is created. This means that the processor virtual storage requirement increases with each OPEN after records have been added and the data set has been extended beyond its previous size.

A separate pool is built for both data and index components, if applicable, for each data set. There is no capability to share a single pool by multiple data sets. The index pool is sized to accommodate all records in the index component. The data pool is sized to accommodate approximately 20% of the user records in the data set. As discussed previously, this size can change based on data set growth. A maximum pool size for the data component will be identified. These buffers are acquired above the 16 MB line unless overridden by the use of the RMODE31 parameter on the AMP=parameter.

The use of the SMBVSP parameter on AMP=parameter can be used to restrict the size of the pool that is built for the data component. There is no capability to override the size of the pool for the index records. The SMBHWT parameter can be used to provide buffering in Hiperspace in combination with virtual buffers for the data component. The value of this parameter is used as a multiplier of the virtual buffer space for Hiperspace buffers. This can reduce the size required for an application region, but does have implications related to processor cycle requirements. That is, all application requests must orient to a virtual buffer address. If the required data is in a Hiperspace buffer, the data must be moved to a virtual buffer after “stealing” a virtual buffer and moving that buffer to a least recently used (LRU) Hiperspace buffer.

If the optimum amount of storage required for this option is not available, SMB will reduce the number of buffers and retry the request. For data, SMB will make two attempts, with a reduced amount and a minimum amount. For an index, SMB reduces the amount of storage only once, to minimum amount. If all attempts fail, the DW technique is used. The system issues an IEC161I message to advise that this has happened. In addition, SMF type-64 records indicate whether a reduced or minimum amount of resource is being used for a data pool and whether DW is used. For more information, see *z/OS MVS System Management Facilities (SMF)*.

Restrictions on the Use of Direct Optimized (DO). The Direct Optimized (DO) technique is elected if the ACB only specifies the MACRF=(DIR) option for accessing the data set. If either SEQ|SKP are specified, either in combination with DIR or independently, DO is not selected. The selection can be overridden by the user specification of ACCBIAS=DO on the AMP=parameter of the associated DD statement.

There are some restrictions for the use of the Direct Optimized (DO) technique:

1. The application must position the data set to the beginning for any sequential processing. This assumes the first retrieval will be set to that point of the data set.
2. Data sets that have alternate indexes defined over them are not eligible for DO.

Sequential Optimized (SO) Guidelines. This technique provides the most efficient buffers for sequential application processing such as data set backup. The size of the data set is not a factor in the processor virtual storage that is required for buffering. The buffering implementation (NSR) specified by the application will not be changed for this technique. Approximately 500K of processor virtual storage for buffers, defaulted to above 16 MB, is required for this technique.

Direct Weighted (DW) Guidelines. This technique is applicable for applications in which the requests to the records in a VSAM data set are random for the majority of the accesses. In addition, it might also give some sequential performance improvement above VSAM defaults.

The size of the data set is a minor factor in the storage that is required for buffering. This technique does not change the buffering implementation that the application specified (NSR). This technique requires approximately 100K of processor storage for buffers, with a default of 16 MB.

Sequential Weighted (SW) Guidelines. This technique is applicable for applications where the requests to the records in a VSAM data set are sequential for the majority of the accesses. In addition, this technique might give some direct performance improvement over VSAM defaults.

The size of the data set is a minor factor in the amount of processor virtual storage that buffering requires. This technique does not change the buffering implementation that the application specified (NSR). This technique requires approximately 100K of processor virtual storage for buffers, with the default above 16 MB.

Create Optimized (CO) Guidelines. This is the most efficient technique, as far as physical I/Os to the data component, for loading a VSAM data set. It only applies when the data set is in initial load status and when defined with the SPEED option. The system invokes it internally, with no user control other than the specification of RECORD ACCESS BIAS in the data class or an AMP=(ACCBIAS=) value of SYSTEM.

The size of the data set is not a factor in the amount of storage that buffering requires. This technique does not change the buffering implementation that the application specified (NSR). This technique requires a maximum of approximately 2 MB of processor virtual storage for buffers, with the default above 16 MB.

Create Recovery Optimized (CR) Guidelines. The system uses this technique when a data set defined with the RECOVERY option is in initial load status. The system invokes CR internally, with no user control other than the specification of RECORD ACCESS BIAS in the data class or an AMP=(ACCBIAS=) value of SYSTEM.

The size of the data set is not a factor in the amount of storage that buffering requires. This technique does not change the buffering implementation that the application specified (NSR). This technique requires a maximum of approximately 1 MB of processor virtual storage for buffers, with the default above 16 MB.

To determine the final SMB processing technique and additional resource-handling information related to SMB, examine SMF type 64 records. This is mainly for diagnostic purposes because SMF type 64 records are gathered during the CLOSE processing of a data set. For more information on SMF records, see *z/OS MVS System Management Facilities (SMF)*.

General Considerations for the Use of SMB

The following factors affect storage requirements for SMB buffers:

- Number of VSAM data sets opened for SMB within a single application program
- Chosen or specified technique
- Data set size for some techniques

The storage for buffers for SMB techniques is obtained above 16 MB. If the application runs as AMODE=RMODE=24 and issues locate-mode requests (RPL OPTCD=(,LOC)), the AMP= parameter must specify RMODE31=NONE for data sets that use SMB.

SMB might not be the answer to all application program buffering requirements. The main purpose of SMB is to improve performance buffering options for batch application processing, beyond the options that the standard defaults provide. In the case of many large data sets and apparently random access to records, it might be better to implement a technique within the application program to share a common resource pool. The application program designer might know the access technique for the data set, but SMB cannot predict it. In such applications, it would be better to let the application program designer define the size and number of buffers for each pool. This is not unlike the requirements of high-performance database systems.

Allocating Buffers for Concurrent Data Set Positioning

To calculate the number of buffers you need, you must determine the number of strings you will use. A string is a request to a VSAM data set requiring data set positioning. If different concurrent accesses to the same data set are necessary, multiple strings are used. If multiple strings are used, each string requires exclusive control of an index I/O buffer. Therefore, the value specified for the STRNO parameter (in the ACB or GENCB macro, or AMP parameter) is the minimum number of index I/O buffers required when requests that require concurrent positioning are issued.

Allocating Buffers for Direct Access

For a key-sequenced data set or variable-length RRDS, increase performance for direct processing by increasing the number of index buffers. Direct processing always requires a top-down search through the index. Many data buffers do not increase performance, because only one data buffer is used for each access.

Data Buffers for Direct Access

Because VSAM does not read ahead buffers for direct processing, only the minimum number of data buffers are needed. Only one data buffer is used for each access. If you specify more data buffers than the minimum, this has little beneficial effect.

When processing a data set directly, VSAM reads only one data control interval at a time. For output processing (PUT for update), VSAM immediately writes the updated control interval, if OPTCD=NSP is not specified in the RPL macro.

Index Buffers for Direct Access

If the number of I/O buffers provided for index records is greater than the number of requests that require concurrent positioning (STRNO), one buffer is used for the highest-level index record. Any additional buffers are used, as required, for other index-set index records. With direct access, you should provide at least enough index buffers to be equal to the value of the STRNO parameter of the ACB, plus one if you want VSAM to keep the highest-level index record always resident.

Unused index buffers do not degrade performance, so you should always specify an adequate number. For optimum performance, the number of index buffers should at least equal the number of high-level index set control intervals plus one per string to contain the entire high-level index set and one sequence set control interval per string in virtual storage.

VSAM reads index buffers one at a time, and if you use shared resources, keep your entire index set in storage. Index buffers are loaded when the index is referred to. When many index buffers are provided, index buffers are not reused until a requested index control interval is not in storage. Note that additional index buffers is not used for more than one sequence set buffer per string unless shared resource pools are used. For large data sets, specify the number of index buffers equal to the number of index levels.

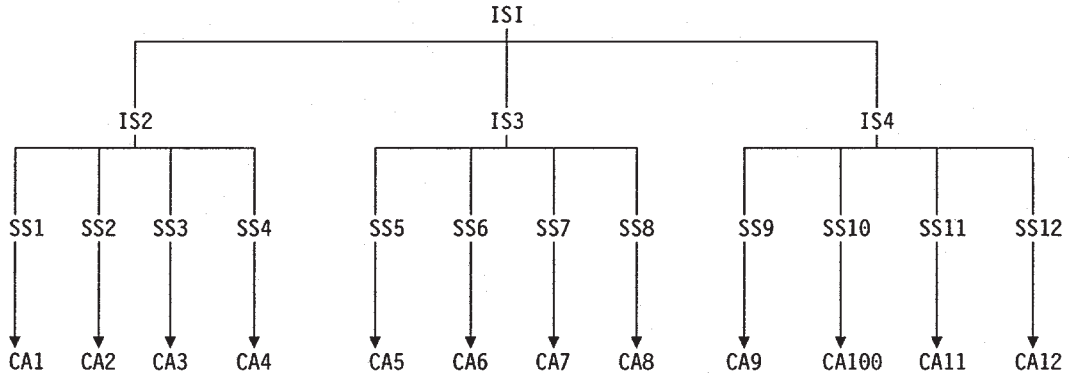
VSAM keeps as many index-set records as the buffer space allows in virtual storage. Ideally, the index would be small enough to permit the entire index set to remain in virtual storage. Because the characteristics of the data set cannot allow a small index, you should be aware of how an index I/O buffers is used to determine how many to provide.

Example of Buffer Allocation for Direct Access

The following example (see Figure 25), demonstrates how buffers are scheduled for direct access. Assume the following:

- Two strings
- Three-level index structure as shown
- Three data buffers (one for each string, and one for splits)
- Four index buffers (one for highest level of index, one for second level, and one for each string)

	Index Buffers				Data Buffers	
Direct Get from CA	Pool		String 1 SS	String 2 SS	String 1	String 2
CA2	IS1	IS2	SS2		CA2-CI	
CA3				SS3		CA3-CI
CA6-1st CI		IS3	SS6		CA6-1st CI	
CA6 1st CI				SS6		CA6-1st CI



IS - Index Set
SS - Sequence Set
CA - Control Area

Figure 25. Scheduling Buffers for Direct Access

The following requests happen.

Request 1. A control interval from CA2 is requested by string 1.

- The highest level index set, IS1, is read into an index buffer. IS1 remains in this buffer for all requests.
- IS1 points to IS2, that is read into a second index buffer.
- IS2 points to the sequence set, SS2, that is read into an index buffer for string 1.
- SS2 points to a control interval in CA2. This control interval is read into a data buffer for string 1.

Request 2. A control interval from CA3 is requested by string 2.

- IS1 and IS2 remain in their respective buffers.
- SS3 is read into an index buffer for string 2.
- SS3 points to a control interval in CA3. This control interval is read into a data buffer for string 2.

Request 3. The first control interval in CA6 is requested by string 1.

- IS1 remains in its buffer.

- Since IS1 now points to IS3, IS3 is read into the second index buffer, replacing IS2.
- SS6 is read into an index buffer for string 1.
- SS6 points to the first control interval in CA6. This control interval is read into a data buffer for string 1.

Request 4. The first control interval in CA6 is now requested by string 2.

- IS1 and IS3 remain in their respective buffers.
- SS6 is read into an index buffer for string 2.
- SS6 points to the first control interval in CA6. This control interval is read into a data buffer for string 2.
- If the string 1 request for this control interval was a GET for update, the control interval would be held in exclusive control, and string 2 would not be able to access it.

Suggested number of buffers for direct processing:

```
Index buffers
   Minimum = STRNO
   Maximum = Number of Index Set Records + STRNO
Data buffers
   STRNO + 1
```

Allocating Buffers for Sequential Access

When you are accessing data sequentially, increase performance by increasing the number of data buffers. When there are multiple data buffers, VSAM uses a read-ahead function to read the next data control intervals into buffers before they are needed. Having only one index I/O buffer does not hinder performance, because VSAM gets to the next control interval by using the horizontal pointers in sequence set records rather than the vertical pointers in the index set. Extra index buffers have little effect during sequential processing.

Suggested number of buffers for initial load mode processing:

```
Index buffers = 3
Data buffers = 2 * (number of Data CI/CA)
```

For straight sequential processing environments, start with four data buffers per string. One buffer is used only for formatting control areas and splitting control intervals and control areas. The other three are used to support the read-ahead function, so that sequential control intervals are placed in buffers before any records from the control interval are requested. By specifying enough data buffers, you can access the same amount of data per I/O operation with small data control intervals as with large data control intervals.

When SHAREOPTIONS 4 is specified for the data set, the read-ahead function can be ineffective because the buffers are refreshed when each control interval is read. Therefore, for SHAREOPTIONS 4, keeping data buffers at a minimum can actually improve performance.

If you experience a performance problem waiting for input from the device, you should specify more data buffers to improve your job's run time. More data buffers let you do more read-ahead processing. An excessive number of buffers, however, can cause performance problems, because of excessive paging.

For mixed processing situations (sequential and direct), start with two data buffers per string and increase BUFND to three per string, if paging is not a problem.

When processing the data set sequentially, VSAM reads ahead as buffers become available. For output processing (PUT for update), VSAM does not immediately write the updated control interval from the buffer unless a control interval split is required. The POINT macro does not cause read-ahead processing unless RPL OPTCD=SEQ is specified; POINT positions the data set for subsequent sequential retrieval.

Suggested number of buffers for sequential access:

```
Index buffers = STRNO
Data buffers = 3 + STRNO (minimum)
```

Allocating Buffers for a Path

Processing data sets using a path can increase the number of buffers that need to be allocated, since buffers are needed for the alternate index, the base cluster, and any alternate indexes in the upgrade set.

The BUFSP, BUFND, BUFNI, and STRNO parameters apply only to the path's alternate index when the base cluster is opened for processing with its alternate index. The minimum number of buffers are allocated to the base cluster unless the cluster's BUFFERSPACE value (specified in the DEFINE command) or BSTRNO value (specified in the ACB macro) permits more buffers. VSAM assumes direct processing and extra buffers are allocated between data and index components accordingly.

Two data buffers and one index buffer are always allocated for each alternate index in the upgrade set. If the path's alternate index is a member of the upgrade set, the minimum buffer increase for each allocation is one for data buffers and one for index buffers. Buffers are allocated to the alternate index as though it were a key-sequenced data set. When a path is opened for output and the path alternate index is in the upgrade set, specify ACB MACRF=DSN and the path alternate index shares buffers with the upgrade alternate index.

Acquiring Buffers

Data and index buffers are acquired and allocated only when the data set is opened. VSAM dynamically allocates buffers based on parameters in effect when the program opens the data set. Parameters that influence the buffer allocation are in the program's ACB: MACRF=(IN|OUT, SEQ|SKP, DIR), STRNO=n, BUFSP=n, BUFND=n, and BUFNI=n. Other parameters that influence buffer allocation are in the DD statement's AMP specification for BUFSP, BUFND, and BUFNI, and the BUFFERSPACE value in the data set's catalog record.

If you open a data set whose ACB includes MACRF=(SEQ,DIR), buffers are allocated according to the rules for sequential processing. If the RPL is modified later in the program, the buffers allocated when the data set was opened do not change.

Data and index buffer allocation (BUFND and BUFNI) can be specified only by the user with access to modify the ACB parameters, or through the AMP parameter of the DD statement. Any program can be assigned additional buffer space by modifying the data set's BUFFERSPACE value, or by specifying a larger BUFSP value with the AMP parameter in the data set's DD statement.

When a buffer's contents are written, the buffer's space is not released. The control interval remains in storage until overwritten with a new control interval; if your program refers to that control interval, VSAM does not have to reread it. VSAM checks to see if the desired control interval is in storage, when your program processes records in a limited key range, you might increase throughput by providing extra data buffers. Buffer space is released when the data set is closed.

Recommendation: Try to have data available just before it is to be used. If data is read into buffers too far ahead of its use in the program, it can be paged out. More data or index buffers than necessary might cause excessive paging or excessive internal processing. There is an optimum point at which more buffers will not help.

Using Index Options

The following options influence performance when using the index of a key-sequenced data set or variable-length RRDS. Each option improves performance, but some require that you provide additional virtual storage or auxiliary storage space. The options are:

- Specifying enough virtual storage to contain all index-set records (if you are using shared resources).
- Ensuring that the index control interval is large enough to contain the key of each control interval in the control area.
- Placing the index and the data set on separate volumes.

Increasing Virtual Storage for Index Set Records

To retrieve a record from a key-sequenced data set or variable-length RRDS, or store a record using keyed access, VSAM needs to examine the index of that data set. Before your processing program begins to process the data set, it must specify the amount of virtual storage it is providing for VSAM to buffer index records. The minimum is enough space for one I/O buffer for index records, but a serious performance problem would occur if an index record were continually deleted from virtual storage to make room for another and then retrieved again later when it is required. Ample buffer space for index records can improve performance.

You ensure virtual storage for index-set records by specifying enough virtual storage for index I/O buffers when you begin to process a key-sequenced data set or variable-length RRDS. VSAM keeps as many index-set records in virtual storage as possible. Whenever an index record must be retrieved to locate a data record, VSAM makes room for it by deleting the index record that VSAM judges to be least useful under the prevailing circumstances. It is generally the index record that belongs to the lowest index level or that has been used the least. VSAM does not keep more than one sequence set index record per string unless shared resource pools are used.

Avoiding Control Area Splits

The second option you might consider is to ensure that the index-set control interval is large enough to contain the key of each control interval in the control area. This reduces the number of control area splits. This option also keeps to a minimum the number of index levels required, thereby reducing search time and improving performance. However, this option can increase rotational delay and data transfer time for the index-set control intervals. It also increases virtual storage requirements for index records.

Putting the Index and Data on Separate Volumes

This information applies to non-SMS-managed volumes. With SMS-managed volumes, SMS selects the volumes. When a key-sequenced data set or variable-length RRDS is defined, the entire index or the high-level index set alone can be placed on a volume separate from the data, either on the same or on a different type of device.

Using different volumes lets VSAM gain access to an index and to data at the same time. Also, the smaller amount of space required for an index makes it economical to use a faster storage device for it.

A performance improvement due to separate volumes generally requires asynchronous processing or multitasking with multiple strings.

Obtaining Diagnostic Information

For information about the generalized trace facility (GTF) and other VSAM diagnostic aids, see *z/OS DFSMSdfp Diagnosis Reference*. The trace is very useful for trying to determine what VSAM is being asked to do.

Migrating from the Mass Storage System

Because MSS is no longer supported, you need to migrate the data off MSS. If you issue the ACQRANGE, CNVTAD, or MNTACQ macros, you receive a return code of 0 and a reason code of 20, that means these macros are no longer supported.

Using Hiperbatch

Hiperbatch is a VSAM extension designed to improve performance in specific situations. It uses the data lookaside facility (DLF) services in MVS to provide an alternate fast path method of making data available to many batch jobs. Through Hiperbatch, applications can take advantage of the performance benefits of MVS without changing existing application programs or the JCL used to run them. For more information about using Hiperbatch, see *MVS Hiperbatch Guide*.

Chapter 12. Processing Control Intervals

This chapter covers the following topics.

Topic	Page
Access to a Control Interval	184
Structure of Control Information	185
User Buffering	190
Improved Control Interval Access	191
Control Blocks in Common (CBIC) Option	192

Control interval access gives you access to the contents of a control interval; keyed access and addressed access give you access to individual data records.

Restriction: You cannot use control interval access to access a compressed data set. The data set can be opened for control interval access to permit VERIFY and VERIFY REFRESH processing only.

With control interval access, you have the option of letting VSAM manage I/O buffers or managing them yourself (user buffering). With keyed and addressed access, VSAM always manages I/O buffers. If you select user buffering, you have the further option of using improved control interval access, which provides faster processing than normal control interval access. With user buffering, only control interval processing is permitted. See “Improved Control Interval Access” on page 191.

Control interval access permits greater flexibility in processing entry-sequenced data sets. The definition of an entry-sequenced data set includes the unchangeability of RBAs and the permanence of records. With control interval access, change the RBAs of records in a control interval and delete records by modifying the RDFs and the CIDF.

When using control interval processing, you are responsible for maintaining alternate indexes. If you have specified keyed or addressed access (ACB MACRF={KEY|ADR},...) and control interval access, then those requests for keyed or addressed access (RPL OPTCD={ KEY|ADR},...) cause VSAM to upgrade the alternate indexes. Those requests specifying control interval access will not upgrade the alternate indexes. You are responsible for upgrading them. Upgrading an alternate index is described in “Maintaining Alternate Indexes” on page 123.

Restriction: You should not update key-sequenced data sets or variable-length RRDSs with control interval access. You cannot use control interval access with compressed format data sets. When you process control intervals, you are responsible for how your processing affects indexes, RDFs, and CIDFs. Bypassing use of the control information in the CIDF and RDFs can make the control interval unusable for record level processing. For instance, key-sequenced data sets depend on the accuracy of their indexes and the RDFs and the CIDF in each control interval.

© Copyright IBM Corp. 1987, 2002

Access to a Control Interval

Control interval access is specified entirely by the ACB MACRF parameter and the RPL (or GENCB) OPTCD parameter. To prepare for opening a data set for control interval access with VSAM managing I/O buffers, specify:

```
ACB      MACRF=(CNV,...),...
```

With NUB (no user buffering) and NCI (normal control interval access), specify in the MACRF parameter that the data set is to be opened for keyed and addressed access, and for control interval access. For example, MACRF=(CNV, KEY, SKP, DIR, SEQ, NUB, NCI, OUT) is a valid combination of subparameters.

You define a particular request for control interval access by coding:

```
RPL      OPTCD=(CNV,...),...
```

Usually, control interval access with no user buffering has the same freedoms and limitations as keyed and addressed access have. Control interval access can be synchronous or asynchronous, can have the contents of a control interval moved to your work area (OPTCD=MVE) or left in VSAM's I/O buffer (OPTCD=LOC), and can be defined by a chain of request parameter lists (except with OPTCD=LOC specified).

Except for ERASE, all the request macros (GET, PUT, POINT, CHECK, and ENDREQ) can be used for normal control interval access. To update the contents of a control interval, you must (with no user buffering) previously have retrieved the contents for update. You cannot alter the contents of a control interval with OPTCD=LOC specified.

Both direct and sequential access can be used with control interval access, but skip sequential access may not. That is, specify OPTCD=(CNV,DIR) or (CNV,SEQ), but not OPTCD=(CNV,SKP).

With sequential access, VSAM takes an EODAD exit when you try to retrieve the control interval whose CIDF is filled with 0s or, if there is no such control interval, when you try to retrieve a control interval beyond the last one. A control interval with such a CIDF contains no data or unused space, and is used to represent the software end-of-file. However, VSAM control interval processing does not prevent you from using a direct GET or a POINT and a sequential GET to retrieve the software end-of-file. The search argument for a direct request with control interval access is the RBA of the control interval whose contents are desired.

The RPL (or GENCB) parameters AREA and AREALEN have the same use for control interval access related to OPTCD=MVE or LOC as they do for keyed and addressed access. With OPTCD=MVE, AREA gives the address of the area into which VSAM moves the contents of a control interval. With OPTCD=LOC, AREA gives the address of the area into which VSAM puts the address of the I/O buffer containing the contents of the control interval.

You can load an entry-sequenced data set with control interval access. If you open an empty entry-sequenced data set, VSAM lets you use only sequential storage. That is, issue only PUTs, with OPTCD=(CNV,SEQ,NUP). PUT with OPTCD=NUP stores information in the next available control interval (at the end of the data set).

You cannot load or extend a data set with improved control interval access. VSAM also prohibits you from extending a fixed-length or variable-length RRDS through normal control interval access.

Update the contents of a control interval in one of two ways:

- Retrieve the contents with OPTCD=UPD and store them back. In this case, the RBA of the control interval is specified during the GET for the control interval.
- Without retrieving the contents, store new contents in the control interval with OPTCD=UPD. (You must specify UBF for user buffering.) Because no GET (or a GET with OPTCD=NUP) precedes the PUT, you have to specify the RBA of the control interval as the argument addressed by the RPL.

Structure of Control Information

With keyed access and addressed access, VSAM maintains the control information in a control interval. With control interval access, you are responsible for that information.

Note: A linear data set has no control information imbedded in the control interval. All of the bytes in the control interval are data bytes; there are no CIDFs or RDFs.

Figure 26 shows the relative positions of data, unused space, and control information in a control interval.

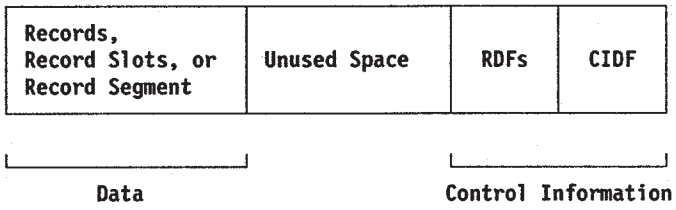


Figure 26. General Format of a Control Interval

For more information about the structure of a control interval, see “Control Intervals” on page 76.

Control information consists of a CIDF (control interval definition field) and, for a control interval containing at least one record, record slot, or record segment, one or more RDFs (record definition fields). The CIDF and RDFs are ordered from right to left. The format of the CIDF is the same even if the control interval size contains multiple smaller physical records.

CIDF—Control Interval Definition Field

The CIDF is a 4-byte field that contains two 2-byte binary numbers.

Offset	Length	Description
0(0)	2	The displacement from the beginning of the control interval to the beginning of the unused space, or, if there is no unused space, to the beginning of the control information. The displacement is equal to the length of the data (records, record slots, or record segment). In a control interval without data, the number is 0.
2(2)	2	The length of the unused space. This number is equal to the length of the control interval, minus the length of the control information, minus the 2-byte value at CIDF+0. In a control interval without data (records, record slots, or record segment), the number is the length of the control interval, minus 4 (the length of the CIDF; there are no RDFs). In a control interval without unused space, the number is 0.
2(2)	1...	Busy flag; set when the control interval is being split; reset when the split is complete.

In an entry-sequenced data set, when there are unused control intervals beyond the last one that contains data, the first of the unused control intervals contains a CIDF filled with 0s. In a key-sequenced data set or an RRDS, the first control interval in the first unused control area (if any) contains a CIDF filled with 0s. A CIDF filled with 0s represents the software end-of-file.

RDF—Record Definition Field

The RBAs of records or relative record numbers of slots in a control interval ascend from left to right. RDFs from right to left describe these records or slots or a segment of a spanned record. RDFs describe records one way for key-sequenced data sets, entry-sequenced data sets, and variable-length RRDSs, and another way for fixed-length RRDSs.

In a key-sequenced or entry-sequenced data set, records might vary in length and can span control intervals. In a variable-length RRDS, records vary in length but do not span control intervals.

- A nonspanned record with no other records of the same length next to it is described by a single RDF that gives the length of the record.
- Two or more consecutive nonspanned records of the same length are described by a pair of RDFs. The RDF on the right gives the length of each record, and the RDF on the left gives the number of consecutive records of the same length.
- Each segment of a spanned record (one segment per control interval) is described by a pair of RDFs. The RDF on the right gives the length of the segment, and the RDF on the left gives its update number. (The update number in each segment is incremented by one each time a spanned record is updated. A difference among update numbers within a spanned record means a possible error in the record.)

In a fixed-length RRDS, records do not vary in length or span control intervals. Each record slot is described by a single RDF that gives its length and indicates if it contains a record.

An RDF is a 3-byte field that contains a 1-byte control field and a 2-byte binary number, as the following table shows.

Offset	Length and Bit Pattern	Description
0(0)	1	Control Field.
	x... ..xx	Reserved.
	.x..	Indicates whether there is (1) or is not (0) a paired RDF to the left of this RDF.
	..xx	Indicates whether the record spans control intervals: **00** No. **01** Yes; this is the first segment. **10** Yes; this is the last segment. **11** Yes; this is an intermediate segment.
	.... x...	Indicates what the 2-byte binary number gives: **0** The length of the record, segment, or slot described by this RDF. **1** The number of consecutive nonspanned records of the same length, or the update number of the segment of a spanned record.
	.... .x..	For a fixed-length RRDS, indicates whether the slot described by this RDF does (0) or does not (1) contain a record.
1(1)	2	Binary number: • When bit 4 of byte 0 is 0, gives the length of the record, segment, or slot described by this RDF. • When bit 4 of byte 0 is 1 and bits 2 and 3 of byte 0 are 0, gives the number of consecutive records of the same length. • When bit 4 of byte 0 is 1 and bits 2 and 3 of byte 0 are not 0, gives the update number of the segment described by this RDF.

Control Field Values for Nonspanned Key-Sequenced, Entry-Sequenced, and Variable-Length Relative Record Data Sets

In a key-sequenced, entry-sequenced data set, or variable-length RRDS with nonspanned records, the possible hexadecimal values in the control field of an RDF follow.

Left RDF	Right RDF	Description
	X'00'	The RDF at X'00' gives the length of a single nonspanned record.
X'08'	X'40'	The right RDF gives the length of each of two or more consecutive nonspanned records of the same length. The left RDF gives the number of consecutive nonspanned records of the same length.

Figure 27 on page 188 shows the contents of the CIDF and RDFs of a 512-byte control interval containing nonspanned records of different lengths.

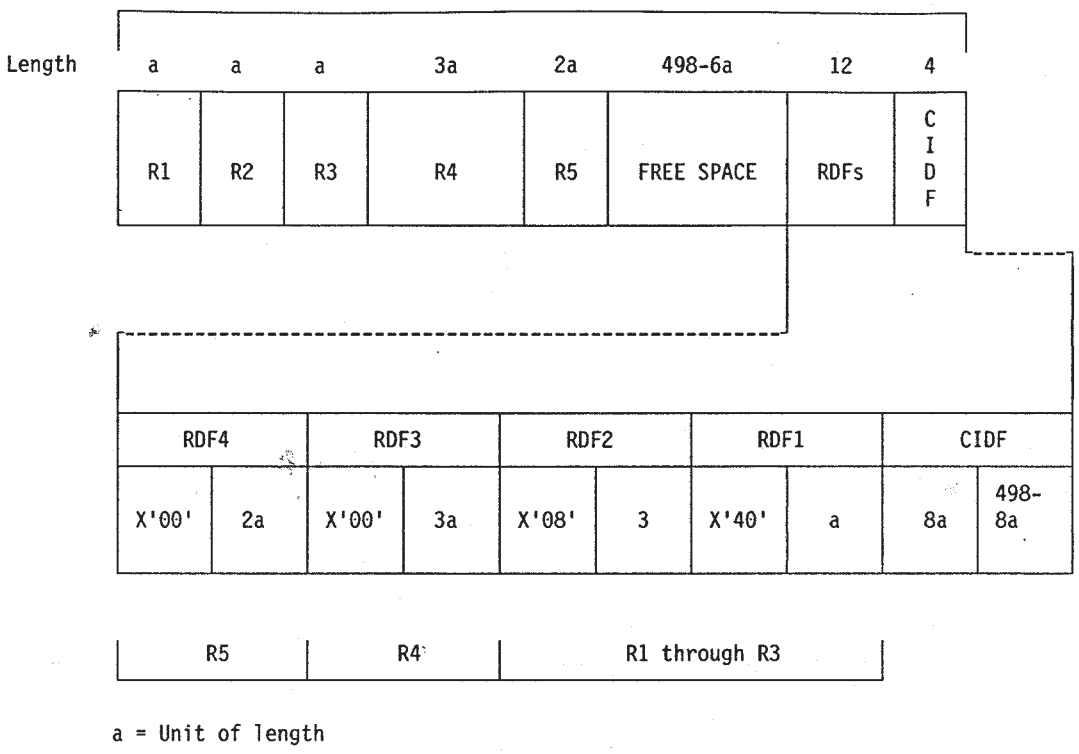


Figure 27. Format of Control Information for Nonspanned Records

The four RDFs and the CIDF comprise 16 bytes of control information as follows:

- RDF4 describes the fifth record.
- RDF3 describes the fourth record.
- RDF2 and RDF1 describe the first three records.
- The first 2-byte field in the CIDF gives the total length of the five records-8a, which is the displacement from the beginning of the control interval to the free space.
- The second 2-byte field gives the length of the free space, which is the length of the control interval minus the total length of the records and the control information-512 minus 8a minus 16, or 496 minus 8a.

Control Field Values for Spanned Key-Sequenced and Entry-Sequenced Data Sets

A control interval that contains the record segment of a spanned record contains no other data; it always has two RDFs. The possible hexadecimal values in their control fields follow.

Left RDF	Right RDF	Description
X'18'	X'50'	The right RDF gives the length of the first segment of a spanned record. The left RDF gives the update number of the segment.
X'28'	X'60'	The right RDF gives the length of the last segment of a spanned record. The left RDF gives the update number of the segment.
X'38'	X'70'	The right RDF gives the length of an intermediate segment of a spanned record. The left RDF gives the update number of the segment.

Figure 28 shows contents of the CIDF and RDFs for a spanned record with a length of 1306 bytes.

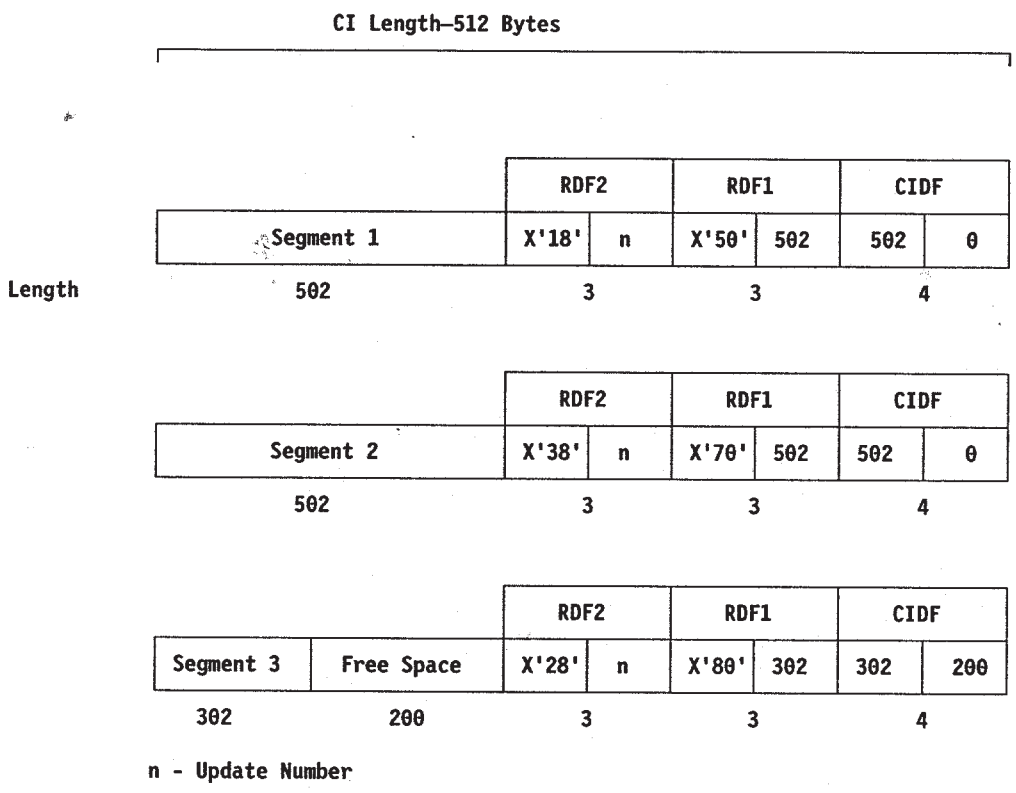


Figure 28. Format of Control Information for Spanned Records

Processing Control Intervals

There are three 512-byte control intervals that contain the segments of the record. The number “n” in RDF2 is the update number. Only the control interval that contains the last segment of a spanned record can have free space. Each of the other segments uses all but the last 10 bytes of a control interval.

In a key-sequenced data set, the control intervals might not be contiguous or in the same order as the segments (for example, the RBA of the second segment can be lower than the RBA of the first segment).

All the segments of a spanned record must be in the same control area. When a control area does not have enough control intervals available for a spanned record, the entire record is stored in a new control area.

Control Field Values for Fixed-Length Relative-Record Data Sets

In a fixed-length RRDS, the possible hexadecimal values in the control field of an RDF are:

X'04' The RDF at X'04' gives the length of an empty slot.

X'00' The RDF at X'00' gives the length of a slot that contains a record.

Every control interval in a fixed-length RRDS contains the same number of slots and the same number of RDFs; one for each slot. The first slot is described by the rightmost RDF. The second slot is described by the next RDF to the left, and so on.

User Buffering

With control interval access, you have the option of user buffering. If you use the user buffering option, you need to provide buffers in your own area of storage for use by VSAM.

User buffering is required for improved control interval access (ICI) and for PUT with OPTCD=NUP.

With ACB MACRF=(CNV,UBF) specified (control interval access with user buffering), the work area specified by the RPL (or GENCB) AREA parameter is, in effect, the I/O buffer. VSAM transmits the contents of a control interval directly between the work area and direct access storage.

If you specify user buffering, you cannot specify KEY or ADR in the MACRF parameter; you can only specify CNV. That is, you cannot intermix keyed and addressed requests with requests for control interval access.

OPTCD=LOC is inconsistent with user buffering and is not permitted.

Improved Control Interval Access

Improved control interval access (ICI) is faster than normal control interval access; however, you can only have one control interval scheduled at a time. Improved control interval access works well for direct processing.

To use ICI, you have to specify user buffering (UBF), which provides the option of specifying improved control interval access:

```
ACB      MACRF=(CNV,UBF,ICI,...),...
```

You cannot load or extend a data set using ICI. Improved control interval processing is not permitted for extended format data sets.

A processing program can achieve the best performance with improved control interval access by combining it with SRB dispatching. SRB dispatching is described in *z/OS MVS Programming: Authorized Assembler Services Guide* and “Operating in SRB or Cross-Memory Mode” on page 155.

Opening an Object for Improved Control Interval Access

Improved control interval processing is faster because functions have been removed from the path. However, improved control interval processing causes several restrictions:

- The object must not be empty.
- The object must not be compressed.
- The object must be one of the following:
  - An entry-sequenced, fixed-length, or variable-length RRDS cluster.
  - The data component of an entry-sequenced, key-sequenced, linear, fixed-length, or variable-length RRDS cluster.
  - The index component of a key-sequenced cluster (index records must not be replicated).
- Control intervals must be the same size as physical records. When you use the access method services DEFINE command to define the object, specify control interval size equal to a physical record size used for the device on which the object is stored. VSAM uses physical record sizes of (n x 512) and (n x 2048), where n is a positive integer from 1 to 16. The physical record size is always equal to the control interval size for an index component.

Processing a Data Set with Improved Control Interval Access

To process a data set with improved control interval access, a request must be:

- Defined by a single RPL (VSAM ignores the NXTRPL parameter).
- A direct GET, GET for update, or PUT for update (no POINT, no processing empty data sets). A RRDS with slots formatted is considered not to be empty, even if no slot contains a record.
- Synchronous (no CHECK, no ENDREQ).

To release exclusive control after a GET for update, you must issue a PUT for update, a GET without update, or a GET for update for a different control interval.

Processing Control Intervals

With improved control interval access, the following assumptions are in effect for VSAM (with no checking):

- An RPL whose ACB has MACRF=ICI has OPTCD=(CNV, DIR, SYN).
- A PUT is for update (RPL OPTCD=UPD).
- Your buffer length (specified in RPL AREALEN=number) is correct.

Because VSAM does not check these parameters, you should debug your program with ACB MACRF=NCI, then change to ICI.

With improved control interval access, VSAM does not take JRNAD exits and does not keep statistics (which are normally available through SHOWCB).

Fixing Control Blocks and Buffers in Real Storage

With improved control interval access, you can specify that control blocks are to be fixed in real storage (ACB MACRF=(CFX,...)). If you so specify, your I/O buffers must also be fixed in real storage. Having your control blocks fixed in real storage, but not your I/O buffers, can cause physical errors or unpredictable results. If you specify MACRF=CFX without ICI, VSAM ignores CFX. NFX is the default; it indicates that buffers are not fixed in real storage, except for an I/O operation. A program must be authorized to fix pages in real storage, either in supervisor state with protection key 0 - 7, or link-edited with authorization. (The authorized program facility (APF) is described in *z/OS MVS Programming: Authorized Assembler Services Guide*). An unauthorized request is ignored.

You can use 64-bit real storage for all VSAM data sets, whether or not they are extended-format data sets. You can obtain buffer storage from any real address location available to the processor. The location can have a real address greater than 2 gigabytes or can be in 31-bit real storage with a real address less than 2 gigabytes.

Control Blocks in Common (CBIC) Option

When you are using improved control interval processing, the CBIC option lets you have multiple address spaces that address the same data and use the same control block structure. The VSAM control blocks associated with a VSAM data set are placed into the common service area (CSA). The control block structure and VSAM I/O operations are essentially the same whether the CBIC option is invoked, except for the location of the control block structure. The user-related control blocks are generated in the protect key (0 - 7). The system-related control blocks are generated in protect key 0. The VSAM control block structure generated when the CBIC option is invoked retains normal interfaces to the address space that opened the VSAM data set (for example, the DEB is chained to the address space's TCB).

The CBIC option is invoked when a VSAM data set is opened. To invoke the CBIC option, you set the CBIC flag (located at offset X'33' (ACBINFL2) in the ACB, bit 2 (ACBCBIC)) to one. When your program opens the ACB with the CBIC option set, your program must be in supervisor state with a protect key from 0 to 7. Otherwise, VSAM will not open the data set.

The following restrictions apply to using the CBIC option:

- The CBIC option must be used only when the ICI option is also specified.
- You cannot also specify LSR or GSR.
- You cannot use the following types of data sets with the CBIC option: catalogs, catalog recovery areas, swap data sets, or system data sets.
- If an address space has opened a VSAM data set with the CBIC option, your program cannot take a checkpoint for that address space.

If another address space accesses the data set's control block structure in the CSA through VSAM record management, the following conditions should be observed:

- An OPEN macro should not be issued against the data set.
- The ACB of the user who opened the data set with the CBIC option must be used.
- CLOSE and temporary CLOSE cannot be issued for the data set (only the user who opened the data set with the CBIC option can close the data set).
- The address space accessing the data set control block structure must have the same storage protect key as the user who opened the data set with the CBIC option.
- User exit routines should be accessible from all regions accessing the data set with the CBIC option.

Chapter 13. Sharing VSAM Data Sets

This chapter explains how to share data sets within a single system and among multiple systems. It also describes considerations for sharing VSAM data sets for NSR or LSR/GSR access. For considerations about sharing VSAM data sets for RLS access, see “Chapter 15. Using VSAM Record-Level Sharing” on page 223.

This chapter covers the following topics.

Topic	Page
Subtask Sharing	196
Cross-Region Sharing	201
Cross-System Sharing	204
Control Block Update Facility (CBUF)	205
Techniques of Data Sharing	208

You can share data sets between:
- Different jobs in a single operating system
- Multiple ACBs in a task or different subtasks
- One ACB in a task or different subtasks
- Different operating systems. To share between different operating systems safely, you need global resource serialization or an equivalent product. Failure to use GRS or an equivalent can result in both data set and VTOC corruption. See *z/OS MVS Planning: Global Resource Serialization*.

When you define VSAM data sets, you can specify how the data is to be shared within a single system or among multiple systems that can have access to your data and share the same direct access devices. Before you define the level of sharing for a data set, you must evaluate the consequences of reading incorrect data (a loss of read integrity) and writing incorrect data (a loss of write integrity)—situations can result when one or more of the data set’s users do not adhere to guidelines recommended for accessing shared data sets.

The extent to which you want your data sets to be shared depends on the application. If your requirements are similar to those of a catalog, where there can be many users on more than one system, more than one user should be permitted to read and update the data set simultaneously. At the other end of the spectrum is an application where high security and data integrity require that only one user at a time have access to the data.

When your program issues a GET request, VSAM reads an entire control interval into virtual storage (or obtains a copy of the data from a control interval already in virtual storage). If your program modifies the control interval’s data, VSAM ensures within a single control block structure that you have exclusive use of the information in the control interval until it is written back to the data set. If the data set is accessed by more than one program at a time, and more than one control block structure contains buffers for the data set’s control intervals, VSAM cannot ensure that your program has exclusive use of the data. You must obtain exclusive control yourself, using facilities such as ENQ/RESERVE and DEQ.

Two ways to establish the extent of data set sharing are the data set disposition specified in the JCL and the share options specified in the access method services

© Copyright IBM Corp. 1987, 2002

DEFINE or ALTER command. If the VSAM data set cannot be shared because of the disposition specified in the JCL, a scheduler allocation failure occurs. If your program attempts to open a data set that is in use and the share options specified do not permit concurrent use of the data, the open fails, and a return code is set in the ACB error field.

During load mode processing, you cannot share data sets. Share options are overridden during load mode processing. When a shared data set is opened for create or reset processing, your program has exclusive control of the data set within your operating system.

You can use ENQ/DEQ to issue VSAM requests, but not to serialize the system resources that VSAM uses.

Subtask Sharing

Subtask sharing is the ability to perform multiple OPENs to the same data set within a task or from different subtasks in a single address space and still share a single control block structure. Subtask sharing allows many logical views of the data set while maintaining a single control block structure. With a single control block structure, you can ensure that you have exclusive control of the buffer when updating a data set.

If you share multiple control block structures within a task or address space, VSAM treats this like cross-address space sharing. You must adhere to the guidelines and restrictions specified in “Cross-Region Sharing” on page 201.

Building a Single Control Block Structure

To share successfully within a task or between subtasks, you should ensure that VSAM builds a single control block structure for the data set. This control block structure includes blocks for control information and input/output buffers. All subtasks access the data set through this single control block structure, independent of the SHAREOPTION or DISP specifications. The three methods of achieving a single control block structure for a VSAM data set while processing multiple concurrent requests are:

- A single access method control block (ACB) and a STRNO>1
- Data definition name (ddname) sharing, with multiple ACBs (all from the same data set) pointing to a single DD statement. This is the default. For example:

```
//DD1   DD   DSN=ABC

       OPEN ACB1,DDN=DD1
       OPEN ACB2,DDN=DD1
```

- Data set name sharing, with multiple ACBs pointing to multiple DD statements with different ddnames. The data set names are related with an ACB open specification (MACRF=DSN). For example:

```
//DD1   DD   DSN=ABC
//DD2   DD   DSN=ABC

       OPEN ACB1,DDN=DD1,MACRF=DSN
       OPEN ACB2,DDN=DD2,MACRF=DSN
```

Multiple ACBs must be in the same address space, and they must be opening to the same base cluster. The connection occurs independently of the path selected to the base cluster. If the ATTACH macro is used to create a new task that will be processing a shared data set, let the ATTACH keyword SZERO to default to YES or code SZERO=YES. This causes subpool 0 to be shared with the subtasks. For more information about the ATTACH macro see *z/OS MVS Programming: Authorized Assembler Services Reference ALE-DYN*. This also applies to when you are sharing one ACB in a task or different subtasks. To ensure correct processing in the shared environment, all VSAM requests should be issued in the same key as the job step TCB key.

Resolving Exclusive Control Conflicts

In this environment with a single control block, VSAM record management serializes updates to any single control interval and provides read and write integrity. When a control interval is not available for the type of user processing requested (shared or exclusive), VSAM record management returns a logical error code with an exclusive control error indicated in the RPL feedback code. When this occurs, you must decide whether to retry later or to free the resource causing the conflict. See Figure 29 on page 198 for a diagram of exclusive control conflict feedback and results of different user requests.

NONSHARED RESOURCES (NSR)

User B Has:

User A Wants:		EXCLUSIVE CONTROL	SHARED
	EXCLUSIVE CONTROL	User A gets logical error	OK User A gets second copy of buffer
	SHARED	OK User A gets second copy of buffer	OK User A gets second copy of buffer

SHARED RESOURCES (LSR/GSR)

User B has:

User A Wants:		EXCLUSIVE CONTROL	SHARED
	EXCLUSIVE CONTROL	User A gets logical error	VSAM queues user A until buffer is released by user B or user A gets logical error via ACB option
	SHARED	User A gets logical error	OK User A shares same buffer with user B

Figure 29. Exclusive Control Conflict Resolution

By default, if an exclusive control conflict is encountered, VSAM defers the request until the resource becomes available. You can change this by using the VSAM avoid LSR error control wait function. By using the NLW subparameter of the MACRF parameter for the ACB macro, instead of deferring the request, VSAM returns the exclusive control return code 20 (X'14') to the application program. The application program can then determine the next action.

Alternatively, you can do this by changing the GENCB ACB macro in the application program.

To test to see if the new function is in effect, the TESTCB ACB macro can be coded into the application program.

The application program must be altered to handle the exclusive control error return code. Register 15 will contain 8 and the RPLERRCD field will contain 20 (X'14').

The address of the RPL that owns the resource is placed in the first word in the RPL error message area. The VSAM avoid LSR exclusive control wait option cannot be changed after OPEN.

Preventing Deadlock in Exclusive Control of Shared Resources

Contention for VSAM data (the contents of a control interval) can lead to deadlocks, in which a processing program is prevented from continuing because its request for data cannot be satisfied. A and B can engage as contenders in four distinct ways:

1. A wants exclusive control, but B has exclusive control. VSAM refuses A's request: A must either do without the data or retry the request.
2. A wants exclusive control, but B is only willing to share. VSAM queues A's request (without notifying A of a wait) and gives A use of the data when B releases it.
3. A wants to share, but B has exclusive control. VSAM refuses A's request: A must either do without the data or retry the request.
4. A wants to share, and B is willing to share. VSAM gives A use of the data, along with B.

VSAM's action in a contention for data rests on two assumptions:

- If a processing program has exclusive control of the data, it can update or delete it.
- If a processing program is updating or deleting the data, it has exclusive control. (The use of MRKBFR, MARK=OUT provides an exception to this assumption. A processing program can update the contents of a control interval without exclusive control of them.)

In 1 and 3 above, B is responsible for giving up exclusive control of a control interval through an ENDREQ, a MRKBFR with MARK=RLS, or a request for access to a different control interval. (The RPL that defines the ENDREQ, MRKBFR, or request is the one used to acquire exclusive control originally.)

Data Set Name Sharing

Data set name sharing is established by the ACB option (MACRF=DSN). To understand DSN sharing, you must understand a sphere and the base of the sphere and how they function.

Spheres. A sphere is a VSAM cluster and its associated data sets. The cluster is originally defined with the access method services ALLOCATE command, the DEFINE CLUSTER command, or through JCL. The most common use of the sphere is to open a single cluster. The base of the sphere is the cluster itself. When opening a path (which is the relationship between an alternate index and base cluster) the base of the sphere is again the base cluster. Opening the alternate index as a data set results in the alternate index becoming the base of the sphere. In Figure 30 on page 200, DSN is specified for each ACB, and output processing is specified.

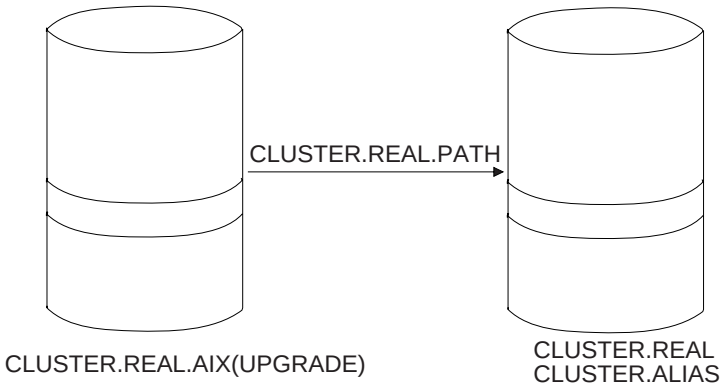


Figure 30. Relationship Between the Base Cluster and the Alternate Index

Connected Spheres. VSAM connects an ACB to an existing control block structure for data set name sharing only when the base of the sphere is the same for both ACBs. The following three OPEN statements show how information is added to a single control block structure, permitting data set name sharing.

1. `OPEN ACB=(CLUSTER.REAL)`
   - Builds control block structure for `CLUSTER.REAL`
   - Builds control block structure for `CLUSTER.REAL.AIX`
2. `OPEN ACB=(CLUSTER.REAL.PATH)`
   - Adds to existing structure for `CLUSTER.REAL`
   - Adds to existing structure for `CLUSTER.REAL.AIX`
3. `OPEN ACB=(CLUSTER.ALIAS)`

   Adds to existing structure for `CLUSTER.REAL`

If you add a fourth statement, the base of the sphere changes, and multiple control block structures are created for the alternate index `CLUSTER.REAL.AIX`:

4. `OPEN ACB=(CLUSTER.REAL.AIX)`

- Does not add to existing structure as the base of the sphere is not the same.
- SHAREOPTIONS are enforced for `CLUSTER.REAL.AIX` since multiple control block structures exist.

Consistent Processing Options

To be compatible, both the new ACB and the existing control block structure must be consistent in their specification of the following processing options.

- The data set specification must be consistent in both the ACB and the existing control block structure. This means that an index of a key-sequenced data set that is opened as an entry-sequenced data set, does not share the same control block structure as the key-sequenced data set opened as a key-sequenced data set.
- The MACRF options DFR, UBF, ICI, CBIC, LSR, and GSR must be consistent. For example, if the new ACB and the existing structure both specify MACRF=DFR, the connection is made. If the new ACB specifies MACRF=DFR and the existing structure specifies MACRF=DFR,UBF, no connection is made.

If compatibility cannot be established, OPEN tries (within the limitations of the share options specified when the data set was defined) to build a new control block structure. If it cannot, OPEN fails.

Shared Subtasks

When processing multiple subtasks sharing a single control block, concurrent GET and PUT requests are allowed. A control interval is protected for write operations using an exclusive control facility provided in VSAM record management. Other PUT requests to the same control interval are not allowed and a logical error is returned to the user issuing the request macro. Depending on the selected buffer option, nonshared (NSR) or shared (LSR/GSR) resources, GET requests to the same control interval as that being updated can or cannot be allowed. Figure 29 on page 198 illustrates the exclusive control facility.

When a subtask issues OPEN to an ACB that will share a control block structure that can have been previously used, issue the POINT macro to obtain the position for the data set. In this case, it should not be assumed that positioning is at the beginning of the data set.

Cross-Region Sharing

The extent of data set sharing within one operating system depends on the data set disposition and the cross-region share option specified when you define the data set. Independent job steps or subtasks in an MVS system or multiple systems with global resource serialization (GRS) can access a VSAM data set simultaneously. For more information about GRS see *z/OS MVS Planning: Global Resource Serialization*. To share a data set, each user must specify DISP=SHR in the data set's DD statement.

Cross-Region Share Options

The level of cross-region sharing permitted by VSAM is established (when the data set is defined) with the SHAREOPTIONS value:

- Cross-region SHAREOPTIONS 1: The data set can be shared by any number of users for read processing, or the data set can be accessed by only one user for read and write processing. With this option, VSAM ensures complete data integrity for the data set. This setting does not permit any type of non-RLS access when the data set is already open for RLS processing.
- Cross-region SHAREOPTIONS 2: If the data set has not already been opened for record-level sharing (RLS) processing, the data set can be accessed by any number of non-RLS users for read processing and it can also be accessed by one non-RLS user for write processing. With this option, VSAM ensures write integrity by obtaining exclusive control for a control interval when it is to be updated.

  If the data set has already been opened for RLS processing, non-RLS accesses for read are allowed. VSAM provides full read and write integrity to its RLS users, but it is the non-RLS user's responsibility to ensure read integrity.

  If you require read integrity, it is your responsibility to use the ENQ and DEQ macros appropriately to provide read integrity for the data the program obtains. For information about using ENQ and DEQ see *z/OS MVS Programming: Authorized Assembler Services Reference ALE-DYN* and *z/OS MVS Programming: Authorized Assembler Services Reference ENF-IXG*.
- Cross-region SHAREOPTIONS 3: The data set can be fully shared by any number of users. With this option, each user is responsible for maintaining both read and write integrity for the data the program accesses. This setting does not allow any type of non-RLS access when the data set is already open for RLS processing.

  This option requires that the user's program use ENQ/DEQ to maintain data integrity while sharing the data set, including the OPEN and CLOSE processing.

User programs that ignore the write integrity guidelines can cause VSAM program checks, lost or inaccessible records, uncorrectable data set failures, and other unpredictable results. This option places responsibility on each user sharing the data set.

- Cross-region SHAREOPTIONS 4: The data set can be fully shared by any number of users, and buffers used for direct processing are refreshed for each request. This setting does not allow any type of non-RLS access when the data set is already open for RLS processing. With this option, as in SHAREOPTIONS 3, each user is responsible for maintaining both read and write integrity for the data the program accesses. See the description of SHAREOPTIONS 3 for ENQ/DEQ and warning information that applies equally to SHAREOPTIONS 4.

With options 3 and 4 you are responsible for maintaining both read and write integrity for the data the program accesses. These options require your program to use ENQ/DEQ to maintain data integrity while sharing the data set, including the OPEN and CLOSE processing. User programs that ignore the write integrity guidelines can cause VSAM program checks, lost or inaccessible records, uncorrectable data set failures, and other unpredictable results. These options place heavy responsibility on each user sharing the data set.

When your program requires that no updating from another control block structure occur before it completes processing of the requested data record, your program can issue an ENQ to obtain exclusive use of the VSAM data set. If your program completes processing, it can relinquish control of the data set with a DEQ. If your program is only reading data and not updating, it is probably a good practice to serialize the updates and have the readers wait while the update is occurring. If your program is updating, after the update has completed the ENQ/DEQ bracket, the reader must determine the required operations for control block refresh and buffer invalidation based on a communication mechanism or assume that everything is down-level and refresh each request.

The extent of cross-region sharing is affected by using DISP=SHR or DISP=OLD in the DD statement. If the data set's DD statement specifies DISP=OLD, only the dsname associated with the DD statement is exclusively controlled. In this case, only the cluster name is reserved for the OPEN routine's exclusive use. You can include DD statements with DISP=OLD for each of the cluster's components to reserve them as well. Doing this ensures that all resources needed to open the data set will be exclusively reserved before your task is initiated.

Protecting the cluster name with DISP processing and the components by VSAM OPEN SHAREOPTIONS is the normally accepted procedure. When a shared data set is opened with DISP=OLD, or is opened for create or reset processing, your program has exclusive control of the data set within your operating system.

Scheduler disposition processing is the same for VSAM and non-VSAM data sets. This is the first level of share protection.

Read Integrity During Cross-Region Sharing

You are responsible for ensuring read integrity when the data set is opened for sharing with cross-region SHAREOPTIONS 2, 3, and 4. When your program issues a GET request, VSAM obtains a copy of the control interval containing the requested data record. Another program sharing the data set can also obtain a copy of the same control interval, and can update the data and write the control interval back into the data set. When this occurs, your program has lost read integrity. The control interval copy in your program's buffer is no longer the current copy.

The following should be considered when you are providing read integrity:

- Establish ENQ/DEQ procedures for all requests, read and write.
- Decide how to determine and invalidate buffers (index and/or data) that are possibly down-level.
- Do not permit secondary allocation for an entry-sequenced data set or for a fixed-length or variable-length RRDS. If you do allow secondary allocation you should provide a communication mechanism to the read-only tasks that the extents are increased, force a CLOSE, and then issue another OPEN. Providing a buffer refresh mechanism for index I/O will accommodate secondary allocations for a key-sequenced data set.
- With an entry-sequenced data set or a fixed-length or variable-length RRDS, you must also use the VERIFY macro before the GET macro to update possible down-level control blocks.
- Generally, the loss of read integrity results in down-level data records and erroneous no-record-found conditions.

Invalidating Index Buffers

To invalidate index buffers, you could perform the following steps:

1. In the ACB, specify:
   STRNO>1.
   MACRF=NSR to indicate nonshared resources.
   Let the value for BUFNI default to the minimum.
2. Ensure that your index is a multilevel index.
3. Ensure that all requests are for positioning by specifying the following:
   GET RPL OPTCD=DIR
   POINT
   PUT RPL OPTCD=NUP

Invalidating Data Buffers

To invalidate data buffers, ensure that all requests are for positioning by specifying one of the following:

- GET/PUT RPL OPTCD=(DIR,NSP) followed by ENDREQ
- POINT GET/PUT RPL OPTCD=SEQ followed by ENDREQ

Write Integrity During Cross-Region Sharing

You are responsible for ensuring write integrity if a data set is opened with cross-region SHAREOPTIONS 3 or 4.

When an application program issues a “direct” or “skip-sequential” PUT-for-update or no-update, (RPL OPTCD=DIR|SKP), the updated control interval is written to direct access storage when you obtain control following a synchronous request (RPL OPTCD=SYN) or following the CHECK macro from an asynchronous request (RPL OPTCD=ASY). To force direct access I/O for a sequential PUT (RPL OPTCD=SEQ), the application program must issue an ENDREQ or MRKBFR TYPE=OUT.

Whenever an ENDREQ is issued, the return code in register 15 should be checked to determine if there is an error. If there is an error, normal check processing should be performed to complete the request.

The considerations that apply to read integrity also apply to write integrity. The serialization for read could be done as a shared ENQ and for write as an exclusive ENQ. You must ensure that all I/O is performed to DASD before dropping the serialization mechanism (usually the DEQ).

Cross-System Sharing

Use either of the following share options when you define a data set that must be accessed or updated by more than one operating system simultaneously:

- **Cross-system SHAREOPTION 3.** The data set can be fully shared. With this option, the access method uses the control block update facility (CBUF) to help. With this option, as in cross-region SHAREOPTIONS 3, each user is responsible for maintaining both read and write integrity for the data the program accesses. User programs that ignore write integrity guidelines can cause VSAM program checks, uncorrectable data set failures, and other unpredictable results. This option places heavy responsibility on each user sharing the data set. The RESERVE and DEQ macros are required with this option to maintain data set integrity.
- **Cross-system SHAREOPTION 4.** The data set can be fully shared, and buffers used for direct processing are refreshed for each request.

  This option requires that you use the RESERVE and DEQ macros to maintain data integrity while sharing the data set. Output processing is limited to update and/or add processing that does not change either the high-used RBA or the RBA of the high key data control interval if DISP=SHR is specified. For information about using RESERVE and DEQ, see *z/OS MVS Programming: Authorized Assembler Services Reference ALE-DYN* and *z/OS MVS Programming: Authorized Assembler Services Reference LLA-SDU*.

System-managed volumes and catalogs that contain system-managed data sets must not be shared with non-system-managed systems. When sharing data sets in a cross-region or cross-system environment, run the VERIFY macro before you open a data set. VERIFY locates the current end of the data set and updates internal control blocks. When the data set closes successfully, the system updates the catalog with the information that VERIFY determined. This information and its effects cannot be evident to all systems sharing the data set. If run as the first step of a job stream, VERIFY causes an update to the end-of-file information in the catalog.

To ensure data integrity in a shared environment, VSAM provides users of SHAREOPTIONS 4 (both cross-region and cross-system) with the following assistance:

- Each PUT request results in the appropriate buffer being written immediately into the VSAM object's direct access device space. VSAM writes out the buffer in the user's address space that contains the new or updated data record.
- Each GET request results in all the user's input buffers being refreshed. The contents of each data and index buffer used by the user's program is retrieved from the VSAM object's direct access device.

When the data set is shared under cross-system SHAREOPTIONS 4, regardless of cross-region requests, VSAM does not allow changes to high-used and high-key RBAs. In addition, VSAM provides assistance to the application to aid in preserving the integrity of the data:

- Control area splits and the addition of a new high-key record for a new control interval that results from a control interval split are not allowed; VSAM returns a logical error to the user's program if this condition should occur.
- The data and sequence-set control interval buffers are marked invalid following I/O operation to a direct access storage device.

Job steps of two or more systems can gain access to the same data set regardless of the disposition specified in each step's JCL. To get exclusive control of a volume, a task in one system must issue a RESERVE macro. For other methods of obtaining exclusive control using global resource serialization (GRS) see *z/OS DFSMS Migration* and *z/OS MVS Planning: Global Resource Serialization*.

Control Block Update Facility (CBUF)

Whenever a data set is opened with DISP=SHR, cross-region SHAREOPTION 3 or 4, and cross-system SHAREOPTION 3, VSAM record management maintains a copy of the critical control block data in common storage. The control block data in the common storage area is available to each program (each memory) sharing the data set. The common storage area is available only to regions within your operating system. Communicating this information to another operating system is your responsibility.

CBUF eliminates the restriction that prohibits control area splits under cross-region SHAREOPTION 4. Therefore, you do not need to restrict code to prevent control area splits, or allow for the control area split error condition. The restriction to prohibit control area splits for cross-systems SHAREOPTION 4 still exists.

CBUF processing is not provided if the data set has cross-system SHAREOPTION 4, but does not reside on shared DASD when it is opened. That is, the data set is still processed as a cross-system SHAREOPTION 4 data set on shared DASD. When a key-sequenced data set or variable-length RRDS has cross-system SHAREOPTION 4, control area splits are prevented. Also, split of the control interval containing the high key of a key range (or data set) is prevented. With control interval access, adding a new control interval is prevented.

Cross-system sharing can be accomplished by sending the VSAM shared information (VSI) blocks to the other host at the conclusion of each output request. Generally, the VSIs will not have changed and only a check occurs.

If you use SHAREOPTION 3, you must continue to provide read/write integrity. Although VSAM ensures that SHAREOPTION 3 and 4 users will have correct control block information if serialization is done correctly, the SHAREOPTION 3 user will not get the buffer invalidation that will occur with SHAREOPTION 4.

When improved control interval processing is specified with SHAREOPTION 3 or 4, the data set can be opened. However, if another control block structure extends the data set, the control block structure using improved control interval processing will not be updated unless it is closed and reopened.

Table 10 on page 206 shows how the SHAREOPTIONS specified in the catalog and the disposition specified on the DD statement interact to affect the type of processing.

Table 10. Relationship between SHAREOPTIONS and VSAM Functions

(CR CS) when DISP=SHR[1]	Functions Provided
(3 3)	CBUF
(3 4)	Data and sequence set buffers invalidated. CA split not allowed.
(4 3)	Data and index component buffers invalidated. CBUF.
(4 4)	Data and sequence set buffers invalidated. CA split not allowed.

Legend:
CA = Control area
CR = Cross-region
CS = Cross-system
CBUF = Control block update facility
Buffer invalidated = Invalidation of buffers is automatic

Note:

1. When DISP=OLD is specified or the data set is in create or reset mode (regardless of the disposition specified), the share options specified in the catalog are ignored. The data set is processed under the rules for SHAREOPTIONS(1 3). OPEN ensures that the user has exclusive control of the data set within a single system. If the data set can be shared between systems, VSAM does nothing to ensure that another system is not accessing the data set concurrently. With cross-system sharing, the user must ensure that another system is not accessing the data set before specifying DISP=OLD.

Considerations for CBUF Processing

If your program shares a data set defined with SHAREOPTIONS(3 3) or SHAREOPTIONS(4 3), you should note that:

- In a shared environment, VSAM does not allow you to process the data set in an initial load or reset mode (create). VSAM forces your data set to be processed as though it were defined with SHAREOPTIONS(1 3).
- A user program cannot share a system data set (for example, the master catalog, page space data sets, SYS1. data sets, duplex data sets, and swap data sets).
- The user's program must serialize all VSAM requests against the data set, using ENQ/DEQ (or a similar function).
- The user's program must insure that all VSAM resources are acquired and released within ENQ/DEQ protocol to:
  - Force VSAM to write sequential update and insert requests.
  - Release VSAM's positioning within the data set.
- VSAM invalidates data and index buffers used with cross-region or cross-system SHAREOPTIONS 4 data sets, but does not invalidate buffers used with SHAREOPTIONS 3 data sets. When a buffer is marked invalid (it is invalidated), it is identified as a buffer that VSAM must refresh (read in a fresh copy of the control interval from DASD) before your program can use the buffer's contents.
- Programs that use GSR and LSR can invalidate and force writing of buffers using the MRKBFR and WRTBFR macros.
- Because programs in many regions can share the same data set, an error that occurs in one region can affect programs in other regions that share the same data set. If a logical error (register 15=8) or physical error (register 15=12) is detected, any control block changes made before the error was detected will be propagated to the shared information in common storage.

- When a VSAM data set requires additional space, VSAM end-of-volume processing acquires new extents for the data set, updates the VSAM control block structure for the data set with the new extent information, and updates the critical control block data in common storage so that this new space is accessible by all regions that use the data set. If the occurrence of an abend or unexpected error prevents this space allocation from being completed, all regions are prevented from further extending the data set. To obtain additional space, you must close the VSAM data set in all regions, then reopen it.
- To correct the control blocks of a data set after an abnormal termination (abend), issue the VERIFY macro to update them. VERIFY does not modify the data set. You must determine what recovery action is required, if any. A subsequent CLOSE updates the catalog record. The system bypasses the update to the data set's catalog record after an abnormal termination.
- Implicit VERIFY is invoked by the open-for-output indicator in the catalog. When a data set is opened and the open-for-output indicator is already on, CLOSE processing resets the indicator only if the data set was just opened for output; otherwise it leaves the bit on.
- Data sets shared in a cross-region or cross-system environment should either use the access method services VERIFY command or issue the VERIFY macro from within the application program.
- Because programs in many regions can share the same data set, an error in one region can affect programs in other regions that share the data set. If a logical error (register 15=8) or physical error (register 15=12) occurs, control block changes made before the error was detected propagate to the shared information control block in common storage. When this condition occurs, that data set can place incorrect information in the catalog. Check the affected data set by using the appropriate diagnostic tool (such as EXAMINE) to determine if the data set has been corrupted. If the data set is damaged, use an appropriate utility, such as REPRO, to recover the data set.

Checkpoints for Shared Data Sets

If you issue a checkpoint or if a restart occurs, none of the VSAM data sets open in your region at that time can be using CBUF processing. If you issue checkpoints, you should open the VSAM data sets that are eligible for CBUF processing with a disposition of OLD, or CLOSE them before the checkpoint. Note that, if an alternate index was using CBUF processing, the associated base cluster and any other paths open over that base cluster must also be closed before the checkpoint, even if they are not using CBUF processing.

Techniques of Data Sharing

This section describes the different techniques of data sharing.

Cross-Region Sharing

To maintain write integrity for the data set, your program must ensure that there is no conflicting activity against the data set until your program completes updating the control interval. Conflicting activity can be divided into two categories:

1. A data set that is totally preformatted and the only write activity is update-in-place.

   In this case, the sharing problem is simplified by the fact that data cannot change its position in the data set. The lock that must be held for any write operation (`GET/PUT RPL OPTCD=UPD`) is the unit of transfer that is the control interval. It is your responsibility to associate a lock with this unit of transfer; the record key is not sufficient unless only a single logical record resides in a control interval.

   The following is an example of the required procedures:

   a. Issue a GET for the RPL that has the parameters `OPTCD=(SYN,KEY,UPD,DIR),ARG=MYKEY`.

   b. Determine the RBA of the control interval (`RELCI`) where the record resides. This is based on the RBA field supplied in the `RPL(RPLDDDD)`.

      ```
      RELCI=CISIZE * integer-part-of (RPLDDDD / CISIZE)
      ```

   c. Enqueue `MYDATA.DSNAME.RELCI` (the calculated value).

   d. Issue an ENDREQ.

   e. Issue a GET for the RPL that has the parameters `OPTCD=(SYN,KEY,UPD,DIR),ARG=MYKEY`. This action will do I/O and get a refreshed copy of the buffer.

   f. Determine the RBA of the control interval (`RELCI`) where the record resides. This is based on the RBA field supplied in the `RPL(RPLDDDD)`.

      ```
      RELCI = CISIZE * integer-part-of (RPLDDDD / CISIZE)
      ```

      Compare the calculated values. If they are equal, you are assured the control interval has not moved. If they are not equal, dequeue resource from step “c” and start over at step “a”.

   g. Issue a PUT for the RPL that has the parameters `OPTCD=(SYN,KEY,DIR,UPD)`. This does not hold position in the buffer. You can do one of the following:

      - Issue a GET for the RPL that has the parameters `OPTCD=(SYN,KEY,UPD,DIR),ARG=MYKEY`. This will acquire position of the buffer.
      - Issue a PUT for the RPL that has the parameters `OPTCD=(SYN,KEY,DIR,NSP)`. This does hold position in the buffer.

   h. Issue an ENDREQ. This forces I/O to DASD, will drop the position, and cause data buffer invalidation.

   i. Dequeue `MYDATA.DSNAME.RELCI`.

2. A data set in which record additions and updates with length changes are permitted.

   In this case, the minimum locking unit is a control area to accommodate control interval splits. A higher level lock must be held during operations involving a control area split. The split activity must be serialized at a data set level. To perform a multilevel locking procedure, you must be prepared to use the information provided during VSAM JRNAD processing in your program. This user exit is responsible for determining the level of data movement and obtaining the appropriate locks.

   Higher concurrency can be achieved by a hierarchy of locks. Based on the particular condition, one or more of the locking hierarchies must be obtained.

Lock	Condition
Control Interval	Updating a record in place or adding a record to a control interval without causing a split.
Control Area	Adding a record or updating a record with a length change, causing a control interval split, but not a control area split.
Data Set	Adding a record or updating a record with a length change, causing a control area split.

The following is a basic procedure to provide the necessary protection. Note that, with this procedure, all updates are locked at the at the data set level:

```
SHAREOPTION = (4 3)          CBUF processing
Enqueue MYDATA.DSNAME        Shared for read only;
                             exclusive for write
Issue VSAM request macros
     ...
Dequeue MYDATA.DSNAME
```

In any sharing situation, it is a general rule that all resources be obtained and released between the locking protocol. All positioning must be released by using all direct requests or by issuing the ENDREQ macro before ending the procedure with the DEQ.

Cross-System Sharing

With cross-system SHAREOPTIONS 3, you have the added responsibility of passing the VSAM shared information (VSI) and invalidating data and/or index buffers. This can be done by using an informational control record as the low key or first record in the data set. The following information is required to accomplish the necessary index record invalidation:

1. Number of data control interval splits and index updates for sequence set invalidation
2. Number of data control area splits for index set invalidation

All data buffers should always be invalidated. See “Techniques of Data Sharing” on page 208 for the required procedures for invalidating buffers. To perform selective buffer invalidation, an internal knowledge of the VSAM control blocks is required.

Sharing VSAM Data Sets

Your program must serialize the following types of requests (precede the request with an ENQ and, when the request completes, issue a DEQ):

- All PUT requests.
- POINT, GET-direct-NSP, GET-skip, and GET-for-update requests that are followed by a PUT-insert, PUT-update, or ERASE request.
- VERIFY requests. When VERIFY is executed by VSAM, your program must have exclusive control of the data set.
- Sequential GET requests.

User Access to VSAM Shared Information

You can code the following instructions to get the length and address of the data to be sent to another processor:

- Load ACB address into register RY.
- To locate the VSI for a data component:

```
L   RX,04(,RY)    Put AMBL address into register RX
L   1,52(,RX)     Get data AMB address
L   1,68(,1)      Get VSI address
LH  0,62(,1)      Load data length
LA  1,62(,1)      Point to data to be communicated
```

- To locate the VSI information for an index component of a key-sequenced data set:

```
L   RX,04(,RY)    Put AMBL address into register RX
L   1,56(,RX)     Get index AMB address
L   1,68(,1)      Get VSI address
LH  0,62(,1)      Load data length
LA  1,62(,1)      Point to data to be communicated
```

Similarly, the location of the VSI on the receiving processor can be located. The VSI level number must be incremented in the receiving VSI to inform the receiving processor that the VSI has changed. To update the level number, assuming the address of the VSI is in register 1:

```
LA  0,1           Place increment into register 0
AL  0,64(,1)      Add level number to increment
ST  0,64(,1)      Save new level number
```

All processing of the VSI must be protected by using ENQ/DEQ to prevent simultaneous updates to the transmitted data.

If the data set can be shared between z/OS operating systems, a user's program in another system can concurrently access the data set. Before you open the data set specifying DISP=OLD, it is your responsibility to protect across systems with ENQ/DEQ using the UCB option. This protection is available with GRS or equivalent functions.

Chapter 14. Sharing Resources Among VSAM Data Sets

This chapter covers the following topics.

Topic	Page
Provision of a Resource Pool	211
Management of I/O Buffers for Shared Resources	216
Restrictions and Guidelines for Shared Resources	220

This chapter is intended to help you share resources among your VSAM data sets. VSAM has a set of macros that lets you share I/O buffers and I/O-related control blocks among many VSAM data sets. In VSAM, an I/O buffer is a virtual storage area from which the contents of a control interval are read and written. Sharing these resources optimizes their use, reducing the requirement for virtual storage and therefore reducing paging of virtual storage.

Sharing these resources is not the same as sharing a data set itself (that is, sharing among different tasks that independently open it). Data set sharing can be done with or without sharing I/O buffers and I/O-related control blocks. For information about data set sharing see “Chapter 13. Sharing VSAM Data Sets” on page 195.

There are also macros that let you manage I/O buffers for shared resources.

Sharing resources does not improve sequential processing. VSAM does not automatically position itself at the beginning of a data set opened for sequential access, because placeholders belong to the resource pool, not to individual data sets. When you share resources for sequential access, positioning at the beginning of a data set has to be specified explicitly with the POINT macro or the direct GET macro with RPL OPTCD=NSP. You may not use a resource pool to load records into an empty data set.

Provision of a Resource Pool

To share resources, follow this procedure to provide a resource pool:

1. Use the BLDVRP macro to build a resource pool.
2. Code a MACRF parameter in the ACB and use OPEN to connect your data sets to the resource pool.
3. After you have closed all the data sets, use the DLVRP macro to delete the resource pool.

Building a Resource Pool: BLDVRP

Issuing BLDVRP causes VSAM to share the I/O buffers and I/O-related control blocks of data sets whose ACBs indicate the corresponding option for shared resources. Control blocks are shared automatically; you may control the sharing of buffers.

When you issue BLDVRP, you specify for the resource pool the size and number of virtual address space buffers for each virtual buffer pool.

© Copyright IBM Corp. 1987, 2002

Using Hiperspace Buffers with LSR

If you are using local shared resources (LSR), you can specify the number of 4K-byte multiple Hiperspace buffers for each buffer pool in the resource pool. The size of the Hiperspace buffers must be equal to the CISIZE of the data sets being used.

The use of Hiperspace buffers can reduce the amount of I/O to a direct access storage device (DASD) by caching data in expanded storage. The data in a Hiperspace buffer is preserved unless there is an expanded storage shortage and the expanded storage that backs the Hiperspace buffer is reclaimed by the system. VSAM invalidates a Hiperspace buffer when it is copied to a virtual address space buffer and, conversely, invalidates a virtual address space buffer when it is copied to a Hiperspace buffer. Therefore at most there is only one copy of the control interval in virtual address space and Hiperspace. When a modified virtual address space buffer is reclaimed, it is copied to Hiperspace and to DASD.

For the data pool or the separate index pool at OPEN time, a data set is assigned the one buffer pool with buffers of the appropriate size—either the exact control interval size requested, or the next larger size available.

You may have both a global resource pool and one or more local resource pools. Tasks in an address space that have a local resource pool may use either the global resource pool, under the restrictions described below, or the local resource pool. There may be multiple buffer pools based on buffer size for each resource pool.

To share resources locally, a task in the address space issues BLDVRP TYPE=LSR, DATA|INDEX. To share resources globally, a system task issues BLDVRP TYPE=GSR. The program that issues BLDVRP TYPE=GSR must be in supervisor state with key 0 - 7.

You can share resources locally or globally, with the following restrictions:

- **LSR (local shared resources).** You can build up to 255 data resource pools and 255 index resource pools in one address space. Each resource pool must be built individually. The data pool must exist before the index pool with the same share pool identification can be built. The parameter lists for these multiple LSR pools can reside above or below 16 MB. The BLDVRP macro RMODE31 parameter indicates where VSAM is to obtain virtual storage when the LSR pool control blocks and data buffers are built.

  These resource pools are built with the BLDVRP macro TYPE=LSR and DATA|INDEX specifications. Specifying MACRF=LSR on the ACB or GENCB-ACB macros causes the data set to use the LSR pools built by the BLDVRP macro. The DLVRP macro processes both the data and index resource pools.
- **GSR (global shared resources).** All address spaces for a given protection key in the system share one resource pool. Only one resource pool can be built for each of the protection keys 0 - 7. With GSR, an access method control block and all related request parameter lists, exit lists, data areas, and extent control blocks must be in the common area of virtual storage with a protection key the same as the resource pool. To get storage in the common area with that protection key, issue the GETMAIN macro while in that key, for storage in subpool 241. If you need to share a data set among address spaces, multiple systems, or both, consider using record-level sharing (RLS) instead of GSR.

  The separate index resource pools are not supported for GSR.

The Hiperspace buffers (specified in the BLDVRP macro) are not supported for GSR.

Generate ACBs, RPLs, and EXLSTs with the GENCB macro: code the WAREA and LENGTH parameters. The program that issues macros related to that global resource pool must be in supervisor state with the same key. (The macros are BLDVRP, CHECK, CLOSE, DLVRP, ENDREQ, ERASE, GENCB, GET, GETIX, MODCB, MRKBFR, OPEN, POINT, PUT, PUTIX, SCHBFR, SHOWCB, TESTCB, and WRTBFR. The SHOWCAT macro is not related to a resource pool, because a program can issue this macro independently of an opened data set.)

Deciding the Size of a Virtual Resource Pool

The virtual resource pool for all components of the clusters or alternate indexes must be successfully built before any open is issued to use the resource pool; otherwise, the results might be unpredictable or performance problems might occur. To specify the BUFFERS, KEYLEN, and STRNO parameters of the BLDVRP macro, you must know the size of the control intervals, data records (if spanned), and key fields in the components that will use the resource pool. You must also know how the components are processed. You can use the SHOWCAT and SHOWCB macros, or the access method services LISTCAT command to get this information.

For example, to find the control interval size using SHOWCB: open the data set for nonshared resources processing, issue SHOWCB, close the ACB, issue BLDVRP, open the ACB for LSR or GSR.

Tip: Because Hiperspace buffers are in expanded storage, you do not need to consider their size and number when you calculate the size of the virtual resource pool.

For each VSAM cluster that will share the virtual resource pool you are building, follow this procedure:

1. Determine the number of concurrent requests you expect to process. The number of concurrent requests represents STRNO for the cluster.
2. Specify BUFFERS=(SIZE(STRNO+1)) for the data component of the cluster.
   - If the cluster is a key-sequenced cluster and the index CISZ (control interval size) is the same as the data CISZ, change the specification to BUFFERS=(SIZE(2 X STRNO)+1).
   - If the index CISZ is not the same as the data component CISZ, specify BUFFERS=(dataCISZ(STRNO+1),indexCISZ(STRNO)).

Following this procedure provides the minimum number of buffers needed to support concurrently active STRNO strings. An additional string is not dynamically added to a shared resource pool. The calculation can be repeated for each cluster which will share the resource pool, including associated alternate index clusters and clusters in the associated alternate index upgrade sets.

For each cluster component having a different CISZ, add another ‘,SIZE(NUMBER)’ range to the ‘BUFFERS=’ specification. Note that the data component and index component buffers may be created as one set of buffers, or, by use of the ‘TYPE=’ statement, may be created in separate index and data buffer sets.

Additional buffers may be added to enhance performance of applications requiring read access to data sets by reducing I/O requirements. You should also consider the need for cross-region or cross-system sharing of the data sets where modified data buffers must be written frequently to enhance read and update integrity. Many

buffers is not usually an advantage in such environments. In some applications where a resource pool is shared by multiple data sets and not all data set strings are active concurrently, less than the recommended number of buffers may produce satisfactory results.

If the specified number of buffers is not adequate, VSAM will return a logical error indicating the out-of-buffer condition.

Displaying Information about an Unopened Data Set

The SHOWCAT macro lets you get information about a component before its cluster or alternate index is opened. The program that is to issue BLDVRP can issue SHOWCAT on all the components to find out the sizes of control intervals, records, and keys. This information lets the program calculate values for the BUFFERS and KEYLEN parameters of BLDVRP.

A program need not be in supervisor state with protection key 0 - 7 to issue SHOWCAT, even though it must be in supervisor state and in protection key 0 - 7 to issue BLDVRP TYPE=GSR.

The SHOWCAT macro is described in *z/OS DFSMS Macro Instructions for Data Sets*.

Displaying Statistics about a Buffer Pool

You can use the SHOWCB macro to obtain statistics about the use of buffer pools. These statistics help you determine how to improve both a previous definition of a resource pool and the mix of data sets that use it. The statistics are available through an ACB that describes an open data set that is using the buffer pool. They reflect the use of the buffer pool from the time it was built to the time SHOWCB is issued. All but one of the statistics are for a single buffer pool. To get statistics for the whole resource pool, issue SHOWCB for each of its buffer pools.

The statistics cannot be used to redefine the resource pool while it is in use. You have to make adjustments the next time you build it.

The use of SHOWCB to display an ACB is described in “Manipulating the Contents of Control Blocks” on page 142. If the ACB has MACRF=GSR, the program that issues SHOWCB must be in supervisor state with protection key 0 - 7. A program check can occur if SHOWCB is issued by a program that is not in supervisor state with the same protection key as the resource pool.

For buffer pool statistics, the keywords described below are specified in FIELDS. These fields may be displayed only after the data set described by the ACB is opened. Each field requires one fullword in the display work area:

Field	Description
BFRFND	The number of requests for retrieval that could be satisfied without an I/O operation (the data was found in a buffer).
BUFRDS	The number of reads to bring data into a buffer.
NUIW	The number of nonuser-initiated writes (that VSAM was forced to do because no buffers were available for reading the contents of a control interval).
STRMAX	The maximum number of placeholders currently active for the resource pool (for all the buffer pools in it).
UIW	The number of user-initiated writes (PUTs not deferred or WRTBFRs, see “Deferring Write Requests” on page 216).

Connecting a Data Set to a Resource Pool: OPEN

You cause a data set to use a resource pool built by BLDVRP by specifying LSR or GSR in the MACRF parameter of the data set's ACB before you open the data set.

```
ACB   MACRF=({NSR|LSR|GSR},...),...
```

NSR, the default, indicates the data set does not use shared resources. LSR indicates it uses the local resource pool. GSR indicates it uses the global resource pool.

If the VSAM control blocks and data buffers reside above 16 MB, RMODE31=ALL must be specified in the ACB before OPEN is issued. If the OPEN parameter list or the VSAM ACB resides above 16 MB, the MODE=31 parameter of the OPEN macro must also be coded.

When an ACB indicates LSR or GSR, VSAM ignores its BSTRNO, BUFNI, BUFND, BUFSP, and STRNO parameters because VSAM will use the existing resource pool for the resources associated with these parameters.

To connect LSR pools with a SHRPOOL identification number other than SHRPOOL=0, you must use the SHRPOOL parameter of the ACB macro to indicate which LSR pool you are connecting.

If more than one ACB is opened for LSR processing of the same data set, the LSR pool identified by the SHRPOOL parameter for the first ACB will be used for all subsequent ACBs.

For a data set described by an ACB with MACRF=GSR, the ACB and all related RPLs, EXLSTs, ECBs, and data areas must be in the common area of virtual storage with the same protection key as the resource pool.

Deleting a Resource Pool Using the DLVRP Macro

After all data sets using a resource pool are closed, delete the resource pool by issuing the DLVRP (delete VSAM resource pool) macro. Failure to delete a local resource pool causes virtual storage to be lost until the end of the job step or TSO/E session. This loss is protected with a global resource pool. If the address space that issued BLDVRP terminates without having issued DLVRP, the system deletes the global resource pool when its use count is 0.

To delete an LSR pool with a SHRPOOL identification number other than SHRPOOL=0, you must use the SHRPOOL parameter to indicate which resource pool you are deleting. If both a data resource pool and an index resource pool have the same SHRPOOL number, both will be deleted.

If the DLVRP parameter list is to reside above 16 MB, the MODE=31 parameter must be coded.

Management of I/O Buffers for Shared Resources

Managing I/O buffers includes:

- Deferring writes for direct PUT requests, which reduces the number of I/O operations.
- Writing buffers that have been modified by related requests.
- Locating buffers that contain the contents of specified control intervals.
- Marking a buffer to be written without issuing a PUT.
- When your program accesses an invalid buffer, VSAM refreshes the buffer (that is, reads in a fresh copy of the control interval) before making its contents available to your program.

Managing I/O buffers should enable you to speed up direct processing of VSAM data sets that are accessed randomly. You probably will not be able to speed up sequential processing or processing of a data set whose activity is consistently heavy.

Deferring Write Requests

VSAM automatically defers writes for sequential PUT requests. It normally writes out the contents of a buffer immediately for direct PUT requests. With shared resources, you can cause writes for direct PUT requests to be deferred. Buffers are finally written out when:

- You issue the WRTBFR macro.
- VSAM needs a buffer to satisfy a GET request.
- A data set using a buffer pool is closed. (Temporary CLOSE is ineffective against a data set that is sharing buffers, and ENDREQ does not cause buffers in a resource pool to be written.)

Deferring writes saves I/O operations when subsequent requests can be satisfied by the data in the buffer pool. If you are going to update control intervals more than once, data processing performance will be improved by deferring writes.

You indicate that writes are to be deferred by coding MACRF=DFR in the ACB, along with MACRF=LSR or GSR.

```
ACB      MACRF=({LSR|GSR},{DFR| NDF},...),...
```

The DFR option is incompatible with SHAREOPTIONS 4. (SHAREOPTIONS is a parameter of the DEFINE command of access method services. It is described in *z/OS DFSMS Access Method Services*.) A request to open a data set with SHAREOPTIONS 4 for deferred writes is rejected.

VSAM notifies the processing program when an unmodified buffer has been found for the current request and there will be no more unmodified buffers into which to read the contents of a control interval for the next request. (VSAM will be forced to write a buffer to make a buffer available for the next I/O request.) VSAM sets register 15 to 0 and puts 12 (X'0C') in the feedback field of the RPL that defines the PUT request detecting the condition.

VSAM also notifies the processing program when there are no buffers available to be assigned to a placeholder for a request. This is a logical error (register 15 contains 8 unless an exit is taken to a LERAD routine). The feedback field in the RPL contains 152 (X'98'). You may retry the request; it gets a buffer if one is freed.

Relating Deferred Requests by Transaction ID

You can relate action requests (GET, PUT, and so forth) according to transaction by specifying the same ID in the RPLs that define the requests.

The purpose of relating the requests that belong to a transaction is to enable WRTBFR to cause all the modified buffers used for a transaction to be written. When the WRTBFR request is complete, the transaction is physically complete.

```
RPL
TRANSID=number,...
```

TRANSID specifies a number from 0 to 31. The number 0, which is the default, indicates that requests defined by the RPL are not associated with other requests. A number from 1 to 31 relates the requests defined by this RPL to the requests defined by other RPLs with the same transaction ID.

You can find out what transaction ID an RPL has by issuing SHOWCB or TESTCB.

```
SHOWCB     FIELDS=([TRANSID],...),...
```

TRANSID requires one fullword in the display work area.

```
TESTCB
    TRANSID=number,...
```

If the ACB to which the RPL is related has MACRF=GSR, the program issuing SHOWCB or TESTCB must be in supervisor state with the same protection key as the resource pool. With MACRF=GSR specified in the ACB to which the RPL is related, a program check can occur if SHOWCB or TESTCB is issued by a program that is not in supervisor state with protection key 0 - 7. For more information about using SHOWCB and TESTCB see “Manipulating the Contents of Control Blocks” on page 142.

Writing Buffers Whose Writing is Deferred: WRTBFR

If any PUTs to a data set using a shared resource pool are deferred, you can use the WRTBFR (write buffer) macro to write:

- All modified unwritten index and data buffers for a given data set (which causes all Hiperspace buffers for the data set to be invalidated)
- All modified unwritten index and data buffers in the resource pool
- The least recently used modified buffers in each buffer pool of the resource pool
- All buffers modified by requests with the same transaction ID
- A buffer, identified by an RBA value, that has been modified and has a use count of zero

You can specify the DFR option in an ACB without using WRTBFR to write buffers. A buffer is written when VSAM needs one to satisfy a GET request, or all modified buffers are written when the last of the data sets that uses them is closed.

Besides using WRTBFR to write buffers whose writing is deferred, you can use it to write buffers that are marked for output with the MRKBFR macro, which is described in “Marking a Buffer for Output: MRKBFR” on page 219.

Using WRTBFR can improve performance, if you schedule WRTBFR to overlap other processing.

VSAM notifies the processing program when there are no more unmodified buffers into which to read the contents of a control interval. (VSAM would be forced to write buffers when another GET request required an I/O operation.) VSAM sets register 15 to 0 and puts 12 (X'0C') in the feedback field of the RPL that defines the PUT request that detects the condition.

VSAM also notifies the processing program when there are no buffers available to which to assign a placeholder for a request. This is a logical error (register 15 contains 8 unless an exit is taken to a LERAD routine); the feedback field in the RPL contains 152 (X'98'). You may retry the request; it gets a buffer if one is freed.

When sharing the data set with a user in another region, your program might want to write the contents of a specified buffer without writing all other modified buffers. Your program issues the WRTBFR macro to search your buffer pool for a buffer containing the specified RBA. If found, the buffer is examined to verify that it is modified and has a use count of zero. If so, VSAM writes the contents of the buffer into the data set.

Recommendation: Before you use WRTBFR TYPE=CHK|TRN|DRBA, be sure to release all buffers. See “Processing Multiple Strings” on page 150 for information about releasing buffers. If one of the buffers is not released, VSAM defers processing until the buffer is released.

Handling Exits to Physical Error Analysis Routines

With deferred writes of buffers, a processing program continues after its PUT request has been completed, even though the buffer has not been written. The processing program is not synchronized with a physical error that occurs when the buffer is finally written. A processing program that uses MRKBFR MARK=OUT is also not synchronized with a physical error. An EXCEPTION or a SYNAD routine must be supplied to analyze the error.

The ddname field of the physical error message identifies the data set that was using the buffer, but, because the buffer might have been released, its contents might be unavailable. You can provide a JRNAD exit routine to record the contents of buffers for I/O errors. It can be coordinated with a physical error analysis routine to handle I/O errors for buffers whose writing has been deferred. If a JRNAD exit routine is used to cancel I/O errors during a transaction, the physical error analysis routine will get only the last error return code. See “SYNAD Exit Routine to Analyze Physical Errors” on page 255 and “JRNAD Exit Routine to Journalize Transactions” on page 245 for information about the SYNAD and JRNAD routines.

Using the JRNAD Exit with Shared Resources

VSAM takes the JRNAD exit for the following reasons when the exit is associated with a data set whose ACB has MACRF=LSR or GSR:

- A data or index control interval buffer has been modified and is about to be written.
- A physical error occurred. VSAM takes the JRNAD exit first—your routine can direct VSAM to bypass the error and continue processing or to terminate the request that occasioned the error and proceed with error processing.
- A control interval or a control area is about to be split for a key-sequenced data set or variable-length RRDS. Your routine can cancel the request for the split and leave VSAM. An example of using the JRNAD exit for this purpose is given in “JRNAD Exit Routine to Journalize Transactions” on page 245.

See “JRNAD Exit Routine to Journalize Transactions” on page 245 for information describing the contents of the registers when VSAM exists to the JRNAD routine, and the fields in the parameter list pointed to by register 1.

Accessing a Control Interval with Shared Resources

Control interval access is not permitted with shared resources.

Locating an RBA in a Buffer Pool: SCHBFR

When a resource pool is built, the buffers in each buffer pool are numbered from 1 through the number of buffers in each buffer pool. At a given time, several buffers in a buffer pool may hold the contents of control intervals for a particular data set. These buffers may or may not contain RBAs of interest to your processing program. The SCHBFR macro lets you find out. Specify in the ARG parameter of the RPL that defines SCHBFR the address of an 8-byte field that contains the first and last control interval RBAs of the range you are interested in.

Note: For compressed format data sets, the RBA of the compressed record is unpredictable. The RBA of another record or the address of the next record in the buffer cannot be determined using the length of the current record or the length of the record provided to VSAM.

The buffer pool to be searched is the one used by the data component defined by the ACB to which your RPL is related. If the ACB names a path, VSAM searches the buffer pool used by the data component of the alternate index. (If the path is defined over a base cluster alone, VSAM searches the buffer pool used by the data component of the base cluster.) VSAM begins its search at the buffer you specify and continues until it finds a buffer that contains an RBA in the range or until the highest numbered buffer is searched.

For the first buffer that satisfies the search, VSAM returns its address (OPTCD=LOC) or its contents (OPTCD=MVE) in the work area whose address is specified in the AREA parameter of the RPL and returns its number in register 0. If the search fails, Register 0 is returned with the user specified buffer number and a one-byte SCHBFR code of X'0D'. To find the next buffer that contains an RBA in the range, issue SCHBFR again and specify the number of the next buffer after the first one that satisfied the search. You continue until VSAM indicates it found no buffer that contains an RBA in the range or until you reach the end of the pool.

Finding a buffer that contains a desired RBA does not get you exclusive control of the buffer. You may get exclusive control only by issuing GET for update. SCHBFR does not return the location or the contents of a buffer that is already under the exclusive control of another request.

Marking a Buffer for Output: MRKBFR

You locate a buffer that contains the RBA you are interested in by issuing a SCHBFR macro, a read-only GET, or a GET for update. When you issue GET for update, you get exclusive control of the buffer. Whether you have exclusive control or not, you can mark the buffer for output by issuing the MRKBFR macro with MARK=OUT and then change the buffer’s contents. Without exclusive control, you should not change the control information in the CIDF or RDFs (do not change the record lengths).

MRKBFR MARK=OUT, indicates that the buffer's contents are modified. You must modify the contents of the buffer itself, not a copy. Therefore, when you issue SCHBFR or GET to locate the buffer, you must specify RPL OPTCD=LOC. (If you use OPTCD=MVE, you get a copy of the buffer but do not learn its location.) The buffer is written when a WRTBFR is issued or when VSAM is forced to write a buffer to satisfy a GET request.

If you are sharing a buffer or have exclusive control of it, you can release it from shared status or exclusive control with MRKBFR MARK=RLS. If the buffer was marked for output, MRKBFR with MARK=RLS does not nullify it; the buffer is eventually written. Sequential positioning is lost. MRKBFR with MARK=RLS is similar to the ENDREQ macro.

Restrictions and Guidelines for Shared Resources

Restrictions for using the LSR and GSR options are:

- Empty data sets cannot be processed (that is, loaded).
- Multiple LSR pools in an address space are obtained by using the SHRPOOL parameter of the BLDVRP macro to identify each LSR pool.
- Control interval access cannot be used (ACB MACRF=CNV and ACB MACRF=ICI).
- Control blocks in common (CBIC) cannot be used.
- User buffering is not allowed (ACB MACRF=UBF).
- Writes for data sets with SHAREOPTIONS 4 cannot be deferred (ACB MACRF=DFR).
- Request parameter lists for MRKBFR, SCHBFR, and WRTBFR cannot be chained (the NXTRPL parameter of the RPL macro is ignored).
- For sequential access, positioning at the beginning of a data set must be explicit: with a POINT macro or a direct GET macro with RPL OPTCD=NSP.
- Temporary CLOSE and ENDREQ do not cause buffers to be written if MACRF=DFR was specified in the associated ACB.
- Address spaces that use Hiperspace buffering (LSR only) should be made nonswappable. Otherwise, the expanded storage (and, therefore, the Hiperspace buffers) will be discarded when the address space is swapped out.
- With GSR, an ACB and all related RPLs, EXLSTs, data areas, and ECBs must be stored in the common area of virtual storage with protection key 0 - 7; all VSAM requests related to the global resource pool may be issued only by a program in supervisor state with protection key 0 - 7 (the same as the resource pool).
- Checkpoints cannot be taken for data sets whose resources are shared in a global resource pool. When a program in an address space that opened a data set whose ACB has MACRF=GSR issues the CHKPT macro, 8 is returned in register 15. If a program in another address space issues the CHKPT macro, the checkpoint is taken, but only for data sets that are not using the global resource pool.

  Checkpoint/restart can be used with data sets whose resources are shared in a local resource pool, but the restart program does not reposition for processing at the point where the checkpoint occurred—processing is restarted at a data set's highest used RBA. See *z/OS DFSMS Checkpoint/Restart* for information about restarting the processing of VSAM data.

- If a physical I/O error is found while writing a control interval to the direct access device, the buffer remains in the resource pool. The write-required flag (BUFCMW) and associated mod bits (BUFCMDBT) are turned off, and the BUFC is flagged in error (BUFCER2=ON). The buffer is not replaced in the pool, and buffer writing is not attempted. To release this buffer for reuse, a WRTBFR macro with TYPE=DS can be issued or the data set can be closed (CLOSE issues the WRTBFR macro).
- When you use the BLDVRP macro to build a shared resource pool, some of the VSAM control blocks are placed in a system subpool and others in subpool 0. When a task ends, the system frees subpool 0 unless it is shared with another task. The system does not free the system subpool until the job step ends. Then, if another task attempts to use the resource pool, an abend might occur when VSAM attempts to access the freed control blocks. This problem does not occur if the two tasks share subpool 0. Code in the ATTACH macro the SZERO=YES parameter, or the SHSPL or SHSPV parameters. SZERO=YES is the default.
- GSR is not permitted for compressed data sets.

Chapter 15. Using VSAM Record-Level Sharing

This chapter covers the following topics.

Topic	Page
Record-Level Sharing CF Caching	224
CICS Use of VSAM RLS	225
Non-CICS Use of VSAM RLS	228
Non-RLS Access to VSAM Data Sets	229
Differences Between RLS Access and Non-RLS Access	229
VSAM RLS Request Execution Mode Requirements	231
VSAM RLS Read Integrity Options	232

VSAM record-level sharing (RLS) is an access option for VSAM data sets. This option provides multisystem sharing of VSAM data sets across a System/390 Parallel Sysplex®. VSAM RLS exploits the data sharing technology of the coupling facility (CF) including a CF-based lock manager and a CF cache manager. VSAM RLS uses the CF-based lock manager and the CF cache manager in its implementation of record-level sharing.

RLS is a mode of access to VSAM spheres. RLS is not an attribute of a sphere. It is an access option interpreted at OPEN time. The option is selected by either specifying a new JCL parameter (RLS) or by specifying MACRF=RLS in the ACB. The RLS MACRF option is mutually exclusive with the MACRF NSR (nonshared resources), LSR (local shared resources), and GSR (global shared resources) options. This chapter uses the term non-RLS access to distinguish between RLS access and NSR/LSR/GRS access.

Access method services do not use RLS when performing EXPORT, IMPORT, PRINT or REPRO commands. If the RLS keyword is specified on the DD statement of a file to be opened by access method services, the keyword is ignored and the file is opened and accessed in non-RLS mode. See “Non-RLS Access to VSAM Data Sets” on page 229 for more information about non-RLS access.

RLS access is supported for KSDS, ESDS, RRDS, and VRRDS data sets. RLS access is supported for VSAM alternate indexes. RLS is not supported for control interval mode access (CNV or ICI) or for HFS file.

The VSAM RLS functions are provided by the SMSVSAM server. This server resides in a system address space. The address space is created and the server is started at MVS IPL time. VSAM internally performs cross-address space accesses and linkages between requestor address spaces and the SMSVSAM server address space.

The SMSVSAM server owns two data spaces. One data space is called the SMSVSAM data space. It contains some VSAM RLS control blocks and a system-wide buffer pool. The other data space is used by VSAM RLS to collect activity monitoring information that is used to produce SMF records. VSAM provides the cross-address space access/linkage between the requestor address spaces and the SMSVSAM address/data spaces. See Figure 31 on page 224.

© Copyright IBM Corp. 1987, 2002

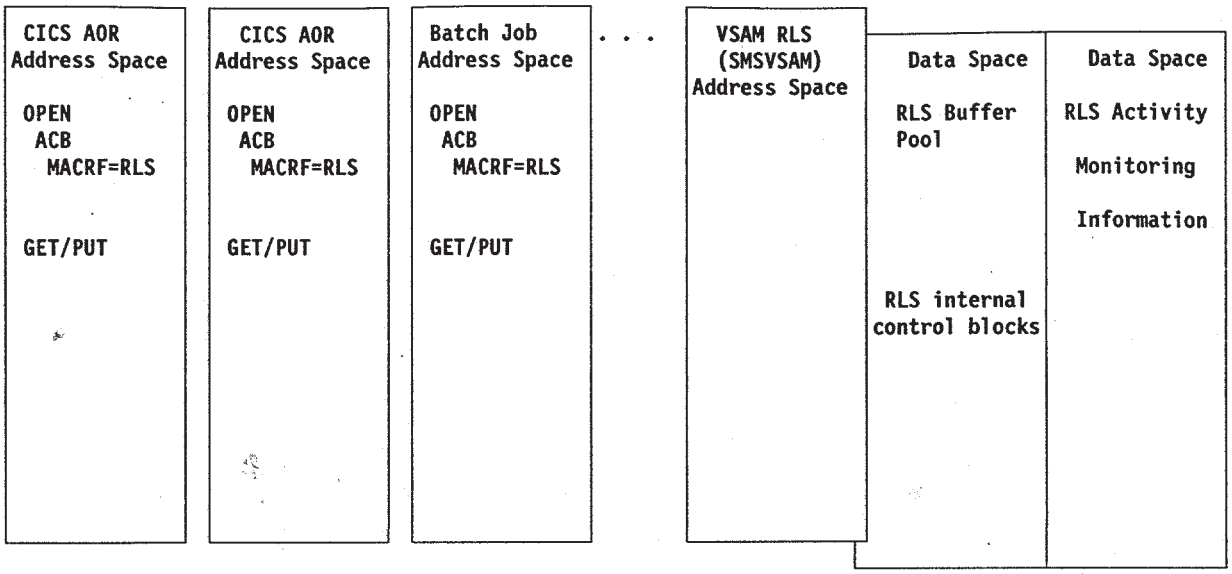


Figure 31. VSAM RLS Address/Data Spaces and Requestor Address Spaces

Record-Level Sharing CF Caching

VSAM record-level sharing allows multiple levels of CF caching for DFSMS cache structures that are defined in the active Storage Management Subsystem (SMS) configuration.

Before z/OS Version 1 Release 3, VSAM RLS placed the data of a VSAM sphere in the CF cache structure only when the sphere's CI size was less than 4K. For CIs greater than 4K, VSAM RLS placed the first 2K in the CF cache (to utilize the cross-invalidate features of the coupling facility). To retrieve a CI that was greater than 4K, VSAM RLS processing would always read the data from a direct access storage device (DASD) volume.

Since z/OS Version 1 Release 3, VSAM RLS has multiple levels of CF caching. The value of the SMS DATACLAS RLS CF Cache Value keyword determines the level of CF caching. The default value, ALL, indicates that RLS caches all data for a sphere in the coupling facility (for both index and data parts of the sphere). If you specify NONE, then RLS caches only the index part of the VSAM sphere. If you specify UPDATESONLY, then RLS caches data in the coupling facility only during write operations.

All active systems in a sysplex must have the greater than 4K CF caching code installed before the function is enabled.

To set up RLS CF caching, use the following values:

- ALL or UPDATESONLY or NONE for the SMS DATACLAS RLS CF Cache Value keyword

  To allow greater than 4K caching of DFSMS VSAM data sets open for RLS processing, you need to make the following changes:

  – You can change the value of the SMS DATACLAS RLS CF Cache Value keyword if you do not want caching of all VSAM RLS data:

    ALL Indicates that RLS is to cache VSAM index and data components. **ALL** is the default.

    NONE Indicates that RLS is to cache only the VSAM index data. The data components are not to be placed in the cache structure.

    UPDATESONLY
    Indicates that RLS is to place only WRITE requests in the cache structure.

  – VSAM honors the RLS CF Cache Value keyword only when you specify RLS_MaxCfFeatureLevel(A) and all systems in the sysplex can run the greater than 4K caching code.

    To determine the code level on each system in the sysplex and whether the RLS CF Cache Value keyword is honored, use the 'D SMS,SMSVSAM', 'D SMS,SMSVSAM,ALL', D SMS,CFCACHE() operator commands. When DFSMS cache structures connect to the system, VSAM RLS issues an IGW500I message to indicate that greater than 4K caching is active. The cache structures connect to the system through the first instance of a sphere opened on each system.

- You can specify the following values for the RLS_MaxCfFeatureLevel keyword:

  – A—This value allows greater than 4K caching if all active VSAM RLS instances in the sysplex have the correct level of code.

  – Z—This is the default value if you do not specify RLS_MaxCfFeatureLevel in the active SMS configuration. Greater than 4K caching is not allowed.

- RLS_MaxCfFeatureLevel keyword in the SETSMS command
- RLS_MaxCfFeatureLevel keyword in the SET SMS*xx* command
- RLS_MaxCfCacheFeatureLevel in the D SMS,OPTIONS command

CICS Use of VSAM RLS

The Customer Information Control System (CICS) file control component is a transactional file system built on top of VSAM. Prior to VSAM RLS, CICS file control performs its own record-level locking. The VSAM files are accessed through a single CICS. Local files are accessed by the CICS application owning region (AOR) submitting requests directly to VSAM. Remote (shared) files are accessed by the CICS AOR submitting (function shipping) a file control request to a CICS file owning region (FOR) and the FOR submitting a request to VSAM. CICS File Control provides transactional function such as commit, rollback, and forward recovery logging functions for recoverable files. Figure 32 shows the AOR, FOR, and VSAM request flow prior to VSAM RLS.

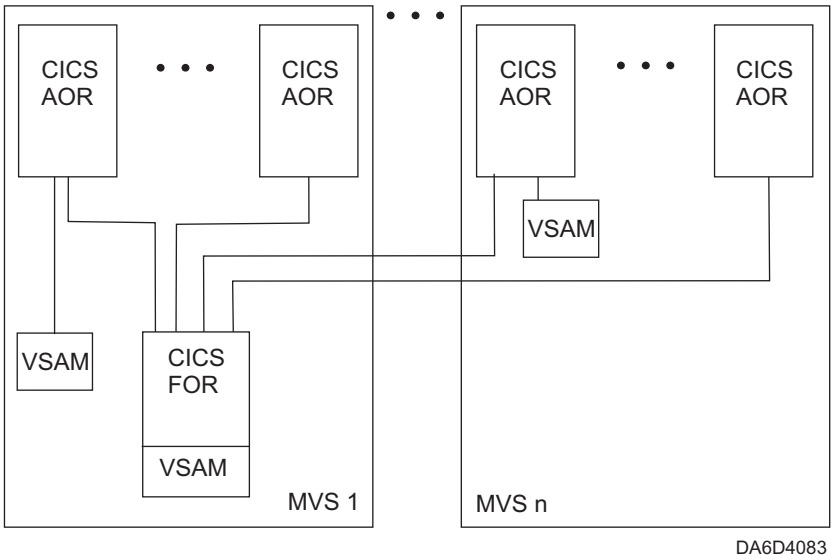


Figure 32. CICS VSAM Non-RLS Access

CICS AOR's function ships VSAM requests to access a specific file to the CICS FOR that owns that file. This distributed access form of data sharing has existed in CICS for some time.

With VSAM RLS, multiple AOR CICSs can directly share access a VSAM data set without CICS function shipping. With VSAM RLS, CICS continues to provide the transactional functions. The transactional functions are not provided by VSAM RLS itself. VSAM RLS provides CF-based record-level locking and CF data caching. Figure 33 shows a CICS configuration with VSAM RLS.

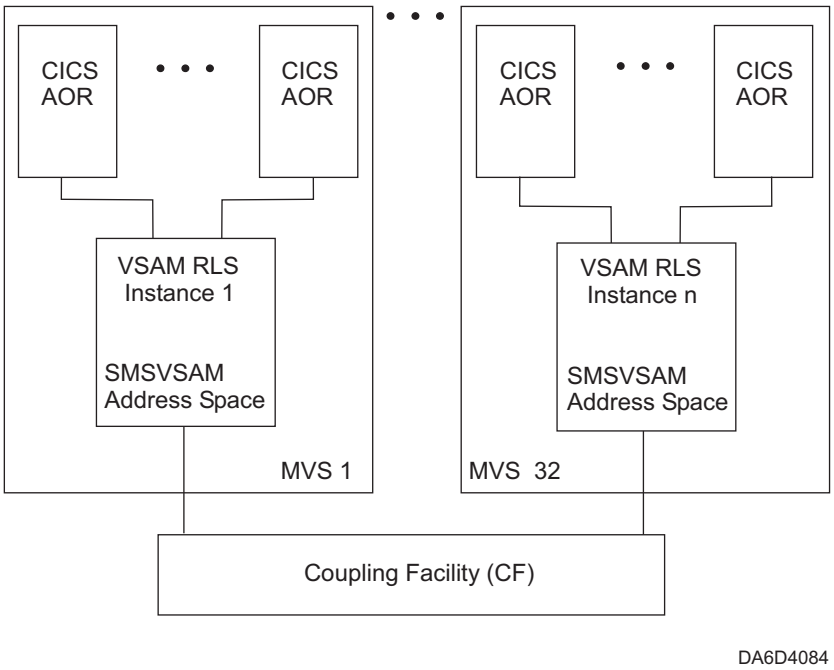


Figure 33. CICS VSAM RLS

VSAM RLS is a multisystem server. The CICS AORs access the shared files by submitting requests directly to the VSAM RLS server. The server uses the CF to serialize access at the record level.

Recoverable and Nonrecoverable Files

CICS File Control supports a recoverable/nonrecoverable file concept. The file definition includes a recoverability attribute LOG. The attribute options are specified as follows:

- LOG(NONE)—nonrecoverable

  Specifies the file as nonrecoverable. CICS does not perform any logging of changes for a file that has this attribute. Neither rollback, nor forward recovery is provided.
- LOG(UNDO)—recoverable

  Specifies the file as commit/rollback recoverable. CICS logs the before (UNDO) images of changes to the file and backs out the changes if the application requests rollback or if the transaction abnormally terminates.
- LOG(ALL)—recoverable

  Specifies the file as both commit/rollback recoverable and forward recoverable. In addition to the logging and recovery functions provided for LOG(UNDO), CICS logs the after (REDO) image of changes to the file. The REDO log records are used by forward recovery programs/products such as CICSVR (CICS VSAM Recovery) to reconstruct the file in the event of hardware or software damage to the file.

CICS recoverable file attributes can be used with VSAM data set attributes specifiable on IDCAMS (Access Method Services) DEFINE and ALTER commands. LOG can be specified in the data class along with BWO and LOGSTREAMID. Only a CICS application can open a recoverable data set for output. The reason for this is that VSAM RLS itself does not provide the logging and other transactional file functions required when writing to a recoverable file.

When a data set is opened in non-RLS (NSR, LSR, GSR) access mode, the recoverable attributes of the data set do not apply and are ignored. Thus, existing programs which do not use RLS access are not impacted by the recoverable data set rules.

CICS Transactional Recovery for VSAM Recoverable Files

The transactional services of CICS provide an ideal environment for data sharing. All changes to recoverable files made by a transaction remain invisible to other transactions until the modifying transaction reaches commit. Exclusive locks held by VSAM RLS on the modified records cause read-with-integrity and write requests to these records by other transactions to wait. Once the modifying transaction commits or rolls back, the locks are released and the other transactions can now access the records.

The CICS rollback (backout) function removes changes made to the recoverable files by a transaction. When a transaction abnormally terminates, CICS implicitly performs a rollback.

The commit and rollback functions isolate a transaction from the changes made to recoverable files or in general, recoverable resources by other transactions. This lets the transaction logic focus on the function it is providing and not have to be concerned with data recovery or cleanup in the event of problems or failures.

Non-CICS Use of VSAM RLS

When VSAM RLS is used outside of CICS, the applications do not have the transactional recovery environment. In most cases, this makes read/write data sharing not feasible. The level of support that VSAM RLS provides for non-CICS applications is described below.

Read Sharing of Recoverable Data Sets

A non-CICS application is permitted to open a recoverable data set in RLS mode only for input. VSAM RLS provides the necessary record-level locking to provide read-with-integrity (if requested) for the non-CICS application. This support lets multiple CICSs have the sphere open for read/write RLS-access. CICS provides the necessary transactional recovery for the writes to the recoverable data set. Concurrently, non-CICS applications can have the sphere open for read RLS access. VSAM provides the necessary locking. Since the non-CICS application is not permitted to write to the sphere, transaction recovery is not required.

Read Sharing Integrity Across KSDS CI and CA Splits

VSAM with non-RLS access does not ensure read integrity across splits for non-RLS access to a data set with cross-region share options 2, 3, and 4. If read integrity is required, the application must ensure it. When KSDS CI and CA splits move records from one CI to another CI, there is no way the writer can invalidate the data and index buffers for the reader. This can result in the reader not seeing some records that were moved.

VSAM RLS can ensure read integrity across splits. It uses the cross-invalidate function of the CF to invalidate copies of data and index CI in buffer pools other than the writer's buffer pool. This ensures that all RLS readers, CICS and non-CICS, are able to see any records moved by a concurrent CI or CA split. On each GET request, VSAM RLS tests validity of the buffers and when invalid, the buffers are refreshed from the CF or DASD.

Read/Write Sharing of Nonrecoverable Data Sets

When accessed through CICS, nonrecoverable data sets (files) do not participate in transactional recovery. Commit, rollback, and forward recovery logging do not apply to these spheres. Since transactional recovery is not required, VSAM RLS permits read/write sharing of nonrecoverable data sets concurrently by CICS and non-CICS applications. Any application can open the sphere for output in RLS mode.

VSAM RLS provides record locking and buffer coherency across the CICS and non-CICS read/write sharers of nonrecoverable data sets. However, the record lock on a new or changed record is released as soon as the buffer containing the change has been written to the CF cache and DASD. This differs from the case where a CICS transaction modifies VSAM RLS recoverable data sets and the corresponding locks on the added and changed records remain held until end-of-transaction.

For sequential and skip-sequential processing, VSAM RLS does not write a modified control interval (CI) until the processing moves to another CI or an ENDREQ is issued by the application. In the event of an application abnormal termination or abnormal termination of the VSAM RLS server, these buffered changes are lost.

While VSAM RLS permits read/write sharing of nonrecoverable data sets across CICS and non-CICS programs, most programs are not designed to tolerate this sharing. The absence of transactional recovery requires very careful design of the data and the programs.

Non-RLS Access to VSAM Data Sets

RLS access does not change the format of the data in the VSAM data sets. The data sets are compatible for non-RLS access. If the data set has been defined with a cross-region share option of 2, a non-RLS open for input is permitted while the data set is open for RLS processing; but a non-RLS open for output fails. If the data set is already open for non-RLS output, an open for RLS fails. Therefore, at any time, a data set (sphere) can be open for non-RLS write access or open for RLS access.

CICS and VSAM RLS provide a quiesce function to assist in the process of switching a sphere from CICS RLS usage to non-RLS usage.

Differences Between RLS Access and Non-RLS Access

This section describes the differences between RLS access and non-RLS access.

Share Options

For non-RLS access, VSAM uses the share options settings to determine the type of sharing permitted. If you set the cross-region share option to 2, a non-RLS open for input is permitted while the data set is already open for RLS access. VSAM provides full read and write integrity for the RLS users, but does not provide read integrity for the non-RLS user. A non-RLS open for output is not permitted when already opened for RLS.

VSAM RLS provides full read and write sharing for multiple users; it does not use share options settings to determine levels of sharing. When an RLS open is requested and the data set is already open for non-RLS input, VSAM does check the cross-region setting. If it is 2, then the RLS open is permitted. The open fails for any other share option or if the data set has been opened for non-RLS output.

Locking

Non-RLS provides local locking (within the scope of a single buffer pool) at the VSAM control interval level of granularity. Locking contention can result in an “exclusive control conflict” error response to a VSAM record management request.

VSAM RLS uses a DFSMS lock manager to provide a system-managed duplexing rebuild process. The locking granularity is at the VSAM record level. When contention occurs on a VSAM record, the request that encountered the contention waits for the contention to be removed. The DFSMS lock manager provides deadlock detection. When a lock request is in deadlock, VSAM rejects the request. This results in the VSAM record management request completing with a deadlock error response.

When you request a user-managed rebuild for a lock structure, the validity check function determines if there is enough space for the rebuild process to complete. If there is not enough space, the system rejects the request and displays an informational message.

When you request an alter operation for a lock structure, the validity check function determines if there is enough space for the alter process to complete. If there is not enough space, the system displays a warning message that includes the size recommendation.

VSAM RLS supports a timeout value which can be specified through the RPL. CICS uses this parameter to ensure that a transaction does not wait indefinitely for a lock to become available. VSAM RLS uses a timeout function of the DFSMS lock manager.

Retaining Locks

VSAM RLS uses share and exclusive record locks to control access to the shared data. An exclusive lock is used to ensure that a single user is updating a specific record. The exclusive lock causes any read-with-integrity request for the record by another user (CICS transaction or non-CICS application) to wait until the update is finished and the lock released.

Failure conditions can delay completion of an update to a recoverable file. This occurs when a CICS transaction enters in-doubt status. This means CICS can neither rollback nor commit the transaction. Therefore, the recoverable records modified by the transaction must remain locked. Failure of a CICS AOR also causes the current transactions updates to recoverable files to not complete. They cannot complete until the AOR is restarted.

When a transaction enters in-doubt, sysplex failure, MVS failure, failure of an instance of the SMSVSAM Address Space, or a CICS AOR terminates, any exclusive locks on records of recoverable files held by the transaction must remain held. However, other users waiting for these locks should not continue to wait. The outage is likely to be longer than the user would want to wait. When these conditions occur, VSAM RLS converts these exclusive record locks into retained locks.

Both exclusive and retained locks are not available to other users. When another user encounters lock contention with an exclusive lock, the user's lock request waits. When another user encounters lock contention with a retained lock, the lock request is immediately rejected with "retained lock" error response. This results in the VSAM record management request that produced the lock request failing with "retained lock" error response.

Supporting Non-RLS Access While Retained Locks Exist

Retained locks are created when a failure occurs. The locks need to remain until completion of the corresponding recovery. The retained locks only have meaning for RLS access. Lock requests issued by RLS access requests can encounter the retained locks. Non-RLS access does not perform record locking and therefore would not encounter the retained locks.

In order to ensure integrity of a recoverable sphere, VSAM does not permit non-RLS update access to the sphere while retained locks exist for that sphere. There can be situations where an installation must execute some non-CICS applications that require non-RLS update access to the sphere. VSAM RLS provides an IDCAMS command (SHCDS PERMITNONRLSUPDATE) that can be used to set the status of a sphere to enable non-RLS update access to a recoverable sphere while retained locks exist. This command does not release the retained locks. If this function is used, VSAM remembers its usage and informs the CICSs that hold the retained locks when they later open the sphere with RLS.

VSAM Options Not Supported by RLS

RLS does not support the following options and capabilities:

- Linear data sets
- Addressed access to a KSDS
- Control interval (CNV or ICI) to any VSAM data set type
- User buffering (UBF)
- Clusters that have been defined with the IMBED option
- Key Range data sets
- Temporary data sets
- GETIX and PUTIX requests
- MVS Checkpoint/Restart facility
- ACBSDS (system data set) specification
- Hiperbatch
- Catalogs, VVDS, the JRNAD exit, and any JCL AMP= parameters in JCL
- Data that is stored in z/OS UNIX System Services

In addition, VSAM RLS has the following restrictions:

- You cannot specify RLS access when accessing a VSAM data set using the ISAM compatibility interface.
- You cannot open individual components of a VSAM cluster for RLS access.
- You cannot specify a direct open of an alternate index for RLS access, but you can specify RLS open of an alternate index path.
- RLS open does not implicitly position to the beginning of the data set. For sequential or skip-sequential processing, specify a POINT or GET DIR, NSP request to establish a position in the data set.
- RLS does not support a request that is issued while the caller is executing in any of the following modes: cross-memory mode, SRB mode, or under an FRR. See “VSAM RLS Request Execution Mode Requirements” for a complete list of mode requirements.
- HFS files.

VSAM RLS Request Execution Mode Requirements

When a program issues a VSAM RLS request (OPEN, CLOSE, or Record Management request), the program must be executing in the following execution mode with the listed constraints:

- Task mode (not SRB mode)
- Address Space Control=Primary
- Home address space=Primary address space=Secondary address space
- No Functional Recovery Routine (FRR) can be in effect but ESTAE

The VSAM RLS record management request task must be the same task that opened the ACB, or the task that opened the ACB must be in the task hierarchy. That is, the record management task was attached by the task that opened the ACB, or by a task that was attached by the task that opened the ACB.

VSAM RLS Read Integrity Options

VSAM RLS provides three levels of read integrity as follows:

1. NRI—no read integrity

   This tells VSAM RLS not to obtain a record lock on the record accessed by a GET or POINT request. This avoids the overhead of record locking. This is sometimes referred to as dirty read because the reader might see an uncommitted change made by another transaction.

   Even with this option specified, VSAM RLS still performs buffer validity checking and buffer refresh when the buffer is invalid. Thus, a sequential reader of a KSDS does not miss records that are moved to new control intervals by control interval (CI) and control area (CA) splits.

   There are situations where VSAM RLS temporarily obtains a shared lock on the record even though NRI is specified. This happens when the read encounters an inconsistency within the VSAM sphere while attempting to access the record. An example of this is path access through an alternate index to a record for which a concurrent alternate index upgrade is being performed. The path access sees an inconsistency between the alternate index and base cluster. This would normally result in an error response return code 8 and reason code 144. Before giving this response to the NRI request, VSAM RLS obtains a shared lock on the base cluster record that was pointed to by the alternate index. This ensures that if the record was being modified, the change and corresponding alternate index upgrade completes. The record lock is released. VSAM retries the access. The retry should find the record correctly. This internal record locking may encounter locking errors such as deadlock or timeout. Your applications must be prepared to accept locking error return codes that may be returned on GET or POINT NRI requests. Normally such errors will not occur.

2. CR—consistent read

   This tells VSAM RLS to obtain a SHARE lock on the record accessed by a GET or POINT request. It ensures the reader does not see an uncommitted change made by another transaction. Instead, the GET/POINT waits for the change to be committed or backed out and the EXCLUSIVE lock on the record to be released.

3. CRE—consistent read explicit

   This is the same as CR, except VSAM RLS keeps the SHARE lock on the record until end-of-transaction. This option is only available to CICS transactions. VSAM does not understand end-of-transaction for non-CICS usage.

   This capability is often referred to as REPEATABLE READ.

The record locks obtained by the VSAM RLS GET requests with CRE option inhibit update or erase of the records by other concurrently executing transactions. However, the CRE requests do not inhibit the insert of other records by other transactions. The following cases need to be considered when using this function.

a. If a GET DIR (Direct) or SKP (Skip Sequential) request with CRE option receives a “record not found” response, VSAM RLS does not retain a lock on the nonexistent record. The record could be inserted by another transaction.

b. A sequence of GET SEQ (sequential) requests with CRE option results in a lock being held on each record that was returned. However, no additional locks are held that would inhibit the insert of new records in between the records locked by the GET CRE sequential processing. If the application were to re-execute the previously executed sequence of GET SEQ,CRE requests, it would see any newly inserted records. Within the transactional recovery community, these records are referred to as “phantom” records. The VSAM RLS CRE function does not inhibit phantom records.

Chapter 16. Checking VSAM Key-Sequenced Data Set Clusters for Structural Errors

This chapter covers the following topics.

Topic	Page
EXAMINE Command	235
How to Run EXAMINE	236
Samples of Output from EXAMINE Runs	238

This chapter describes how the service aid, EXAMINE, is used to analyze a key-sequenced data set (KSDS) cluster for structural errors.

EXAMINE Command

EXAMINE is an access method services command that lets users analyze and collect information on the structural consistency of key-sequenced data set clusters. This service aid consists of two tests: INDEXTEST and DATATEST.

INDEXTEST examines the index component of the key-sequenced data set cluster by cross-checking vertical and horizontal pointers contained within the index control intervals, and by performing analysis of the index information. It is the default test of EXAMINE.

DATATEST evaluates the data component of the key-sequenced data set cluster by sequentially reading all data control intervals, including free space control intervals. Tests are then carried out to ensure record and control interval integrity, free space conditions, spanned record update capacity, and the integrity of various internal VSAM pointers contained within the control interval.

For a description of the EXAMINE command syntax, see *z/OS DFSMS Access Method Services*.

Types of Data Sets

EXAMINE can test the following types of data sets:

- Key-sequenced data set
- Catalog

EXAMINE Users

EXAMINE end users fall into two categories:

1. **Application Programmer/Data Set Owner.** These users want to know of any structural inconsistencies in their data sets, and they are directed to corresponding recovery methods that IBM supports by the appropriate summary messages. The users' primary focus is the condition of their data sets; therefore, they should use the ERRORLIMIT(0) parameter of EXAMINE to suppress printing of detailed error messages.
2. **System Programmer/Support Personnel.** System programmers or support personnel need the information from detailed error messages to document or fix a problem with a certain data set.

© Copyright IBM Corp. 1987, 2002

Checking VSAM Key-Sequenced Data Set Clusters for Structural Errors

Users must have master level access to a catalog or control level access to a data set to examine it. Master level access to the master catalog is also sufficient to examine a user catalog.

How to Run EXAMINE

During an EXAMINE run, the following considerations for sharing data sets apply:

- No users should be open to the data set while EXAMINE runs.
- EXAMINE issues the message “IDC01723I ERRORS MAY BE DUE TO CONCURRENT ACCESS” if it detects any errors; the data set might have been open for output during testing. This message does not necessarily indicate that the reported errors are because of concurrent access.
- When you run EXAMINE against a catalog, concurrent access might have occurred without the message being issued. Because system access to the catalog can be difficult to stop, you should not run jobs that would cause an update to the catalog.

For further considerations for data set sharing, see “Chapter 13. Sharing VSAM Data Sets” on page 195.

Deciding to Run INDEXTEST, DATATEST, or Both Tests

INDEXTEST reads the entire index component of the KSDS cluster.

DATATEST reads the sequence set from the index component and the entire data component of the KSDS cluster. So, it should take considerably more time and more system resources than INDEXTEST.

If you are using EXAMINE to document an error in the data component, run both tests. If you are using EXAMINE to document an error in the index component, it is usually not necessary to run DATATEST.

If you are using EXAMINE to confirm a data set’s integrity, your decision to run one or both tests depends on the time and resources available.

Skipping DATATEST on Major INDEXTEST Errors

If you decide to run both tests (INDEXTEST and DATATEST), INDEXTEST runs first. If INDEXTEST finds major structural errors, DATATEST does not run, even though you requested it. This gives you a chance to review the output from INDEXTEST and to decide whether you need to run DATATEST.

If you want to run DATATEST unconditionally, you must specify the NOINDEXTEST parameter in the EXAMINE command to bypass INDEXTEST.

Examining a User Catalog

You must have master-level access for either the user catalog being examined or for the master catalog. If you have master-level access for the master catalog, the self-describing records in the user catalog will not be read during open. If you have master-level access only for the user catalog being examined, the catalog self-describing records will be read during open.

If the master catalog is protected by RACF or an equivalent product and you do not have alter authority for the catalog, a message can be issued indicating an

authorization failure when the check indicated above is made. This is normal, and, if you have master level access to the catalog being examined, the examination can continue.

Recommendation: When you analyze a catalog, use the VERIFY command before you use the EXAMINE command.

Understanding Message Hierarchy

Messages describing errors or inconsistencies are generated during EXAMINE processing as that condition is detected. The detection of an error condition can result in the generation of many messages. There are five distinct types of EXAMINE error messages:

1. **Status and Statistical Messages.** This type of message tells you the status of the EXAMINE operation, such as the beginning and completion of each test. It provides general statistical data, such as the number of data records, the percentage of free space in data control intervals (CIs), and the number of deleted CIs. The four status messages are IDC01700I, IDC01701I, IDC01709I, and IDC01724I. The five statistical messages are IDC01708I, IDC01710I, IDC01711I, IDC01712I, and IDC01722I.
2. **Supportive (Informational) Messages.** Supportive messages (all remaining IDC0-type messages) issued by EXAMINE clarify individual data set structural errors and provide additional information pertinent to an error.
3. **Individual Data Set Structural Error Messages.** The identification of an error is always reported by an individual data set structural error (IDC1-type) message that can be immediately followed by one or more supportive messages.
4. **Summary Error Messages.** One or more summary error (IDC2-type) messages are generated at the completion of either INDEXTEST or DATATEST to categorize all individual data set structural error (IDC1-type) messages displayed during the examination. The summary error message represents the final analysis of the errors found, and the user should follow the course of recovery action as prescribed by the documentation.
5. **Function-Not-Performed Messages.** Function-not-performed messages (all of the IDC3-type messages) indicate that the function you requested cannot be successfully performed by EXAMINE. In each case, the test operation terminates before the function completes.

   Function-not-performed messages are issued for a variety of reasons, some of that follows:

   - An invalid request (such as an attempt to examine an entry-sequenced data set (ESDS))
   - A physical I/O error in a data set
   - A system condition (such as insufficient storage)
   - A system error (such as an OBTAIN DSCB failed)
   - An error found during INDEXTEST (see “Skipping DATATEST on Major INDEXTEST Errors” on page 236).

Controlling Message Printout

Use the ERRORLIMIT parameter in the EXAMINE command to suppress supportive and individual data set structural error messages during an EXAMINE run. This parameter indicates the number of these error messages to print. When EXAMINE reaches this number of errors, it stops issuing error messages but continues to scan the data set. ERRORLIMIT (0) means that none of these messages will be

printed. When you do not specify the ERRORLIMIT parameter (the default condition), all supportive and individual data set structural error messages are printed. Note that the status and statistical messages, summary messages, and function-not-performed messages are not under the control of ERRORLIMIT, and print regardless of the ERRORLIMIT settings. The ERRORLIMIT parameter is used separately by INDEXTEST and DATATEST. For more information about using this parameter see *z/OS DFSMS Access Method Services*.

Samples of Output from EXAMINE Runs

This section shows examples of output from EXAMINE runs.

INDEXTEST and DATATEST Tests of an Error-Free Data Set

In this run, INDEXTEST and DATATEST are both executed successfully against an error-free data set. The first four messages tell us the status of the two EXAMINE tests, that performed with no errors detected. The next five messages then summarize component statistics as revealed by the DATATEST.

```
IDCAMS SYSTEM SERVICES

    EXAMINE NAME(EXAMINE.KD05) -
      INDEXTEST -
      DATATEST

IDC01700I INDEXTEST BEGINS
IDC01724I INDEXTEST COMPLETES NO ERRORS DETECTED
IDC01701I DATATEST BEGINS
IDC01709I DATATEST COMPLETES NO ERRORS DETECTED

IDC01708I 45 CONTROL INTERVALS ENCOUNTERED
IDC01710I DATA COMPONENT CONTAINS 1000 RECORDS
IDC01711I DATA COMPONENT CONTAINS 0 DELETED CONTROL INTERVALS
IDC01712I MAXIMUM LENGTH DATA RECORD CONTAINS 255 BYTES
IDC01722I 65 PERCENT FREE SPACE

IDC0001I FUNCTION COMPLETED, HIGHEST CONDITION CODE WAS 0
```

INDEXTEST and DATATEST Tests of a Data Set with a Structural Error

The user intended to run both tests, INDEXTEST and DATATEST, but INDEXTEST found an error in the sequence set. From the messages, we learn the following:

- A structural problem was found in the index component of the KSDS cluster.
- The current index level is 1 (that is the sequence set).
- The index control interval (beginning at the relative byte address of decimal 23552) where it found the error is displayed.
- The error is located at offset hexadecimal 10 into the control interval.

Because of this severe INDEXTEST error, DATATEST did not run in this particular case.

```
IDCAMS SYSTEM SERVICES

  EXAMINE NAME(EXAMINE.KD99) INDEXTEST DATATEST
IDC01700I INDEXTEST BEGINS
IDC11701I STRUCTURAL PROBLEM FOUND IN INDEX
IDC01707I CURRENT INDEX LEVEL IS 1
IDC01720I INDEX CONTROL INTERVAL DISPLAY AT RBA 23552 FOLLOWS
000000 01F90301 00000000 00005E00 00000000  02000021 010701BC 2D2C2B2A 29282726  X.9........;......................X
000020 25000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  X...............................X
000040 00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  X...............................X
000060 00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  X...............................X
000080 00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  X...............................X
0000A0 00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  X...............................X
0000C0 00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  X...............................X
0000E0 00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  X...............................X
000100 00000000 0000F226 01240007 F7F12502  23F82601 22F72601 21F42601 20F32601  X......2.....71...8...7...4...3..X
000120 1FF6F025 021E001B F525011D F626011C  F526011B F226011A F5F12502 19F82601  X.60.....5...6...5...2...51...8..X
000140 18001CF4 F7250217 F4260116 F3260115  F4F02502 14260013 F6260112 001BF3F5  X...47...4...3...40......6.....35X
000160 250211F2 260110F3 F125020F F826010E  F726010D F426010C 001CF2F3 25020BF2  X...2...31...8...7...4.....23...2X
000180 F025020A 260009F6 260108F5 260107F2  26010600 40F0F0F0 F0F0F0F0 F0F0F0F0  X0......6...5...2.... 00000000000X
0001A0 F0F0F0F0 F0F0F0F0 F0F0F0F0 F0F0F0F0  F0F0F0F0 F0F0F0F0 F0F0F1F1 002705F8  X0000000000000000000000000011...8X
0001C0 260104F7 260103F4 260102F3 260101F0  F0F0F0F0 F0F0F0F0 F0F0F0F0 F0F0F0F0  X...7...4...3...00000000000000000X
0001E0 F0F0F0F0 F0F0F0F0 F0F0F0F0 F0F0F0F0  F0F0F0F0 F0F0F027 000001F9 01F90000  X0000000000000000000000.....9.9..X
IDC01714I ERROR LOCATED AT OFFSET 00000010
IDC21701I MAJOR ERRORS FOUND BY INDEXTEST
IDC31705I DATATEST NOT PERFORMED DUE TO SEVERE INDEXTEST ERRORS
IDC3003I FUNCTION TERMINATED.  CONDITION CODE IS 12

IDC0002I IDCAMS PROCESSING COMPLETE.  MAXIMUM CONDITION CODE WAS 12
```

INDEXTEST and DATATEST Tests of a Data Set with a Duplicate Key Error

The user intended to run both tests, INDEXTEST and DATATEST. INDEXTEST began and completed successfully. DATATEST looked at the data component of the KSDS cluster and found a “duplicate key” error.

EXAMINE then displayed the prior key (11), the data control interval at relative byte address decimal 512, and the offset address hexadecimal 9F into the control interval where the duplicate key was found.

Checking VSAM Key-Sequenced Data Set Clusters for Structural Errors

```
IDCAMS SYSTEM SERVICES

  EXAMINE NAME(EXAMINE.KD99) INDEXTEST DATATEST
IDC01700I INDEXTEST BEGINS
IDC01724I INDEXTEST COMPLETE NO ERRORS DETECTED
IDCC17OLI DATATEST COMPLETE
IDC11741I DUPLICATE CONSECUTIVE KEYS FOUND
IDC01717I DATA KEYS FOLLOW
000000 F0F0F0F0 F0F0F0F0 F0F0F0F0 F0F0F0F0  F0F0F0F0 F0F0F0F0 F0F0F0F0 F0F0F0F0  X0000000000000000000000000000000X
000020 F0F0F0F0 F0F0F1F1                                                         X00000011                        X

DIC01713I DATA CONTROL INTERVAL DISPLAY AT RBA 512 FOLLOWS
000000 00000000 0000D0C1 AB33F0F0 F0F0F0F0  F0F0F0F0 F0F0F0F0 F0F0F0F0 F0F0F0F0  X.......A..0000000000000000000000X
000020 F0F0F0F0 F0F0F0F0 F0F0F0F0 F0F0F0F0  F1F1C3C3 C3C3C3C3 C3C3C3C3 C3C3C3C3  X0000000000000000011CCCCCCCCCCCCCCX
000040 C3C3C3C3 C3C3C3C3 C3C3C3C3 C3C3C3C3  C3C3C3C3 C3C3C3C3 C3C3C3C3 C3C3C3C3  XCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCX
000060 C3C3C3C3 C3C3C3C3 C3C3C3C3 C3C3C3C3  C3C3C3C3 C3C3C3C3 C3C3C3C3 C3C3C3C3  XCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCX
000080 C3C3C3C3 C3C3C3C3 C3C3C3C3 C3C3C3C3  C3C3C3C3 C3000000 0000006E E065DEF0  XCCCCCCCCCCCCCCCCCCCCCC......>...0X
0000A0 F0F0F0F0 F0F0F0F0 F0F0F0F0 0F0F0F0F  F0F0F0F0 F0F0F0F0 F0F0F0F0 F0F0F0F0  X0000000000000000000000000000000X
0000C0 F0F0F0F0 F0F1F1C4 C4C4C4C4 C4C4C4C4  C4C4C4C4 C4C4C4C4 C4C4C4C4 C4C4C4C4  X0000011DDDDDDDDDDDDDDDDDDDDDDDDDX
0000E0 C4C4C4C4 C4C4C4C4 C4C4C4C4 C4C4C4C4  C4C4C4C4 00000000 000055E2 0706F0F0  XDDDDDDDDDDDDDDDDDDDD.......S..00X
000100 F0F0F0F0 F0F0F0F0 F0F0F0F0 0F0F0F0F  F0F0F0F0 F0F0F0F0 F0F0F0F0 F0F0F0F0  X0000000000000000000000000000000X
000120 F0F0F0F0 F3F1C1C1 C1C1C100 00000000  00000000 00000000 00000000 00000000  X000031AAAAA.....................X
000140 00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  X................................X
000160 00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  X................................X
000180 00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  X................................X
0001A0 00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  X................................X
0001C0 00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  X................................X
0001E0 00000000 00000000 00000000 00000000  00000000 00370000 5F000095 012B00C8  X........................-......HX
IDC01714I ERROR LOCATED AT OFFSET 00000C9F
IDC21703I MAJOR ERRORS FOUND BY DATATEST
IDC0001I FUNCTION COMPLETED.  HIGHEST CONDITION CODE WAS 8

IDC0002I IDCAMS PROCESSING COMPLETE.  MAXIMUM CONDITION CODE WAS 8
```

Chapter 17. Coding VSAM User-Written Exit Routines

This chapter covers the following topics.

Topic	Page
Guidelines for Coding Exit Routines	241
EODAD Exit Routine to Process End-of-Data	243
EXCEPTIONEXIT Exit Routine	244
JRNAD Exit Routine to Journalize Transactions	245
LERAD Exit Routine to Analyze Logical Errors	252
RLSWAIT Exit Routine	253
SYNAD Exit Routine to Analyze Physical Errors	255
UPAD Exit Routine for User Processing	257
User-Security-Verification Routine	260

Guidelines for Coding Exit Routines

VSAM user-written routines can be supplied to do the following tasks:

- Analyze logical errors
- Analyze physical errors
- Perform end-of-data processing
- Record transactions made against a data set
- Perform special user processing
- Perform wait user processing
- Perform user-security verification.

VSAM user-written exit routines are identified by macro parameters in access methods services commands.

You can use the EXLST VSAM macro to create an exit list. EXLST parameters EODAD, JRNAD, LERAD, SYNAD and UPAD are used to specify the addresses of your user-written routines. Only the exits marked active are executed. See *z/OS DFSMS Macro Instructions for Data Sets* for information about the EXLST macro.

You can use access methods services commands to specify the addresses of user-written routines to perform exception processing and user-security verification processing. See *z/OS DFSMS Access Method Services* for information about exits from access methods services commands.

Table 11 on page 242 shows the exit locations available from VSAM.

© Copyright IBM Corp. 1987, 2002

Coding VSAM User-Written Exit Routines

Table 11. VSAM User-Written Exit Routines

Exit Routine	When Available	Where Specified
End-of-data-set	When no more sequential records or blocks are available	EODAD parameter of EXLST macro
Exception exit	After an uncorrectable input/output error	EXCEPTIONEXIT parameter on access methods services commands
Journalize transactions against a data set	After an input/output completion or error, change to buffer contents, shared or nonshared request, program issues GET, PUT, ERASE, shift in data in a control interval	JRNAD parameter of EXLST macro
Analyze logical errors	After an uncorrectable logical error	LERAD parameter of EXLST macro
Wait	For non-cross-memory mode callers using RLS	RLSWAIT parameter of EXLST macro
Error analysis	After an uncorrectable input/output error	SYNAD parameter of EXLST macro
User processing	WAIT for I/O completion or for a serially reusable request	UPAD parameter of EXLST macro
User security verification	When opening a VSAM data set	AUTHORIZATION parameter on access methods services commands

Programming Considerations

To code VSAM user exit routines, you should be familiar with the contents and have available the *z/OS DFSMS Macro Instructions for Data Sets* and *z/OS DFSMS Access Method Services* manuals.

Usually, you should observe the following guidelines in coding a routine:

- Code your routine reentrant
- Save and restore registers (see individual routines for other requirements).
- Be aware of registers used by the VSAM request macros.
- Be aware of the addressing mode (24 bit or 31 bit) in which your exit routine will receive control.
- Determine if VSAM or your program should load the exit routine.

If the exit routine is used by a program that is doing asynchronous processing with multiple request parameter lists (RPL) or, if the exit routine is used by more than one data set, it must be coded so that it can handle an entry made before the previous entry's processing is completed. Saving and restoring registers in the exit routine, or by other routines called by the exit routine, is best accomplished by coding the exit routine reentrant. Another way of doing this is to develop a technique for associating a unique save area with each RPL.

If the LERAD, EODAD, or SYNAD exit routine reuses the RPL passed to it, you should be aware that:

- The exit routine is called again if the request issuing the reused RPL results in the same exception condition that caused the exit routine to be entered originally.
- The original feedback code is replaced with the feedback code that indicates the status of the latest request issued against the RPL. If the exit routine returns to VSAM, VSAM (when it returns to the user's program) sets register 15 to also indicate the status of the latest request.

- JRNAD, UPAD, exception exits are extensions of VSAM and therefore must return to VSAM in the same processing mode (that is, cross-memory, SRB, or task mode) in which they were entered.

A user exit that is loaded by VSAM will be invoked in the addressing mode specified when the module was link edited. A user exit that is not loaded by VSAM will receive control in the same addressing mode as the issuer of the VSAM record-management, OPEN, or CLOSE request that causes the exit to be taken. It is the user's responsibility to ensure that the exit is written for the correct addressing mode.

Your exit routine can be loaded within your program or by using the JOBLIB or STEPLIB with the DD statement to point to the library location of your exit routine.

Return to a Main Program

Six exit routines can be entered when your main program issues a VSAM request macro (GET, PUT, POINT, and ERASE) and the macro has not completed: LERAD, SYNAD, EODAD, UPAD, RLSWAIT or the EXCEPTIONEXIT routine. Entering the LERAD, SYNAD, EODAD, or EXCEPTIONEXIT indicates that the macro failed to complete successfully. When your exit routine completes its processing, it can return to your main program in one of two ways:

- The exit routine can return to VSAM (via the return address in register 14); VSAM then returns to your program at the instruction following the VSAM request macro that failed to complete successfully. This is the easier way to return to your program.

  If your error recovery and correction process needs to reissue the failing VSAM macro against the RPL to retry the failing request or to correct it:

  – Your exit routine can correct the RPL (using MODCB), then set a switch to indicate to your main program that the RPL is now ready to retry. When your exit routine completes processing, it can return to VSAM (via register 14), which returns to your main program. Your main program can then test the switch and reissue the VSAM macro and RPL.

  – Your exit routine can issue a GENCB macro to build an RPL, and then copy the RPL (for the failing VSAM macro) into the newly built RPL. At this point, your exit routine can issue VSAM macros against the newly built RPL. When your exit routine completes processing, it can return to VSAM (via register 14), which returns to your main program.

- The exit routine can determine the appropriate return point in your program, then branch directly to that point. Note that when VSAM enters your exit routine, none of the registers contains the address of the instruction following the failing macro.

  You are required to use this method to return to your program if, during the error recovery and correction process, your exit routine issued a GET, PUT, POINT, or ERASE macro that refers to the RPL referred to by the failing VSAM macro. (That is, the RPL has been reissued by the exit routine.) In this case, VSAM has lost track of its reentry point to your main program. If the exit routine returns to VSAM, VSAM issues an error return code.

EODAD Exit Routine to Process End-of-Data

VSAM exits to an EODAD routine when an attempt is made to sequentially retrieve or point to a record beyond the last record in the data set (one with the highest key for keyed access and the one with the highest RBA for addressed access). VSAM doesn't take the exit for direct requests that specify a record beyond the end. If the EODAD exit isn't used, the condition is considered a logical error (FDBK code

X'04') and can be handled by the LERAD routine, if one is supplied. See “LERAD Exit Routine to Analyze Logical Errors” on page 252.

Register Contents

Table 12 gives the contents of the registers when VSAM exits to the EODAD routine.

Table 12. Contents of Registers at Entry to EODAD Exit Routine

Register	Contents
0	Unpredictable.
1	Address of the RPL that defines the request that occasioned VSAM’s reaching the end of the data set. The register must contain this address if you return to VSAM.
2-13	Unpredictable. Register 13, by convention, contains the address of your processing program’s 72-byte save area, which must not be used as a save area by the EODAD routine if it returns control to VSAM.
14	Return address to VSAM.
15	Entry address to the EODAD routine.

Programming Considerations

The typical actions of an EODAD routine are to:

- Examine RPL for information you need, for example, type of data set
- Issue completion messages
- Close the data set
- Terminate processing without returning to VSAM

If the routine returns to VSAM and another GET request is issued for access to the data set, VSAM exits to the LERAD routine.

If a processing program retrieves records sequentially with a request defined by a chain of RPLs, the EODAD routine must determine whether the end of the data set was reached for the first RPL in the chain. If not, then one or more records have been retrieved but not yet processed by the processing program.

The type of data set whose end was reached can be determined by examining the RPL for the address of the access method control block that connects the program to the data set and testing its attribute characteristics.

If the exit routine issues GENCB, MODCB, SHOWCB, or TESTCB and returns to VSAM, it must provide a save area and restore registers 13 and 14, which are used by these macros.

When your EODAD routine completes processing, return to your main program as described in “Return to a Main Program” on page 243.

EXCEPTIONEXIT Exit Routine

You can provide an exception exit routine to monitor I/O errors associated with a data set. You specify the name of your routine via the access method services DEFINE command using the EXCEPTIONEXIT parameter to specify the name of your user-written exit routine.

Register Contents

Table 13 gives the contents of the registers when VSAM exits to the EXCEPTIONEXIT.

Table 13. Contents of Registers at Entry to EXCEPTIONEXIT Routine

Register	Contents
0	Unpredictable.
1	Address of the RPL that contains a feedback return code and the address of a message area, if any.
2-13	Unpredictable. Register 13, by convention, contains the address of your processing program's 72-byte save area, which must not be used by the routine if it returns control to VSAM.
14	Return address to VSAM.
15	Entry address to the exception exit routine.

Programming Considerations

The exception exit is taken for the same errors as a SYNAD exit. If you have both an active SYNAD routine and an EXCEPTIONEXIT routine, the exception exit routine is processed first.

The exception exit is associated with the attributes of the data set (specified by the DEFINE) and is loaded on every call. Your exit must reside in the LINKLIB and the exit cannot be called when VSAM is in cross-memory mode.

When your exception exit routine completes processing, return to your main program as described in "Return to a Main Program" on page 243.

See *z/OS DFSMS Access Method Services* for information about how exception exits are established, changed, or nullified.

JRNAD Exit Routine to Journalize Transactions

A JRNAD exit routine can be provided to record transactions against a data set, to keep track of changes in the RBAs of records, and to monitor control interval splits. It is only available for VSAM shared resource buffering. When using the JRNAD exit routine with compressed data sets, all RBA's and data length values returned represent compressed data. For shared resources, you can use a JRNAD exit routine to deny a request for a control interval split. VSAM takes the JRNAD exit each time one of the following occurs:

- The processing program issues a GET, PUT, or ERASE
- Data is shifted right or left in a control interval or is moved to another control interval to accommodate a record's being deleted, inserted, shortened, or lengthened
- An I/O error occurs
- An I/O completion occurs
- A shared or nonshared request is received
- The buffer contents are to be changed

Restriction: The JRNAD exit is not supported by RLS.

Register Contents

Table 14 gives the contents of the registers when VSAM exits to the JRNAD routine.

Table 14. Contents of Registers at Entry to JRNAD Exit Routine

Register	Contents
0	Byte 0—the subpool-id token created by a BLDVRP request. Bytes 2 - 3—the relative buffer number, that is, the buffer array index within a buffer pool.
1	Address of a parameter list built by VSAM.
2-3	Unpredictable.
4	Address of buffer control block (BUFC).
5-13	Unpredictable.
14	Return address to VSAM.
15	Entry address to the JRNAD routine.

Programming Considerations

If the JRNAD is taken for I/O errors, a journal exit can zero out, or otherwise alter, the physical-error return code, so that a series of operations can continue to completion, even though one or more of the operations failed.

The contents of the parameter list built by VSAM, pointed to by register 1, can be examined by the JRNAD exit routine, which is described in Table 15 on page 250.

If the exit routine issues GENCB, MODCB, SHOWCB, or TESTCB, it must restore register 14, which is used by these macros, before it returns to VSAM.

If the exit routine uses register 1, it must restore it with the parameter list address before returning to VSAM. (The routine must return for completion of the request that caused VSAM to exit.)

The JRNAD exit must be indicated as active before the data set for which the exit is to be used is opened, and the exit must not be made inactive during processing. If you define more than one access method control block for a data set and want to have a JRNAD routine, the first ACB you open for the data set must specify the exit list that identifies the routine.

When the data set being processed is extended addressable, the JRNAD exits dealing with RBAs are not taken or are restricted due to the increase in the size of the field required to provide addressability to RBAs which may be greater than 4 GB. The restrictions are for the entire data set without regard to the specific RBA value.

Journalizing Transactions

For journalizing transactions (when VSAM exits because of a GET, PUT, or ERASE), you can use the SHOWCB macro to display information in the request parameter list about the record that was retrieved, stored, or deleted (FIELDS=(AREA,KEYLEN,RBA,RECLEN), for example). You can also use the TESTCB macro to find out whether a GET or a PUT was for update (OPTCD=UPD).

If your JRNAD routine only journals transactions, it should ignore reason X'0C' and return to VSAM; conversely, it should ignore reasons X'00', X'04', and X'08' if it records only RBA changes.

RBA Changes

For recording RBA changes, you must calculate how many records there are in the data being shifted or moved, so you can keep track of the new RBA for each. If all the records are the same length, you calculate the number by dividing the record length into the number of bytes of data being shifted. If record length varies, you can calculate the number by using a table that not only identifies the records (by associating a record's key with its RBA), but also gives their length.

You should provide a routine to keep track of RBA changes caused by control interval and control area splits. RBA changes that occur through keyed access to a key-sequenced data set must also be recorded if you intend to process the data set later by direct-addressed access.

Control Interval Splits

Some control interval splits involve data being moved to two new control intervals, and control area splits normally involve many control intervals' contents being moved. In these cases, VSAM exits to the JRNAD routine for each separate movement of data to a new control interval.

You might also want to use the JRNAD exit to maintain shared or exclusive control over certain data or index control intervals; and in some cases, in your exit routine you can reject the request for certain processing of the control intervals. For example, if you used this exit to maintain information about a data set in a shared environment, you might reject a request for a control interval or control area split because the split might adversely affect other users of the data set.

Figure 34 is a skeleton program USERPROG with a user exit routine USEREXIT. It demonstrates the use of the JRNAD exit routine to cancel a request for a control interval or control area split.

```
USERPROG  CSECT
          SAVE(R14,R12)       Standard entry code
          .
          .
          .
          BLDVRP BUFFERS=(512(3)),  Build resource pool              X
                KEYLEN=4,                                            X
                STRNO=4,                                             X
                TYPE=LSR,                                            X
                SHRPOOL=1,                                           X
                RMODE31=ALL
          OPEN (DIRACB)       Logically connect KSDS1
          .
          .
          .
          PUT RPL=DIRRPL      This PUT causes the exit routine USEREXIT
                              to be taken with an exit code X'50' if
                              there is a CI or CA split
          LTR R15,R15         Check return code from PUT
          BZ NOCANCEL         Retcode = 0 if USEREXIT did not cancel
                                          CI/CA split
                                      = 8 if cancel was issued, if
                                          we know a CI or CA split
                                          occurred
          .
          .                   Process the cancel situation
          .
NOCANCEL  .                   Process the noncancel situation
          .
          .
          CLOSE (DIRACB)      Disconnect KSDS1
          DLVRP TYPE=LSR,SHRPOOL=1  Delete the resource pool
          .
          .
          .
          RETURN              Return to caller.
          .
          .
          .
DIRACB    ACB AM=VSAM,                                               X
          DDNAME=KSDS1,                                              X
          BUFND=3,                                                   X
          BUFNI=2,                                                   X
          MACRF=(KEY,DDN,SEQ,DIR,OUT,LSR),                           X
          SHRPOOL=1,                                                 X
          EXLST=EXITLST
```

Figure 34. Example of a JRNAD Exit (Part 1 of 2)

```
*
DIRRPL    RPL AM=VSAM,                                              X
          ACB=DIRACB,                                               X
          AREA=DATAREC,                                             X
          AREALEN=128,                                              X
          ARG=KEYNO,                                                X
          KEYLEN=4,                                                 X
          OPTCD=(KEY,DIR,FWD,SYN,NUP,WAITX),                        X
          RECLEN=128
*
DATAREC   DC  CL128'DATA RECORD TO BE PUT TO KSDS1'
KEYNO     DC  F'0'            Search key argument for RPL
EXITLST   EXLST AM=VSAM,JRNAD=(JRNADDR,A,L)
JRNADDR   DC  CL8'USEREXIT'   Name of user exit routine
          END                 End of USERPROG


USEREXIT  CSECT               On entry to this exit routine, R1 points
                              to the JRNAD parameter list and R14 points
                              back to VSAM.
          .
          .                   Nonstandard entry code -- need not save
          .                   the registers at caller's save area and,
          .                   since user exit routines are reentrant for
          .                   most applications, save R1 and R14 at some
          .                   registers only if R1 and R14 are to be
          .                   destroyed
          .
          CLI 20(R1),X'50'    USEREXIT called because of CI/CA split?
          BNE EXIT            No.  Return to VSAM
          MVI 21(R1),X'8C'    Tell VSAM that user wants to cancel split
          .
          .
          .
EXIT      .                   Nonstandard exit code -- restore R1 and
          .                   R14 from save registers
          BR R14              Return to VSAM which returns to USERPROG
                              if cancel is specified
          END                 End of USEREXIT
```

Figure 34. Example of a JRNAD Exit (Part 2 of 2)

Parameter List

The parameter list built by VSAM contains reason codes to indicate why the exit was taken, and also locations where you can specify return codes for VSAM to take or not take an action on returning from your routine. The information provided in the parameter list varies depending on the reason the exit was taken. Table 15 shows the contents of the parameter list.

The parameter list will reside in the same area as the VSAM control blocks, either above or below the 16 MB line. For example, if the VSAM data set was opened and the ACB stated RMODE31=CB, the exit parameter list will reside above the 16 MB line. To access a parameter list that resides above the 16 MB line, you will need to use 31-bit addressing.

Table 15. Contents of Parameter List Built by VSAM for the JRNAD Exit

Offset	Bytes	Description
0(X'0')	4	Address of the RPL that defines the request that caused VSAM to exit to the routine.
4(X'4')	4	Address of a 5-byte field that identifies the data set being processed. This field has the format: **4 bytes** Address of the access method control block specified by the RPL that defines the request occasioned by the JRNAD exit. **1 byte** Indication of whether the data set is the data (X'01') or the index (X'02') component.
8(X'8')	4	Variable, depends on the reason indicator at offset 20: **Offset 20** **Contents at offset 8** **X'0C'** The RBA of the first byte of data that is being shifted or moved. **X'20'** The RBA of the beginning of the control area about to be split. **X'24'** The address of the I/O buffer into which data was going to be read. **X'28'** The address of the I/O buffer from which data was going to be written. **X'2C'** The address of the I/O buffer that contains the control interval contents that are about to be written. **X'30'** Address of the buffer control block (BUFC) that points to the buffer into which data is about to be read under exclusive control. **X'34'** Address of BUFC that points to the buffer into which data is about to be read under shared control. **X'38'** Address of BUFC that points to the buffer which is to be acquired in exclusive control. The buffer is already in the buffer pool. **X'3C'** Address of the BUFC that points to the buffer which is to be built in the buffer pool in exclusive control. **X'40'** Address of BUFC which points to the buffer whose exclusive control has just been released. **X'44'** Address of BUFC which points to the buffer whose contents have been made invalid. **X'48'** Address of the BUFC which points to the buffer into which the READ operation has just been completed. **X'4C'** Address of the BUFC which points to the buffer from which the WRITE operation has just been completed.

Table 15. Contents of Parameter List Built by VSAM for the JRNAD Exit (continued)

Offset	Bytes	Description
12(X'C')	4	Variable, depends on the reason indicator at offset 20: **Offset 20** **Contents at offset 12** **X'0C'** The number of bytes of data that is being shifted or moved (this number does not include free space, if any, or control information, except for a control area split, when the entire contents of a control interval are moved to a new control interval.) **X'20'** Unpredictable. **X'24'** Unpredictable. **X'28'** Bits 0 - 31 correspond with transaction IDs 0 - 31. Bits set to 1 indicate that the buffer that was being written when the error occurred was modified by the corresponding transactions. You can set additional bits to 1 to tell VSAM to keep the contents of the buffer until the corresponding transactions have modified the buffer. **X'2C'** The size of the control interval whose contents are about to be written. **X'30'** Zero. **X'34'** Zero. **X'38'** Zero. **X'3C'** Size of the buffer which is to be built in the buffer pool in exclusive control. **X'48'** Size of the buffer into which the READ operation has just been completed. **X'4C'** Size of the buffer from which the WRITE operation has just been completed.
16(X'10')	4	Variable, depends on the reason indicator at offset 20: **Offset 20** **Contents at offset 16** **X'0C'** The RBA of the first byte to which data is being shifted or moved. **X'20'** The RBA of the last byte in the control area about to be split. **X'24'** The fourth byte contains the physical error code from the RPL FDBK field. You use this fullword to communicate with VSAM. Setting it to 0 indicates that VSAM is to ignore the error, bypass error processing, and let the processing program continue. Leaving it nonzero indicates that VSAM is to continue as usual: terminate the request that occasioned the error and proceed with error processing, including exiting to a physical error analysis routine. **X'28'** Same as for X'24'. **X'2C'** The RBA of the control interval whose contents are about to be written. **X'48'** Unpredictable. **X'4C'** Unpredictable.

Table 15. Contents of Parameter List Built by VSAM for the JRNAD Exit (continued)

Offset	Bytes	Description
20(X'14')	1	Indication of the reason VSAM exited to the JRNAD routine: **X'00'** GET request. **X'04'** PUT request. **X'08'** ERASE request. **X'0C'** RBA change. **X'10'** Read spanned record segment. **X'14'** Write spanned record segment. **X'18'** Reserved. **X'1C'** Reserved. The following codes are for shared resources only: **X'20'** Control area split. **X'24'** Input error. **X'28'** Output error. **X'2C'** Buffer write. **X'30'** A data or index control interval is about to be read in exclusive control. **X'34'** A data or index control interval is about to be read in shared status. **X'38'** Acquire exclusive control of a control interval already in the buffer pool. **X'3C'** Build a new control interval for the data set and hold it in exclusive control. **X'40'** Exclusive control of the indicated control interval already has been released. **X'44'** Contents of the indicated control interval have been made invalid. **X'48'** Read completed. **X'4C'** Write completed. **X'50'** Control interval or control area split. **X'54'–X'FF'** Reserved.
21(X'15')	1	JRNAD exit code set by the JRNAD exit routine. Indication of action to be taken by VSAM after resuming control from JRNAD (for shared resources only): **X'80'** Do not write control interval. **X'84'** Treat I/O error as no error. **X'88'** Do not read control interval. **X'8C'** Cancel the request for control interval or control area split.

LERAD Exit Routine to Analyze Logical Errors

A LERAD exit routine should examine the feedback field in the request parameter list to determine what logical error occurred. What the routine does after determining the error depends on your knowledge of the kinds of things in the processing program that can cause the error.

VSAM does not call the LERAD exit if the RPL feedback code is 64.

Register Contents

Table 16 gives the contents of the registers when VSAM exits to the LERAD exit routine.

Table 16. Contents of Registers at Entry to LERAD Exit Routine

Register	Contents
0	Unpredictable.
1	Address of the RPL that contains the feedback field the routine should examine. The register must contain this address if you return to VSAM.
2-13	Unpredictable. Register 13, by convention, contains the address of your processing program's 72-byte save area, which must not be used as a save area by the LERAD routine if the routine returns control to VSAM.
14	Return address to VSAM.
15	Entry address to the LERAD routine. The register does not contain the logical-error indicator.

Programming Considerations

The typical actions of a LERAD routine are:

1. Examine the feedback field in the RPL to determine what error occurred.
2. Determine what action to take based on error.
3. Close the data set.
4. Issue completion messages.
5. Terminate processing and exit VSAM or return to VSAM.

If the LERAD exit routine issues GENCB, MODCB, SHOWCB, or TESTCB and returns to VSAM, it must restore registers 1, 13, and 14, which are used by these macros. It must also provide two save areas; one, whose address should be loaded into register 13 before the GENCB, MODCB, SHOWCB, or TESTCB is issued, and the second, to separately store registers 1, 13, and 14.

If the error cannot be corrected, close the data set and either terminate processing or return to VSAM.

If a logical error occurs and no LERAD exit routine is provided (or the LERAD exit is inactive), VSAM returns codes in register 15 and in the feedback field of the RPL to identify the error.

When your LERAD exit routine completes processing, return to your main program as described in "Return to a Main Program" on page 243.

RLSWAIT Exit Routine

The RLSWAIT exit is entered at the start of the record management request and the request is processed asynchronously under a separate VSAM execution unit. If a UPAD is specified, RLS ignores it.

The exit can do its own wait processing associated with the record management request that is being asynchronously executed. When the record management request is complete, VSAM will post the ECB that the user specified in the RPL.

For RLS, the RLSWAIT exit is entered only for a request wait, never for a resource- or I/O- wait or post as with non-RLS VSAM UPAD.

The RLSWAIT exit is optional. It is used by applications that cannot tolerate VSAM suspending the execution unit that issued the original record management request. The RLSWAIT exit is required for record management request issued in cross memory mode.

RLSWAIT should be specified on each ACB which requires the exit. If the exit is not specified on the ACB via the EXLST, there is no RLSWAIT exit processing for record management requests associated with that ACB. This differs from non-RLS VSAM where the UPAD exit is associated with the control block structure so that all ACBs connected to that structure inherit the exit of the first connector.

To activate RLSWAIT exit processing for a particular record management request, the RPL must specify OPTCD=(SYN,WAITX). RLSWAIT is ignored if the request is asynchronous.

Register Contents

Table 17 gives the contents of the registers when RLSWAIT is entered in 31 bit mode.

Table 17. Contents of Registers for RLSWAIT Exit Routine

Register	Contents
1	Address of the user RPL. If a chain of RPLs was passed on the original record management request, this is the first RPL in the chain.
12	Reserved and must be the same on exit as on entry.
13	Reserved and must be the same on exit as on entry.
14	Return address. The exit must return to VSAM using this register.
15	Address of the RLSWAIT exit.

The RLSWAIT exit must conform to the following restrictions:

- The exit must return to VSAM using register 14 and it must return with the same entry environment. That is, under the same execution unit as on entry and with the same cross memory environment as on entry.
- The exit must not issue any request using the RPL passed in register 1.
- The exit must be reentrant if multiple record management request that use the exit can be concurrently outstanding.

Request Environment

VSAM RLS record management requests must be issued in PRIMARY ASC mode and cannot be issued in home, secondary, or AR ASC mode. The user RPL, EXLST, ACB, must be addressable from primary. Open must have been issued from the same primary address space. VSAM RLS record management request task must be the same as the task that opened the ACB, or the task that opened the ACB must be in the task hierarchy (i.e., the record management task was attached by that task that opened the ACB, or by a task that was attached by the task that opened that ACB). VSAM RLS record management requests must not be issued in SRB mode, and must not have functional recovery routine (FRR) in effect.

If the record management request is issued in cross memory mode, then the caller must be in supervisor state and must specify that an RLSWAIT exit is associated with the request (RPLWAITX = ON). The request must be synchronous.

The RLSWAIT exit is optional for non-cross-memory mode callers.

The RLSWAIT exit, if specified, is entered at the beginning of the request and VSAM processes the request asynchronously under a separate execution unit. VSAM RLS does not enter the RLSWAIT exit for post processing.

VSAM assumes that the ECB supplied with the request is addressable form both home and primary, and that the key of the ECB is the same as the key of the record management caller.

SYNAD Exit Routine to Analyze Physical Errors

VSAM exits to a SYNAD routine if a physical error occurs when you request access to data. It also exits to a SYNAD routine when you close a data set if a physical error occurs while VSAM is writing the contents of a buffer out to direct-access storage.

Register Contents

Table 18 gives the contents of the registers when VSAM exits to the SYNAD routine.

Table 18. Contents of Registers at Entry to SYNAD Exit Routine

Register	Contents
0	Unpredictable.
1	Address of the RPL that contains a feedback return code and the address of a message area, if any. If you issued a request macro, the RPL is the one pointed to by the macro. If you issued an OPEN, CLOSE, or cause an end-of-volume to be done, the RPL was built by VSAM to process an internal request. Register 1 must contain this address if the SYNAD routine returns to VSAM.
2-13	Unpredictable. Register 13, by convention, contains the address of your processing program's 72-byte save area, which must not be used by the SYNAD routine if it returns control to VSAM.
14	Return address to VSAM.
15	Entry address to the SYNAD routine.

Programming Considerations

A SYNAD routine should typically:

- Examine the feedback field in the request parameter list to identify the type of physical error that occurred.
- Get the address of the message area, if any, from the request parameter list, to examine the message for detailed information about the error.
- Recover data if possible.
- Print error messages if uncorrectable error.
- Close the data set.
- Terminate processing.

The main problem with a physical error is the possible loss of data. You should try to recover your data before continuing to process. Input operation (ACB MACRF=IN) errors are generally less serious than output or update operation (MACRF=OUT) errors, because your request was not attempting to alter the contents of the data set.

If the routine cannot correct an error, it might print the physical-error message, close the data set, and terminate the program. If the error occurred while VSAM was closing the data set, and if another error occurs after the exit routine issues a CLOSE macro, VSAM doesn't exit to the routine a second time.

If the SYNAD routine returns to VSAM, whether the error was corrected or not, VSAM drops the request and returns to your processing program at the instruction following the last executed instruction. Register 15 is reset to indicate that there was an error, and the feedback field in the RPL identifies it.

Physical errors affect positioning. If a GET was issued that would have positioned VSAM for a subsequent sequential GET and an error occurs, VSAM is positioned at the control interval next in key (RPL OPTCD=KEY) or in entry (OPTCD=ADR) sequence after the control interval involved in the error. The processing program can therefore ignore the error and proceed with sequential processing. With direct processing, the likelihood of reencountering the control interval involved in the error depends on your application.

If the exit routine issues GENCB, MODCB, SHOWCB, or TESTCB and returns to VSAM, it must provide a save area and restore registers 13 and 14, which are used by these macros.

See "Example of a SYNAD User-Written Exit Routine" for the format of a physical-error message that can be written by the SYNAD routine.

When your SYNAD exit routine completes processing, return to your main program as described in "Return to a Main Program" on page 243.

If a physical error occurs and no SYNAD routine is provided (or the SYNAD exit is inactive), VSAM returns codes in register 15 and in the feedback field of the RPL to identify the error. For a description of these return codes see *z/OS DFSMS Macro Instructions for Data Sets*.

Example of a SYNAD User-Written Exit Routine

The example in Figure 35 on page 257 demonstrates a user-written exit routine. It is a SYNAD exit routine that examines the FDBK field of the RPL checking for the type of physical error that caused the exit. After the checking, special processing can be performed as necessary. The routine returns to VSAM after printing an appropriate error message on SYSPRINT.

```
ACB1     ACB    EXLST=EXITS

EXITS    EXLST  SYNAD=PHYERR
RPL1     RPL    ACB=ACB1,
                MSGAREA=PERRMSG,
                MSGLEN=128

PHYERR   USING  *,15             This routine is nonreentrant.
*                                Register 15 is entry address.
         .
         .                       Save caller's register
                                 (1, 13, 14).
         LA     13,SAVE          Point to routine's save area.
         .
         .                       If register 1=address of RPL1,
         .                       then error did not occur for a
                                 CLOSE.
         SHOWCB RPL=RPL1,
                FIELDS=FDBK,
                AREA=ERRCODE,
                LENGTH=4
*                                Show type of physical error.
         .
         .                       Examine error, perform special
         .                       processing.
         PUT    PRTDCB,ERRMSG    Print physical error message.
         .
         .                       Restore caller's registers
         .                       (1, 13, 14).

         BR     14               Return to VSAM.
         .
         .
         .
ERRCODE  DC     F'0'             RPL reason code from SHOWCB.
PERRMSG  DS     0XL128           Physical error message.

         DS     XL12             Pad for unprintable part.

ERRMSG   DS     XL116            Printable format part of message.
         .
         .
         .
PRTDCB   DCB    .....            QSAM DCB.

SAVE     DS     18F              SYNAD routine's save area.

SAVREG   DS     3F               Save registers 1, 13, 14.
```

Figure 35. Example of a SYNAD Exit Routine

UPAD Exit Routine for User Processing

VSAM calls the UPAD routine only when the request's RPL specifies OPTCD=(SYN, WAITX) and the ACB specifies MACRF=LSR or MACRF=GSR, or MACRF=ICI. VSAM CLOSE can also cause a UPAD exit to be taken to post a record-management request deferred for VSAM internal resource. VSAM takes the UPAD exit to wait for I/O completion or for a serially reusable resource and the UPAD can also be taken to do the corresponding post processing subject to conditions listed in Table 19 on page 258.

If you are executing in cross-memory mode, you must have a UPAD routine and RPL must specify WAITX. Cross-memory mode is described in “Operating in SRB or Cross-Memory Mode” on page 155. The UPAD routine is optional for non-cross-memory mode.

Table 19 describes the conditions in which VSAM calls the UPAD routine for synchronous requests with shared resources. UPAD routines are only taken for synchronous requests with shared resources or improved control interval processing (ICI).

Table 19. Conditions when UPAD Routines are Taken

XMM	Sup. State	UPAD Needed	I/O Wait	I/O Post	Resource Wait	Resource Post
Yes	Yes	Yes	UPAD taken	UPAD taken	UPAD taken	UPAD taken
No	Yes	No	UPAD taken if requested	UPAD not taken even if requested	UPAD taken if requested	UPAD taken if either resource owner or the deferred request runs in XM mode
No	No	No	UPAD taken if requested	UPAD not taken even if requested	UPAD taken if requested	UPAD taken if either resource owner or the deferred request runs in XM mode

Notes:

- You must be in supervisor state when you are in cross-memory mode or SRB mode.
- RPL WAITX is required if UPAD is required. A UPAD routine can be taken only if RPL specifies WAITX.
- VSAM gives control to the UPAD exit in the same primary address space of the VSAM record management request. However, VSAM can give control to UPAD with home and secondary ASIDs different from those of the VSAM record management request because the exit was set up during OPEN.
- When a UPAD is taken to do post processing, make sure the ECB is marked posted before returning to VSAM. VSAM does not check the UPAD return code and does not do post after UPAD has been taken. For non-cross-memory task mode only, if the UPAD exit taken for wait returns with ECB not posted, VSAM issues a WAIT SVC.
- The UPAD exit must return to VSAM in the same address space, mode, state, and addressing mode, and under the same TCB or SRB from which the UPAD exit was called. Registers 1, 13, and 14 must be restored before the UPAD exit returns to VSAM.
- ICI does not require UPAD for any mode. Resource wait and post processings do not apply to ICI.

RLS ignores the UPAD exit.

Register Contents

Table 20 shows the register contents passed by VSAM when the UPAD exit routine is entered.

Table 20. Contents of Registers at Entry to UPAD Exit Routine

Register	Contents
0	Unpredictable.
1	Address of a parameter list built by VSAM.
2-12	Unpredictable.
13	Reserved.
14	Return address to VSAM.
15	Entry address of the UPAD routine.

Programming Considerations

The UPAD exit routine must be active before the data set is opened. The exit must not be made inactive during processing. If the UPAD exit is desired and multiple ACBs are used for processing the data set, the first ACB that is opened must specify the exit list that identifies the UPAD exit routine.

The contents of the parameter list built by VSAM, pointed to by register 1, can be examined by the UPAD exit routine (see Table 21).

Table 21. Parameter List Passed to UPAD Routine

Offset	Bytes	Description
0(X'0')	4	Address of user's RPL; address of system-generated RPL if UPAD is taken for CLOSE processing or for an alternate index through a path.
4(X'4')	4	Address of a 5-byte data set identifier. The first four bytes of the identifier are the ACB address. The last byte identifies the component; data (X'01'), or index (X'02').
8(X'8')	4	Address of the request's ECB.
12(X'0C')	4	Reserved.
12(X'10')	1	UPAD flags: Bit 0 = ON: Wait for resource Bit 0 = OFF: Wait for I/O (Bit 0 is only applicable to UPAD taken for wait processing.) Lower 7 bits are reserved.
16(X'11')	4	Reserved.
20(X'14')	1	Reason code: X'00' VSAM to do wait processing X'04' UPAD to do post processing X'08'–X'FC' Reserved

If the UPAD exit routine modifies register 14 (for example, by issuing a TESTCB), the routine must restore register 14 before returning to VSAM. If register 1 is used, the UPAD exit routine must restore it with the parameter list address before returning to VSAM.

The UPAD routine must return to VSAM under the same TCB from which it was called for completion of the request that caused VSAM to exit. The UPAD exit routine cannot use register 13 as a save area pointer without first obtaining its own save area.

The UPAD exit routine, when taken before a WAIT during LSR or GSR processing, might issue other VSAM requests to obtain better processing overlap (similar to asynchronous processing). However, the UPAD routine must not issue any synchronous VSAM requests that do not specify WAITX, because a started request might issue a WAIT for a resource owned by a starting request.

If the UPAD routine starts requests that specify WAITX, the UPAD routine must be reentrant. After multiple requests have been started, they should be synchronized by waiting for one ECB out of a group of ECBs to be posted complete rather than waiting for a specific ECB or for many ECBs to be posted complete. (Posting of some ECBs in the list might be dependent on the resumption of some of the other requests that entered the UPAD routine.)

If you are executing in cross-memory mode, you must have a UPAD routine and RPL must specify WAITX. When waiting or posting of an event is required, the UPAD routine is given control to do wait or post processing (reason code 0 or 4 in the UPAD parameter list).

User-Security-Verification Routine

If you use VSAM password protection, you can also have your own routine to check a requester's authority. Your routine is invoked from OPEN, rather than via an exit list. VSAM transfers control to your routine, which must reside in SYS1.LINKLIB, when a requester gives a correct password other than the master password.

Recommendation: Do not use VSAM password protection. Instead, use RACF or an equivalent product.

Through the access method services DEFINE command with the AUTHORIZATION parameter you can identify your user-security-verification routine (USVR) and associate as many as 256 bytes of your own security information with each data set to be protected. The user-security-authorization record (USAR) is made available to the USVR when the routine gets control. You can restrict access to the data set as you choose. For example, you can require that the owner of a data set give ID when defining the data set and then permit only the owner to gain access to the data set.

If the USVR is being used by more than one task at a time, you must code the USVR reentrant or develop another method for handling simultaneous entries.

When your USVR completes processing, it must return (in register 15) to VSAM with a return code of 0 for authority granted or not 0 for authority withheld in register 15. Table 22 gives the contents of the registers when VSAM gives control to the USVR.

Table 22. Communication with User-Security-Verification Routine

Register	Contents
0	Unpredictable.
1	Address of a parameter list with the following format: 44 bytes—Name of the data set for which authority to process is to be verified (the name you specified when you defined it with access method services) 8 bytes—Prompting code (or 0s). 8 bytes—Owner identification (or 0s). 8 bytes—The password that the requester gave (it has been verified by VSAM). 2 bytes - Length of the user-security-authorization routine (in binary). – —The user-security authorization.
2-13	Unpredictable.
14	Return address to VSAM.
15	Entry address to the USVR. When the routine returns to VSAM, it indicates by the following codes in register 15 if the requester has been authorized to gain access to the data set: 0 Authority granted. not 0 Authority withheld.

Chapter 18. Using 31-Bit Addressing Mode with VSAM

This chapter covers rules, guidelines, and keyword parameters for 31-bit addressing.

VSAM lets you obtain control blocks, buffers, and multiple local shared resource (LSR) pools above or below 16 MB. However, if your program uses a 24-bit address, it can generate a program check if you attempt to reference control blocks, buffers, or LSR pools located above 16 MB. With a 24-bit address, you do not have addressability to the data buffers.

If you specify that control blocks, buffers, or pools can be above the line and attempt to use LOCATE mode to access records while in 24-bit mode, your program will program check (ABEND 0C4).

Rule: You cannot specify the location of buffers or control blocks for RLS processing. RLS ignores the ACB RMODE31= keyword.

When you use 31-bit addresses, observe the following rules:

- All VSAM control blocks that contain addresses must contain valid 31-bit addresses. If you are using 24-bit or 31-bit addresses, do not use the high-order byte of a 31-bit address field as a user-defined flag field.
- I/O buffers and control blocks can be obtained either above or below 16 MB in storage.
  - I/O buffers and control blocks can be requested below 16 MB by taking the ACB, GENCB, MODCB, or BLDVRP macro defaults.
  - I/O buffers can be requested above 16 MB and control blocks below 16 MB by specifying the RMODE31=BUFF parameter on the ACB, GENCB, MODCB, or BLDVRP macros.
  - Control blocks can be requested above 16 MB and buffers below 16 MB by specifying the RMODE31=CB parameter on the ACB, GENCB, MODCB, or BLDVRP macros.
  - Control blocks and buffers can be requested above 16 MB by specifying the RMODE31=ALL parameter on the ACB, GENCB, MODCB, or BLDVRP macros.
  - Buffers are obtained in 24-bit addressable storage by specifying the RMODE31=NONE or CB subparameter on the AMP parameter.
  - Control blocks are obtained in 24-bit addressable storage by specifying the RMODE31=NONE or BUFF subparameter on the AMP parameter.
- The parameter list passed to your UPAD and JRNAD exit routine resides in the same area specified with the VSAM control blocks. If RMODE31=CB or RMODE31=ALL is specified, the parameter list resides above 16 MB.
- You must recompile the portion of your program that contains the ACB, BLDVRP, and DLVRP macro specifications.
- You cannot use JCL parameters to specify 31-bit parameters.

Table 23 on page 264 summarizes the 31-bit address keyword parameters and their use in the applicable VSAM macros.

© Copyright IBM Corp. 1987, 2002

Using 31-Bit Addressing Mode with VSAM

Table 23. 31-Bit Address Keyword Parameters

MACRO	RMODE31=	MODE=	LOC=
ACB	Virtual storage location of VSAM control blocks and I/O buffers	INVALID	INVALID
BLDVRP	Virtual storage location of VSAM LSR pool, VSAM control blocks and I/O buffers	Format of the BLDVRP parameter list (24-bit or 31-bit format)	INVALID
CLOSE	INVALID	Format of the CLOSE parameter list (24-bit or 31-bit format)	INVALID
DLVRP	INVALID	Format of the DLVRP parameter list (24-bit or 31-bit format)	INVALID
GENCB	RMODE31 values to be placed in the ACB that is being created. When the generated ACB is opened, the RMODE31 values will then determine the virtual storage location of VSAM control blocks and I/O buffers.	INVALID	Location for the virtual storage obtained by VSAM for the ACB, RPL, or EXIT LIST.
MODCB	RMODE31 values to be placed in a specified ACB	INVALID	INVALID
OPEN	INVALID	Format of the OPEN parameter list (24-bit or 31-bit format)	INVALID

See “Obtaining Buffers Above 16 MB” on page 170 for information about creating and accessing buffers that reside above 16 MB. See “Chapter 14. Sharing Resources Among VSAM Data Sets” on page 211 for information about building multiple LSR pools in an address space.

Chapter 19. Using Job Control Language for VSAM

This chapter covers the following topics.

Topic	Page
Using JCL Statements and Keywords	265
Creating VSAM Data Sets with JCL	266
Retrieving an Existing VSAM Data Set	272

Using JCL Statements and Keywords

All VSAM data sets, except for variable-length RRDSs, can be defined, created, and retrieved using Job Control Language (JCL). When a JCL DD statement is used to identify a VSAM data set, the DD statement must contain data set name and disposition (DISP) keywords. You can add other keywords to the DD statement when appropriate. “Creating VSAM Data Sets with JCL” on page 266 and “Retrieving an Existing VSAM Data Set” on page 272 describe the required JCL keywords. See *z/OS MVS JCL Reference* for descriptions of all the DD statements.

Data Set Name

The data set name (DSNAME) parameter specifies the name of the data set being processed. For a new data set, the specified name is assigned to the data set. For an existing data set, the system uses the name to locate the data set.

Optionally, the DSNAME parameter can be used to specify one of the components of a VSAM data set. Each VSAM data set is defined as a cluster of one or more components. Key-sequenced data sets contain a data component and an index component. Entry-sequenced and linear data sets and fixed-length RRDSs contain only a data component. Process a variable-length RRDS as a cluster. Each alternate index contains a data component and an index component. For further information on specifying a cluster name see “Naming a Cluster” on page 106.

Disposition

The disposition (DISP) parameter describes the status of a data set to the system and tells the system what to do with the data set after the step or job terminates.

All new system-managed and VSAM data sets are treated as if DISP=(NEW,CATLG) were specified. They are cataloged at step initiation time.

To protect non-system-managed data sets in a shared environment, specify DISP=OLD for any data set that can be accessed improperly in a shared environment. Specifying DISP=OLD permits only one job step to access the data set. If the data set’s share options permit the type of sharing your program anticipates, you can specify DISP=SHR in the DD statements of separate jobs to enable two or more job steps to share a data set. With separate DD statements, several subtasks can share a data set under the same rules as for cross-region sharing. When separate DD statements are used and one or more subtasks will perform output processing, the DD statements must specify DISP=SHR. For more details on sharing data sets see “Chapter 13. Sharing VSAM Data Sets” on page 195.

© Copyright IBM Corp. 1987, 2002

Creating VSAM Data Sets with JCL

You can use the JCL DD statement with the RECORG parameter to create a permanent or temporary VSAM data set. SMS must be active, but the data set does not have to be system managed. The system catalogs a permanent VSAM data set when the data set is allocated.

With SMS, you can optionally specify a data class that contains RECORG. If your storage administrator, through the ACS routines, creates a default data class that contains RECORG, you have the option of taking this default as well.

The following list contains the keywords, including RECORG, used to allocate a VSAM data set. See *z/OS MVS JCL Reference* for a detailed description of these keywords.

AVGREC—Specifies the scale value of an average record request on the SPACE keyword. The system applies the scale value to the primary and secondary quantities specified in the SPACE keyword. The AVGREC keyword is ignored if the SPACE keyword specifies anything but an average record request.

Possible values for the AVGREC keyword follow:

U—Use a scale of 1
K—Use a scale of 1024
M—Use a scale of 1 048 576

DATACLAS—Is a list of the data set allocation parameters and their default values. The storage administrator can specify KEYLEN, KEYOFF, LRECL, LGSTREAM, and RECORG in the DATACLAS definition, but you can override them.

EXPDT—Specifies the date up to which a data set cannot be deleted without specifying the PURGE keyword on the access method services DELETE command. On and after the expiration date, the data set can be deleted or written over by another data set.

KEYLEN—Specifies key length.

LGSTREAM—Specifies the log stream used. LOG and BWO parameters can be derived from the data class.

KEYOFF—Specifies offset to key.

LIKE—Specifies that the properties of an existing cataloged data set should be used to allocate a new data set. For a list of the properties that can be copied, see *z/OS MVS JCL Reference*.

LRECL—Specifies logical record length. Implies a system determined control interval size.

MGMTCLAS—Specifies the name, 1 to 8 characters, of the management class for a new system-managed data set. Your storage administrator defines the names of the management classes you can specify on the MGMTCLAS parameter. After the data set is allocated, attributes in the management class control the following:

- The migration of the data set, which includes migration criteria from primary storage to migration storage and from one migration level to another in a hierarchical migration scheme.
- The backup of the data set, which includes frequency of backup, number of versions, and retention criteria for backup versions.

RECORG—Specifies the type of data set desired: KS, ES, RR, LS.

KS = key-sequenced data set
ES = entry-sequenced data set
RR = fixed-length relative-record data set
LS = linear data set

REFDD—Specifies that the properties on the JCL statement and from the data class of a previous DD statement should be used to allocate a new data set.

RETPD—Specifies the number of days a data set cannot be deleted by specifying the PURGE keyword on the access method services DELETE command. After the retention period, the data set can be deleted or written over by another data set.

SECMODEL—Permits specification of the name of a “model” profile that RACF should use in creating a discrete profile for the data set. For a list of the information that is copied from the model profile see *z/OS MVS JCL Reference*.

STORCLAS—Specifies the name, 1 to 8 characters, of the storage class for a new, system-managed data set.

Your storage administrator defines the names of the storage classes you can specify on the STORCLAS parameter. A storage class is assigned when you specify STORCLAS or an ACS routine selects a storage class for the new data set.

Use the storage class to specify the storage service level to be used by SMS for storage of the data set. The storage class replaces the storage attributes specified on the UNIT and VOLUME parameter for non-system-managed data sets.

If a guaranteed space storage class is assigned to the data set (cluster) and volume serial numbers are specified, space is allocated on all specified volumes if the following conditions are met:

- All volumes specified belong to the same storage group.
- The storage group to which these volumes belong is in the list of storage groups selected by the ACS routines for this allocation.

Temporary VSAM Data Sets

A temporary data set is allocated and deleted within a job. Also, temporary VSAM data sets must reside in storage managed by the Storage Management Subsystem. SMS manages a data set if you specify a storage class (using the DD STORCLAS parameter) or if an installation-written automatic class selection (ACS) routine selects a storage class for the data set.

Data Set Names

When defining a temporary data set, you can omit the DSNAME. If omitted, the system will generate a qualified name for the data set. If you specify a DSNAME, it must begin with & or &&. The DSNAME can be simple or qualified.

```
&ABC
&XYZ
```

Allocation

You can allocate VSAM temporary data sets by specifying RECORG=KS|ES|LS|RR as follows:

- By the RECORG keyword on the DD statement or the dynamic allocation parameter
- By the data class (as long as the selected data class has the RECORG attribute)
- By the default data class established by your storage administrator (as long as one exists and it has the RECORG attribute)

Using temporary data sets avoids the following problems:

- The data set can be defined in a catalog for which you are not authorized.
- The data set can be assigned to an RACF user or group ID for which you are not authorized. The data set can also be assigned to an ID that is not defined to RACF, in which case the allocation will fail either if the RACF option that required protection of all data sets is in effect, or if the data set requires a discrete RACF profile.
- The allocation can fail because there is already a data set cataloged with that name. This failure would be likely if there are many jobs using the same data set names for what were regarded as temporary data sets.

Restrictions for Temporary VSAM Data Sets

The following restrictions apply to the use of temporary VSAM data sets:

- Multivolume temporary VSAM data sets is not permitted.
- You cannot reference a temporary data set once a job is completed.
- The use of VOL=SER and UNIT will not let you refer to a temporary data set outside the job that created it.
- The EXPDT and RETPD keywords are ignored.

For additional information on temporary data sets see *z/OS MVS JCL Reference* and *z/OS MVS Programming: Assembler Services Guide*. See “Example 4: Allocate a Temporary VSAM Data Set” on page 271 for an example of creating a temporary VSAM data set.

Examples Using JCL to Allocate VSAM Data Sets

The following examples contain allocation information.

Example 1: Allocate a Key-Sequenced Data Set

The following example shows allocating a key-sequenced data set:

```
//DDNAME   DD   DSNAME=KSDATA,DISP=(NEW,KEEP),
//              SPACE=(80,(20,2)),AVGREC=U,RECORG=KS,
//              KEYLEN=15,KEYOFF=0,LRECL=250
```

Explanation of Keywords:

- DSNAME specifies the data set name.
- DISP specifies that a new data set is to be allocated in this step and that the data set is to be kept on the volume if this step terminates normally. If the data set is not system managed, KEEP is the only normal termination disposition subparameter permitted for a VSAM data set. Non-system-managed VSAM data sets should not be passed, cataloged, uncataloged, or deleted.
- SPACE specifies an average record length of 80, a primary space quantity of 20 and a secondary space quantity of 2.
- AVGREC specifies that the primary and secondary space quantity specified on the SPACE keyword represents the number of records in units (multiplier of 1). If DATACLAS were specified in this example, AVGREC would override the data class space allocation.
- RECORG specifies a VSAM key-sequenced data set.
- KEYLEN specifies that the length of the keys used in the data set is 15 bytes. If DATACLAS were specified in this example, KEYLEN would override the data class key length allocation.
- KEYOFF specifies an offset of zero of the first byte of the key in each record. If DATACLAS were specified in this example, KEYOFF would override the data class key offset allocation.
- LRECL specifies a record length of 250 bytes. If DATACLAS were specified in this example, LRECL would override the data class record length allocation.
- The system determines an appropriate size for the control interval.

Example 2: Allocate a System-Managed Key-Sequenced Data Set Using Keywords

The following example shows allocating a system-managed key-sequenced data set:

```
//DDNAME   DD   DSNAME=KSDATA,DISP=(NEW,KEEP),
//              DATACLAS=STANDARD,STORCLAS=FAST,
//              MGMTCLAS=STANDARD
```

Explanation of Keywords:

- DSNAME specifies the data set name.
- DISP specifies that a new data set is to be allocated in this step and that the data set is to be kept on the volume if this step terminates normally. Because a system-managed data set is being allocated, all dispositions are valid for VSAM data sets; however, UNCATLG is ignored.
- DATACLAS specifies a data class for the new data set. If SMS is not active, the system syntax checks and then ignores DATACLAS. SMS also ignores the DATACLAS keyword if you specify it for an existing data set, or a data set that SMS does not support.

  This keyword is optional. If you do not specify DATACLAS for the new data set and your storage administrator has provided an ACS routine, the ACS routine can select a data class for the data set.
- STORCLAS specifies a storage class for the new data set. If SMS is not active, the system syntax checks and then ignores STORCLAS. SMS also ignores the STORCLAS keyword if you specify it for an existing data set.

  This keyword is optional. If you do not specify STORCLAS for the new data set and your storage administrator has provided an ACS routine, the ACS routine can select a storage class for the data set.
- MGMTCLAS specifies a management class for the new data set. If SMS is not active, the system syntax checks and then ignores MGMTCLAS. SMS also ignores the MGMTCLAS keyword if you specify it for an existing data set.

  This keyword is optional. If you do not specify MGMTCLAS for the new data set and your storage administrator has provided an ACS routine, the ACS routine can select a management class for the data set.

Example 3: Allocate a VSAM Data Set Using Keyword Defaults

The following example shows the minimum number of keywords needed to allocate a permanent VSAM data set through JCL:

```
//DDNAME   DD   DSNAME=DSVSAM,DISP=(NEW,CATLG)
```

Explanation of Keywords:

- DSNAME specifies the data set name.
- DISP specifies that a new data set is to be allocated in this step and that the system is to place an entry pointing to the data set in the system or user catalog.
- DATACLAS, STORCLAS, and MGMTCLAS are not required if your storage administrator has provided ACS routines that will select the SMS classes for you, and DATACLAS defines RECORG.

Example 4: Allocate a Temporary VSAM Data Set

The following example shows allocating a temporary VSAM data set:

```
//VSAM1    DD   DSN=&CLUSTER,DISP=(NEW,PASS),
//              RECORG=ES,SPACE=(1,(10)),AVGREC=M,
//              LRECL=256,STORCLAS=TEMP
```

Explanation of Keywords:

- DSN specifies the data set name. If you specify a data set name for a temporary data set, it must begin with & or &&. This keyword is optional, however. If you do not specify a DSN, the system will generate a qualified data set name for the temporary data set.
- DISP specifies that a new data set is to be allocated in this step and that the data set is to be passed for use by a subsequent step in the same job. If KEEP or CATLG are specified for a temporary data set, the system changes the disposition to PASS and deletes the data set at job termination.
- RECORG specifies a VSAM entry-sequenced data set.
- SPACE specifies an average record length of 1 and a primary quantity of 10.
- AVGREC specifies that the primary quantity (10) specified on the SPACE keyword represents the number of records in megabytes (multiplier of 1048576).
- LRECL specifies a record length of 256 bytes.
- STORCLAS specifies a storage class for the temporary data set.

  This keyword is optional. If you do not specify STORCLAS for the new data set and your storage administrator has provided an ACS routine, the ACS routine can select a storage class.

Example 5: Allocate a Temporary VSAM Data Set Taking All Defaults

The following example shows the minimum number of keywords required to allocate a temporary VSAM data set:

```
//VSAM2    DD   DISP=(NEW,PASS)
```

If no DSNAME is specified, the system will generate one. If no STORCLAS name is specified, and your storage administrator has provided an ACS routine, the ACS routine can select a storage class. The key length for a key-sequenced data set must be defined in your default DATACLAS. If you do not, this example will fail.

Retrieving an Existing VSAM Data Set

To retrieve an existing VSAM data set, code a DD statement in the form:

```
//ddname  DD   DSNAME=dsname,DISP=OLD
or
//ddname  DD   DSNAME=dsname,DISP=SHR
```

If SMS is active, you can pass VSAM data sets within a job. The system replaces PASS with KEEP for permanent VSAM data sets. When you refer to the data set later in the job, the system obtains data set information from the catalog. Without SMS you cannot pass VSAM data sets within a job.

Migration Consideration

If you have existing JCL that allocates a VSAM data set with DISP=(OLD,DELETE), the system ignores DELETE and keeps the data set if SMS is inactive. If SMS is active, DELETE is valid and the system deletes the data set.

Keywords Used to Process VSAM Data Sets

Use the following keywords to process VSAM data sets once they have been retrieved. Use these parameters to process existing VSAM data sets, not to allocate new VSAM data sets. For information about the JCL parameters used to allocate a new VSAM data set see “Creating VSAM Data Sets with JCL” on page 266.

AMP is only used with VSAM data sets. The AMP parameter takes effect when the data set defined by the DD statement is opened.

Note: This is not supported by RLS.

DDNAME lets you postpone defining a data set until later in the job step.

DISP =(**SHR**|**OLD**[,**PASS**]) describes the status of a data set, and tells the system what to do with the data set after the step or job ends.

DSNAME specifies the name of a data set.

DUMMY specifies that no disposition processing is to be performed on the data set. It also specifies that no device or external storage space is to be allocated to the data set.

DYNAM increases by one the control value for dynamically allocated resources held for reuse.

FREE specifies when the system is to deallocate resources for the data set.

PROTECT tells RACF to protect the data set.

UNIT =(*device number*|*type*|*group*,p|*unitcount*) places the data set on a specific device or a group of devices.

VOLUME =(**PRIVATE**|**SER**) identifies the volume on which a data set will reside.

With SMS, you do not need the AMP, UNIT, and VOLUMES parameters to retrieve an existing VSAM data set. With SMS, you can use the DISP subparameters MOD, NEW, CATLG, KEEP, PASS, and DELETE for VSAM data sets.

Certain JCL keywords should either not be used, or used only with caution when processing VSAM data sets. See the VSAM data set section in *z/OS MVS JCL User's Guide* for a list of these keywords. Additional descriptions of these keywords also appear in *z/OS MVS JCL Reference*.

Chapter 20. Processing Indexes of Key-Sequenced Data Sets

This chapter covers the following topics.

Topic	Page
Access to a Key-Sequenced Data Set Index	275
Format of an Index Record	279
Key Compression	282

VSAM lets you access indexes of key-sequenced data sets to help you diagnose index problems. This can be useful if your index is damaged or if pointers are lost and you want to know exactly what the index contains. You should not attempt to duplicate or substitute the index processing done by VSAM during normal access to data records.

Access to a Key-Sequenced Data Set Index

You can gain access to the index of a key-sequenced data set in one of two ways:

- By opening the cluster and using the GETIX and PUTIX macros
- By opening the index component alone and using the macros for normal data processing (GET, PUT, and so forth)

Access to an Index with GETIX and PUTIX

To process the index of a key-sequenced data set with GETIX and PUTIX, you must open the cluster with ACB MACRF=(CNV,...) specified. CNV provides for control interval access, which you use to gain access to the index component.

Access using GETIX and PUTIX is direct, by control interval: VSAM requires RPL OPTCD=(CNV,DIR). The search argument for GETIX is the RBA of a control interval. The increment from the RBA of one control interval to the next is control interval size for the index.

GETIX can be issued either for update or not for update. VSAM recognizes OPTCD=NUP or UPD but interprets OPTCD=NSP as NUP.

The contents of a control interval cannot be inserted through PUTIX. VSAM requires OPTCD=UPD. The contents must previously have been retrieved for update through GETIX.

RPL OPTCD=MVE or LOC can be specified for GETIX, but only OPTCD=MVE is valid for PUTIX. If you retrieve with OPTCD=LOC, you must change OPTCD to MVE to store. With OPTCD=MVE, AREALEN must be at least index control interval size.

Beyond these restrictions, access to an index through GETIX and PUTIX follows the rules found in “Chapter 12. Processing Control Intervals” on page 183.

Access to the Index Component Alone

You can gain addressed or control interval access to the index component of a key-sequenced cluster by opening the index component alone and using the

© Copyright IBM Corp. 1987, 2002

request macros for normal data processing. To open the index component alone, specify: DSNAME=*indexcomponentname* in the DD statement identified in the ACB (or GENCB) macro.

You can gain access to index records with addressed access and to index control intervals with control interval access. The use of these two types of access for processing an index is identical in every respect with their use for processing a data component.

Processing the index component alone is identical to processing an entry-sequenced data set. An index itself has no index and thus cannot be processed by keyed access.

Prime Index

A key-sequenced data set always has an index that relates key values to the relative locations of the logical records in a data set. This index is called the prime index. The prime index, or simply index, has two uses:

- Locate the collating position when inserting records
- Locate records for retrieval

When a data set is initially loaded, records must be presented to VSAM in key sequence. The index for a key-sequenced data set is built automatically by VSAM as the data set is loaded with records. The index is stored in control intervals. An index control interval contains pointers to index control intervals in the next lower level, or one entry for each data control interval in a control area.

When a data control interval is completely loaded with logical records, free space, and control information, VSAM makes an entry in the index. The entry consists of the highest possible key in the data control interval and a pointer to the beginning of that control interval. The highest possible key in a data control interval is one less than the value of the first key in the next sequential data control interval.

Figure 36 shows that a single index entry, such as **19**, contains all the information necessary to locate a logical record in a data control interval.

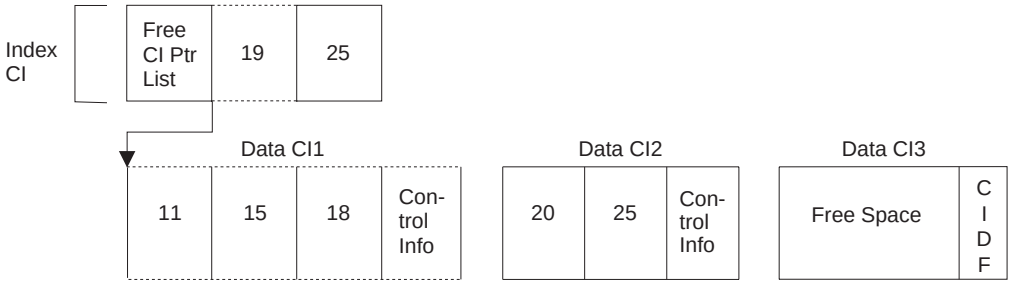


Figure 36. Relation of Index Entry to Data Control Interval

Figure 37 on page 277 shows that a single index control interval contains all the information necessary to locate a record in a single data control area.

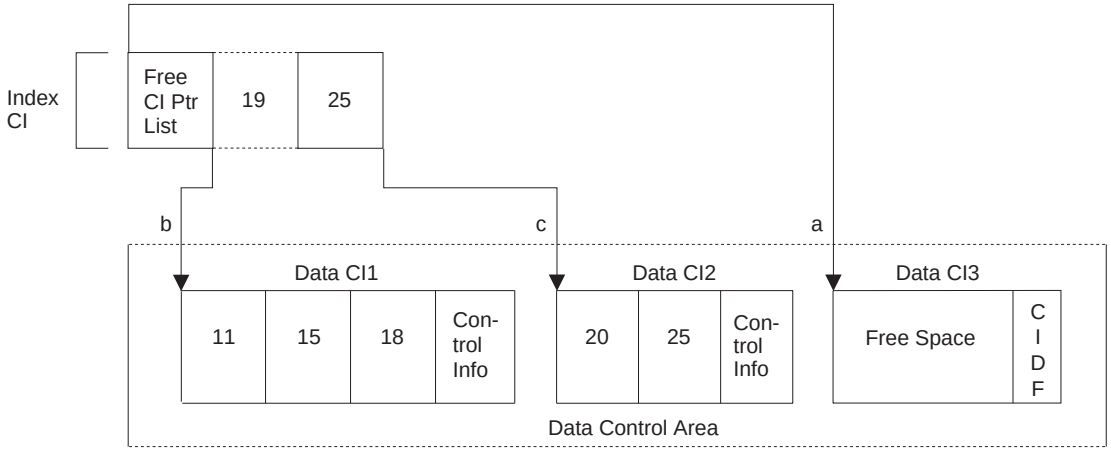


Figure 37. Relation of Index Entry to Data Control Interval

The index contains the following entries:

1. A free control interval pointer list, which indicates available free space control intervals. Because this control area has a control interval that is reserved as free space, VSAM places a free space pointer in the index control interval to locate the free space data control interval.
2. 19, the highest possible key in data control interval 1. This entry points to the beginning of data control interval 1.
3. 25, the highest possible key in data control interval 2. This entry points to the beginning of data control interval 2.

Index Levels

A VSAM index can consist of more than one index level. Each level contains a set of records with entries giving the location of the records in the next lower level.

Figure 38 on page 278 shows the levels of a prime index and shows the relationship between sequence set index records and control areas. The sequence set shows both the horizontal pointers used for sequential processing and the vertical pointers to the data set. Although the values of the keys are actually compressed in the index, the figure shows the full key values.

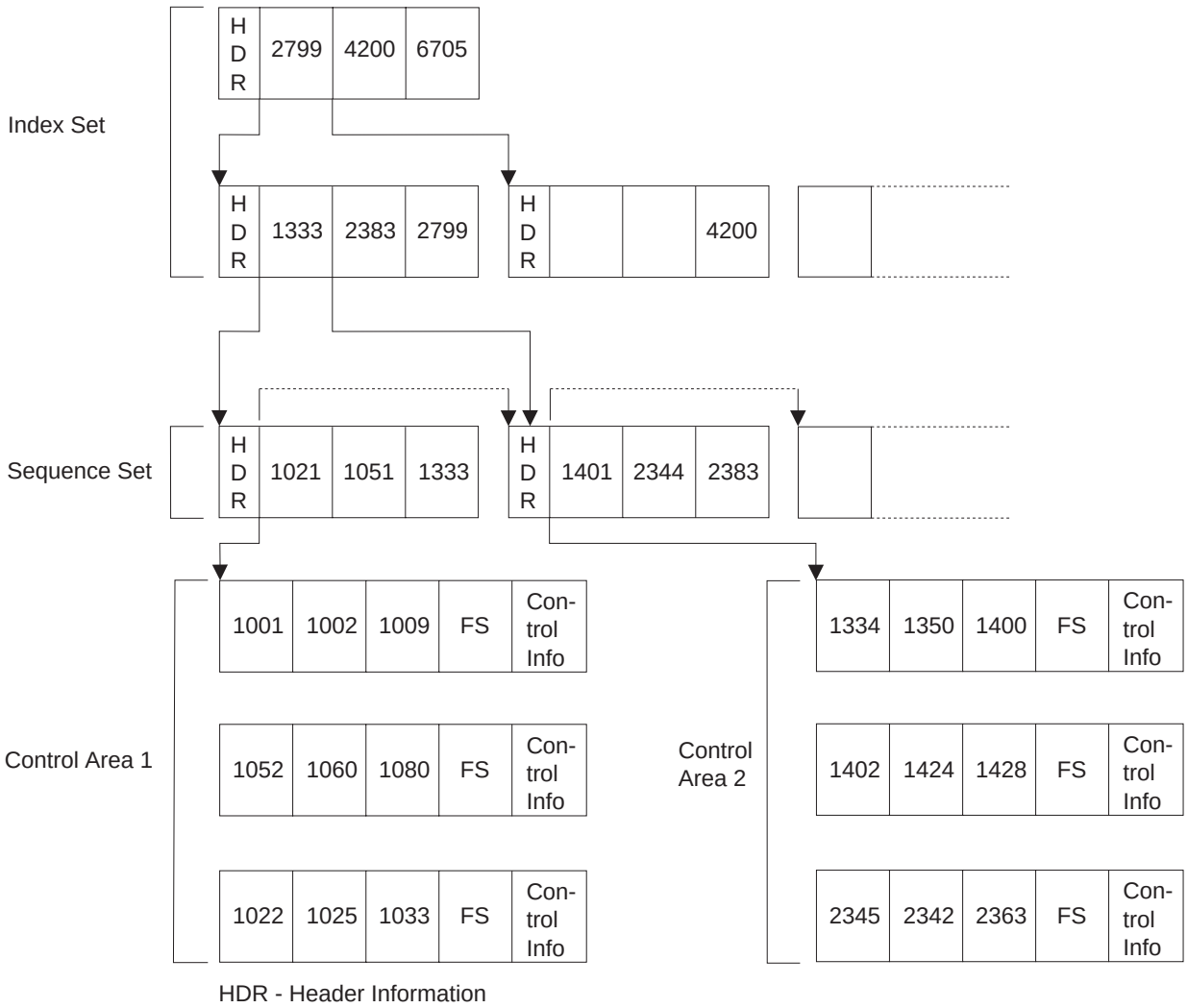


Figure 38. Levels of a Prime Index

Sequence Set. The index records at the lowest level are the sequence set. There is one index sequence set level record for each control area in the data set. This sequence set record gives the location of data control intervals. An entry in a sequence set record consists of the highest possible key in a control interval of the data component, paired with a pointer to that control interval.

Index Set. If there is more than one sequence set level record, VSAM automatically builds another index level. Each entry in the second level index record points to one sequence set record. The records in all levels of the index above the sequence set are called the index set. An entry in an index set record consists of the highest possible key in an index record in the next lower level, and a pointer to the beginning of that index record. The highest level of the index always contains only a single record.

When you access records sequentially, VSAM refers only to the sequence set. It uses a horizontal pointer to get from one sequence set record to the next record in collating sequence. When you access records directly (not sequentially), VSAM follows vertical pointers from the highest level of the index down to the sequence set to find vertical pointers to data.

Format of an Index Record

Index records are stored in control intervals the same as data records, except that only one index record is stored in a control interval, and there is no free space between the record and the control information. So, there is only one RDF that contains the flag X'00' and the length of the record (a number equal to the length of the control interval minus 7). The CIDF also contains the length of the record (the displacement from the beginning of the control interval to the control information); its second number is 0 (no free space). The contents of the RDF and CIDF are the same for every used control interval in an index. The control interval after the last-used control interval has a CIDF filled with 0s, and is used to represent the software end-of-file (SEOF).

Index control intervals are not grouped into control areas as are data control intervals. When a new index record is required, it is stored in a new control interval at the end of the index data set. As a result, the records of one index level are not segregated from the records of another level, except when the sequence set is separate from the index set. The level of each index record is identified by a field in the index header (see “Header Portion”).

When an index record is replicated on a track, each copy of the record is identical to the other copies. Replication has no effect on the contents of records.

Figure 39 shows the parts of an index record.

Figure 39. General Format of an Index Record

An index record contains the following parts:

- A 24-byte header containing control information about the record.
- For a sequence-set index record governing a control area that has free control intervals, there are entries pointing to those free control intervals.
- Unused space, if any.
- A set of index entries used to locate, for an index-set record, control intervals in the next lower level of the index, or, for a sequence-set record, used control intervals in the control area governed by the index record.

Header Portion

The first 24 bytes of an index record is the header, which gives control information about the index record. Table 24 shows its format. All lengths and displacements are in bytes. The discussions in the following two sections amplify the meaning and use of some of the fields in the header.

Table 24. Format of the Header of an Index Record

Field	Offset	Length	Description
IXHLL	0(0)	2	Index record length. The length of the index record is equal to the length of the control interval minus 7.

Table 24. Format of the Header of an Index Record (continued)

Field	Offset	Length	Description
IXHFLPLN	2(2)	1	Index entry control information length. This is the length of the last three of the four fields in an index entry. (The length of the first field is variable.) The length of the control information is 3, 4, or 5 bytes.
IXHPTLS	3(3)	1	Vertical-pointer-length indicator. The fourth field in an index entry is a vertical pointer to a control interval. In an index-set record, the pointer is a binary number that designates a control interval in the index. The number is calculated by dividing the RBA of the control interval by the length of the control interval. To permit for a possibly large index, the pointer is always 3 bytes. In a sequence-set record, the pointer is a binary number, beginning at 0, and calculated the same as for index-set record, that designates a control interval in the data control area governed by the sequence-set record. A free-control-interval entry is nothing more than a vertical pointer. There are as many index entries and free-control-interval entries in a sequence-set record as there are control intervals in a control area. Depending on the number of control intervals in a control area, the pointer is 1, 2, or 3 bytes. An IXHPTLS value of X'01' indicates a 1-byte pointer; X'03' indicates a 2-byte pointer; X'07' indicates a 3-byte pointer.
IXHBRBA	4(4)	4	Base RBA. In an index-set record, this is the beginning RBA of the index. Its value is 0. The RBA of a control interval in the index is calculated by multiplying index control interval length times the vertical pointer and adding the result to the base RBA. In a sequence-set record, this is the RBA of the control area governed by the record. The RBA of a control interval in the control area is calculated by multiplying data control interval length times the vertical pointer and adding the result to the base RBA. Thus, the first control interval in a control area has the same RBA as the control area (length times 0, plus base RBA, equals base RBA).
IXHHP	8(8)	4	Horizontal-pointer RBA. This is the RBA of the next index record in the same level as this record. The next index record contains keys next in ascending sequence after the keys in this record.
	12(C)	4	Reserved.
IXHLV	16(10)	1	Level number. The sequence set is the first level of an index, and each of its records has an IXHLV of 1. Records in the next higher level have a 2, and so on.
	17(11)	1	Reserved.
IXHFSO	18(12)	2	Displacement to the unused space in the record. In an index-set record, this is the length of the header (24). There are no free control interval entries. In a sequence-set record, the displacement is equal to 24, plus the length of free control interval entries, if any.
IXHLEO	20(14)	2	Displacement to the control information in the last index entry. The last (leftmost) index entry contains the highest key in the index record. In a search, if the search-argument key is greater than the highest key in the preceding index record but less than or equal to the highest key in this index record, then this index record governs either the index records in the next lower level that have the range of the search-argument key or the control area in which a data record having the search-argument key is stored.

Table 24. Format of the Header of an Index Record (continued)

Field	Offset	Length	Description
IXHSEO	22(16)	2	Displacement to the control information in the last (leftmost) index entry in the first (rightmost) section. Index entries are divided into sections to simplify a quick search. Individual entries are not examined until the right section is located.

Free Control Interval Entry Portion

If the control area governed by a sequence-set record has free control intervals, the sequence-set record has entries pointing to those free control intervals. Each entry is 1, 2, or 3 bytes long (indicated by IXHPTLS in the header: the same length as the pointers in the index entries).

The entries come immediately after the header. They are used from right to left. The rightmost entry is immediately before the unused space (whose displacement is given in IXHFSO in the header). When a free control interval gets used, its free entry is converted to zero, the space becomes part of the unused space, and a new index entry is created in the position determined by ascending key sequence.

Thus, the free control interval entry portion contracts to the left, and the index entry portion expands to the left. When all the free control intervals in a control area have been used, the sequence-set record governing the control area no longer has free control interval entries, and the number of index entries equals the number of control intervals in the control area. Note that if the index control interval size was specified with too small a value, it is possible for the unused space to be used up for index entries before all the free control intervals have been used, resulting in control intervals within a data control area that cannot be used.

Index Entry Portion

The index entry portion of an index record takes up all of the record that is left over after the header, the free control interval entries, if any, and the unused space.

Figure 40 shows the format of the index entry portion of an index record. To improve search speed, index entries are grouped into sections, of which there are approximately as many as the square root of the number of entries. For example, if there are 100 index entries in an index record, they are grouped into 10 sections of 10 entries each. (The number of sections does not change, even though the number of index entries increases as free control intervals get used.)

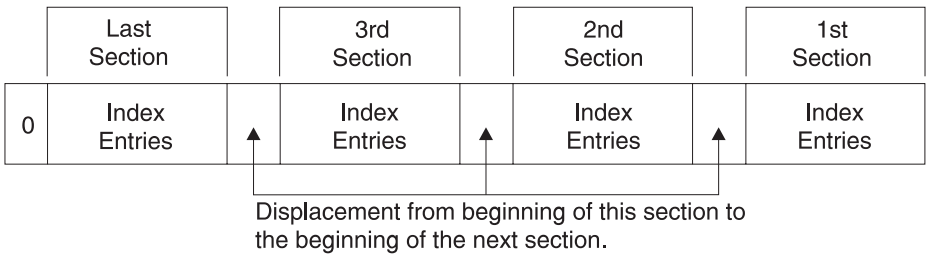


Figure 40. Format of the Index Entry Portion of an Index Record

The sections, and the entries within a section, are arranged from right to left. IXHLEO in the header gives the displacement from the beginning of the index record to the control information in the leftmost index entry. IXHSEO gives the

displacement to the control information in the leftmost index entry in the rightmost section. You calculate the displacement of the control information of the rightmost index entry in the index record (the entry with the lowest key) by subtracting IXHFLPLN from IXHLL in the header (the length of the control information in an index entry from the length of the record).

Each section is preceded by a 2-byte field that gives the displacement from the control information in the leftmost index entry in the section to the control information in the leftmost index entry in the next section (to the left). The last (leftmost) section's 2-byte field contains 0s.

Figure 41 gives the format of an index entry.

	Control Information		
Compressed key	F	L	P

F-Number of characters eliminated from the front
L-Number of characters left in key after compression
P-Vertical pointer

Figure 41. Format of an Index Record

Key Compression

Index entries are variable in length within an index record because VSAM compresses keys. That is, it eliminates redundant or unnecessary characters from the front and back of a key to save space. The number of characters that can be eliminated from a key depends on the relationship between that key and the preceding and following keys.

For front compression, VSAM compares a key in the index with the preceding key in the index and eliminates from the key those leading characters that are the same as the leading characters in the preceding key. For example, if key 12356 follows key 12345, the characters 123 are eliminated from 12356 because they are equal to the first three characters in the preceding key. The lowest key in an index record has no front compression; there is no preceding key in the index record.

There is an exception for the highest key in a section. For front compression, it is compared with the highest key in the preceding section, rather than with the preceding key. The highest key in the rightmost section of an index record has no front compression; there is no preceding section in the index record.

What is called "rear compression" of keys is actually the process of eliminating the insignificant values from the end of a key in the index. The values eliminated can be represented by X'FF'. VSAM compares a key in the index with the following key in the data and eliminates from the key those characters to the right of the first character that are unequal to the corresponding character in the following key. For example, if the key 12345 (in the index) precedes key 12356 (in the data), the character 5 is eliminated from 12345 because the fourth character in the two keys is the first unequal pair.

The first of the control information fields gives the number of characters eliminated from the front of the key, and the second field gives the number of characters that remain. When the sum of these two numbers is subtracted from the full key length (available from the catalog when the index is opened), the result is the number of characters eliminated from the rear. The third field indicates the control interval that contains a record with the key.

The example in Figure 42 on page 284 gives a list of full keys and shows the contents of the index entries corresponding to the keys that get into the index (the highest key in each data control interval). A sequence-set record is assumed, with vertical pointers 1 byte long. The index entries shown in the figure from top to bottom are arranged from right to left in the assumed index record.

Key 12345 has no front compression because it is the first key in the index record. Key 12356 has no rear compression because, in the comparison between 12356 and 12357, there are no characters following 6, which is the first character that is unequal to the corresponding character in the following key.

You can always figure out what characters have been eliminated from the front of a key. You cannot figure out the ones eliminated from the rear. Rear compression, in effect, establishes the key in the entry as a boundary value instead of an exact high key. That is, an entry does not give the exact value of the highest key in a control interval, but gives only enough of the key to distinguish it from the lowest key in the next control interval. For example, in Figure 42 on page 284 the last three index keys are 12401, 124, and 134 after rear compression. Data records with key field between:

- 12402 and 124FF are associated with index key 124.
- 12500 and 134FF are associated with index key 134.

If the last data record in a control interval is deleted, and if the control interval does not contain the high key for the control area, then the space is reclaimed as free space. Space reclamation can be suppressed by setting the RPLNOCIR bit, which has an equated value of X'20', at offset 43 into the RPL.

The last index entry in an index level indicates the highest possible key value. The convention for expressing this value is to give none of its characters and indicate that no characters have been eliminated from the front. The last index entry in the last record in the sequence set looks like this:

F	L	P
0	0	X

Where x is a binary number from 0 to 255, assuming a 1-byte pointer.

In a search, the two 0s signify the highest possible key value in this way:

- The fact that 0 characters have been eliminated from the front implies that the first character in the key is greater than the first character in the preceding key.
- A length of 0 indicates that no character comparison is required to determine if the search is successful. That is, when a search finds the last index entry, a hit has been made.

Full Key of Data Record	Index Entry	Eliminated From Front	Eliminated From Rear
12345	K: 1 2 3 4; F: 0; L: 4; P: 0	None	5
12350			
12353			
12354			
12356	K: 5 6; F: 3; L: 2; P: 1	123	none
12357			
12358			
12359	F: 4; L: 0; P: 2	1235	9
12370			
12373			
12380			
12385			
12390			
12401	K: 4 0 1; F: 2; L: 3; P: 3	12	none
12405			
12410			
12417			
12421	F: 3; L: 0; P: 4	124	21
12600			
13200			
13456	K: 3 4; F: 1; L: 2; P: 5	1	56

Note: 'Full Keys' are the full keys of the data records that reside in data CIs where highest possible keys are compressed in the corresponding index entries.

Legend:
K-Characters left in key after compression
F-Number of characters eliminated from the front
L-Number of characters left in key after compression
P-Vertical pointer

DA6D4032

Figure 42. Example of Key Compression

Index Update Following a Control Interval Split

When a data set is first loaded, the key sequence of data records and their physical order are the same. However, when data records are inserted, control interval splits can occur, causing the data control intervals to have a physical order that differs from the key sequence.

Figure 43 shows how the control interval is split and the index is updated when a record with a key of 12 is inserted in the control area shown in Figure 37 on page 277.

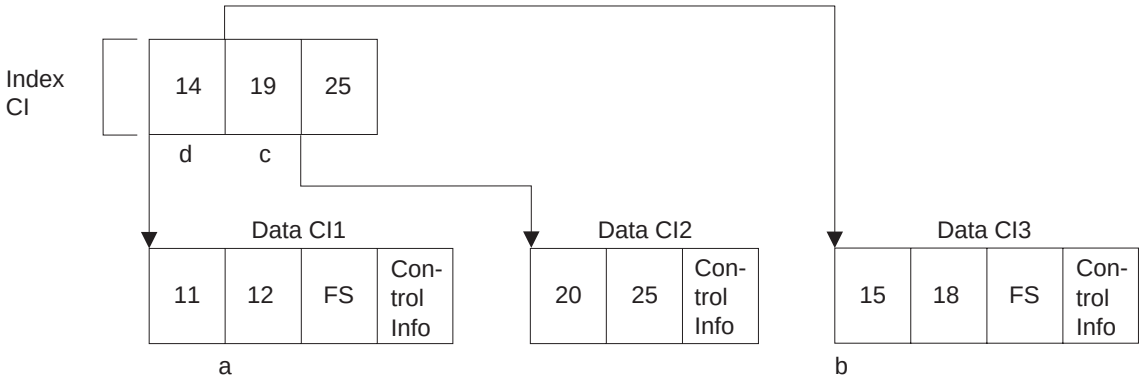


Figure 43. Control Interval Split and Index Update

1. A control interval split occurs in data control interval 1, where a record with the key of 12 must be inserted.
2. Half the records in data control interval 1 are moved by VSAM to the free space control interval (data control interval 3).
3. An index entry is inserted in key sequence to point to data control interval 3, that now contains data records moved from data control interval 1.
4. A new index entry is created for data control interval 1, because after the control interval split, the highest possible key is 14. Because data control interval 3 now contains data, the pointer to this control interval is removed from the free list and associated with the new key entry in the index. Note that key values in the index are in proper ascending sequence, but the data control intervals are no longer in physical sequence.

Index Entries for a Spanned Record

In a key-sequenced data set, there is an index entry for each control interval that contains a segment of a spanned record. All the index entries for a spanned record are grouped together in the same section. They are ordered from right to left according to the sequence of segments (first, second, third, and so on).

Only the last (leftmost) index entry for a spanned record contains the key of the record. The key is compressed according to the rules described above. All the other index entries for the record look like this:

F	L	P
Y	0	X

Where y is a binary number equal to the length of the key
(y indicates that the entire key has been eliminated from the front).
L indicates that 0 characters remain.
X identifies the control interval that contains the segment.

DA6D4034

Part 3. Non-VSAM Data Sets

© Copyright IBM Corp. 1987, 2002

Chapter 21. Selecting Record Formats for Non-VSAM Data Sets

This chapter covers the following topics.

Topic	Page
Format Selection	289
Fixed-Length Record Formats	290
Variable-Length Record Formats	292
Undefined-Length Record Format	298
ISO/ANSI Tapes	299
Record Format—Device Type Considerations	307

This chapter discusses record formats of non-VSAM data sets and device type considerations. Records are stored in one of four formats:

- Fixed length (RECFM=F)
- Variable length (RECFM=V)
- ASCII variable length (RECFM=D)
- Undefined length (RECFM=U)

For information about disk format, see “Direct Access Storage Device (DASD) Volumes” on page 8.

Format Selection

Before selecting a record format, you should consider:

- The data type (for example, EBCDIC) your program can receive and the type of output it can produce
- The I/O devices that contain the data set
- The access method you use to read and write the records
- Whether the records can be blocked

Blocking is the process of grouping records into blocks before they are written on a volume. A block consists of one or more logical records. Each block is written between consecutive interblock gaps. Blocking conserves storage space on a volume by reducing the number of interblock gaps in the data set, and increases processing efficiency by reducing the number of I/O operations required to process the data set.

If you do not specify a block size, the system generally determines a block size that is optimum for the device to which your data set is allocated. See “System-Determined Block Size” on page 324.

You select your record format in the data control block (DCB) using the options in the DCB macro, the DD statement, dynamic allocation, automatic class selection routines or the data set label. Before executing your program, you must supply the operating system with the record format (RECFM) and device-dependent information in data class, a DCB macro, a DD statement, or a data set label. A complete description of the DD statement keywords and a glossary of DCB subparameters is contained in *z/OS MVS JCL Reference*.

© Copyright IBM Corp. 1987, 2002

Selecting Record Formats for Non-VSAM Data Sets

All record formats except U can be blocked. Variable-length records can be spanned (RECFM=DS or VS). Spanned records can span more than one block. Fixed-length records (RECFM=F or FB) can be specified as standard (RECFM=FS or FBS). Standard format means there are no short blocks or unfilled tracks within the data set, except for the last block or track.

Fixed-Length Record Formats

The size of fixed-length (format-F or -FB) records, shown in Figure 44, is constant for all records in the data set.

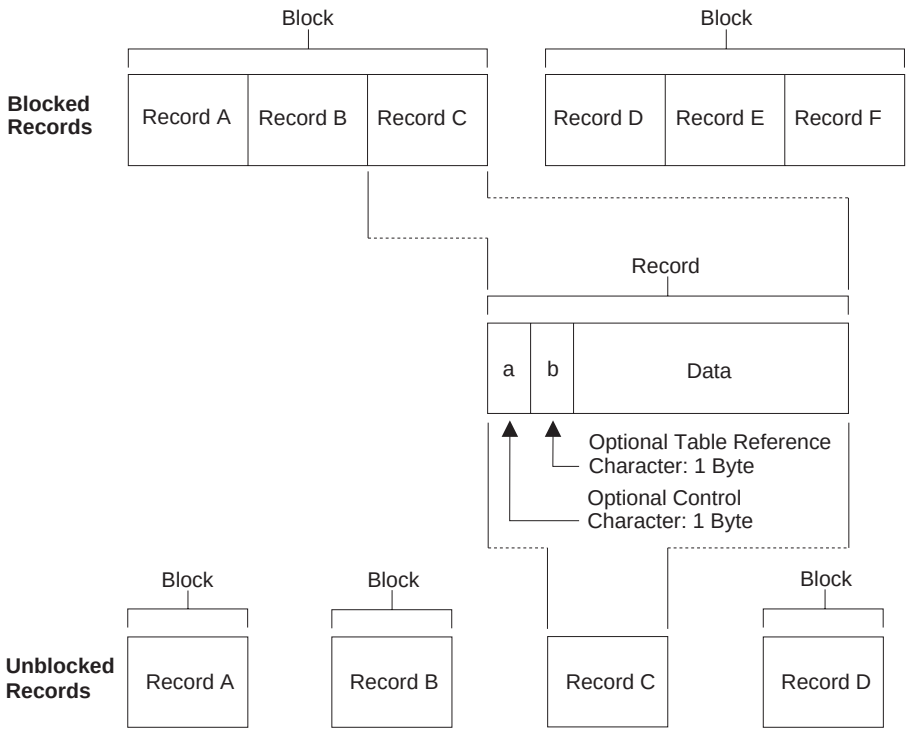


Figure 44. Fixed-Length Records

The records can be blocked or unblocked. If the data set contains unblocked format-F records, one record constitutes one block. If the data set contains blocked format-F records, the number of records within a block typically is constant for every block in the data set. The data set can contain truncated (short) blocks. The system automatically checks the length (except for card readers) on blocked or unblocked format-F records. Allowances are made for truncated blocks.

Format-F records can be used in any type of data set.

The optional control character (a), used for stacker selection or carriage control, can be included in each record to be printed or punched. The optional table reference character (b) is a code to select the font to print the record on a page printer. See “Using Optional Control Characters” on page 308 and “Table Reference Character” on page 310.

Standard Format

During creation of a sequential data set (to be processed by BSAM or QSAM) with fixed-length records, the RECFM subparameter of the DCB macro can specify a

standard format (RECFM=FS or FBS). A sequential data set with standard format records (format-FS or -FBS) sometimes can be read more efficiently than a data set with format-F or -FB records. This efficiency is possible because the system is able to determine the address of each record to be read, because each track contains the same number of blocks.

A standard-format data set must conform to the following specifications:

- All records in the data set are format-F records.
- No block except the last block is truncated. (With BSAM, you must ensure that this specification is met.) If the last block is truncated, the system writes it where a full size block would have been written.
- Every track except the last contains the same number of blocks.
- Every track except the last is filled as determined by the track capacity formula established for the device.
- The data set organization is physically sequential. You cannot use format-FS for a PDS or PDSE.

Restrictions

If the last block is truncated, you should never extend a standard-format data set by coding:

- EXTEND or OUTINX on the OPEN macro
- OUTPUT, OUTIN, or INOUT on the OPEN macro with DISP=MOD on the allocation
- CLOSE LEAVE, TYPE=T, and then a WRITE
- POINT to after the last block and then a WRITE
- CNTRL on tape to after the last block, and then a WRITE

If the data set becomes extended, it contains a truncated block that is not the last block. Reading an extended data set with this condition results in a premature end-of-data condition when the truncated block is read, giving the appearance that the blocks following this truncated block do not exist.

Standard-format data sets that end in a short block on magnetic tape should not be read backward because the data set would begin with a truncated block.

A format-F data set will not meet the requirements of a standard-format data set if you do the following:

- Extend a fixed-length, blocked standard data set when the last block was truncated.
- Use the POINT macro to prevent BSAM from filling a track other than the last one. Do not skip a track when writing to a data set.

Standard format should not be used to read records from a data set that was created using a record format other than standard, because other record formats might not create the precise format required by standard.

If the characteristics of your data set are altered from the specifications described above at any time, the data set should no longer be processed with the standard format specification.

Variable-Length Record Formats

In a variable-length record data set, each record or record segment can have a different length. Variable-length records can be used with all types of data sets. The variable-length record formats are format-V and format-D. They can also be spanned format (-VS or -DS), blocked format (-VB or -DB), or both format (-VBS and -DBS). Format-D, -DS, and -DBS records are used for ISO/ANSI tape data sets.

Format-V Records

Figure 45 shows blocked and unblocked variable-length (format-V) records without spanning. A block in a data set containing unblocked records is in the same format as a block in a data set containing blocked records. The only difference is that with blocked records each block can contain multiple records.

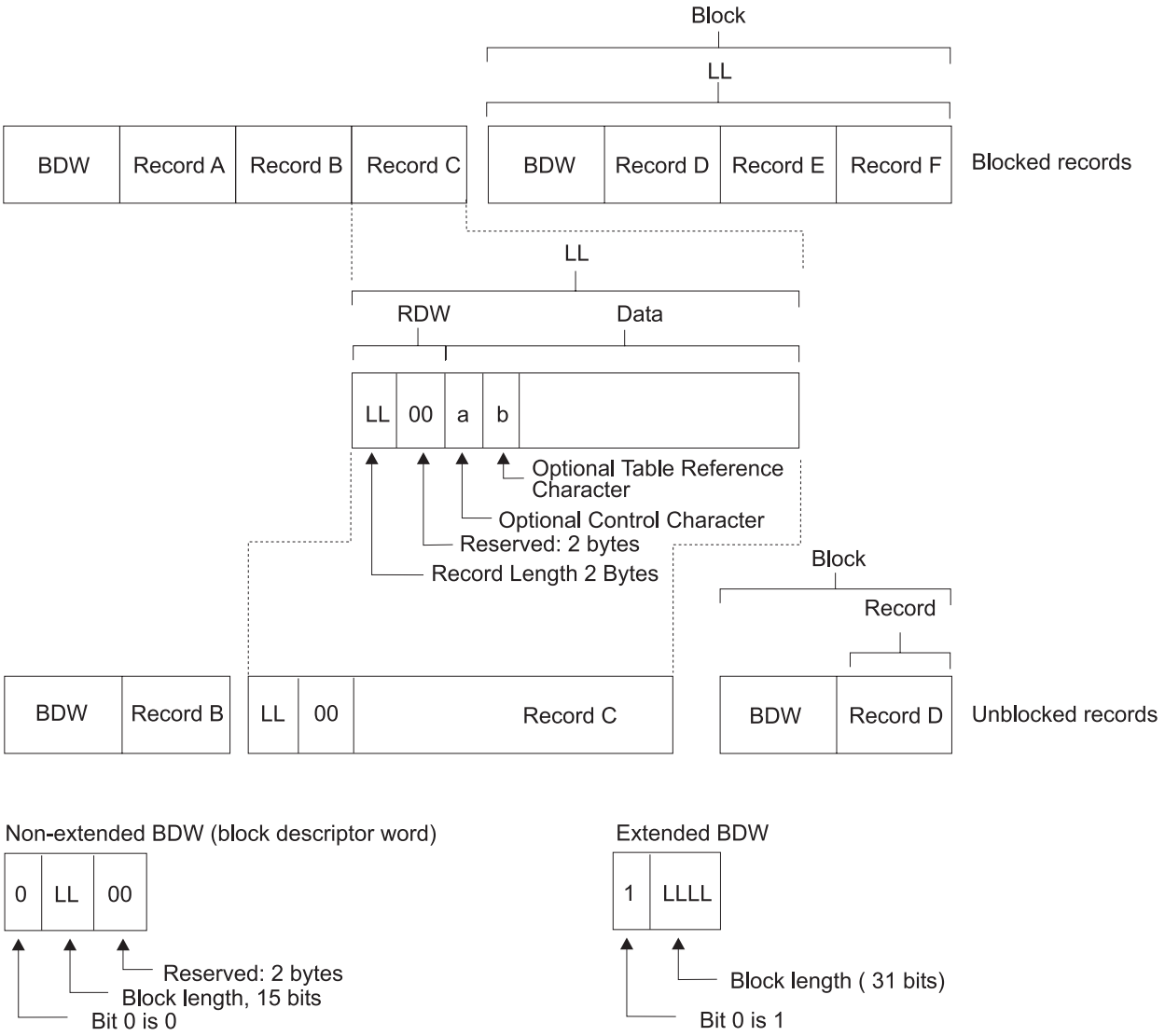


Figure 45. Nonspanned, Format-V Records

The system uses the record or segment length information in blocking and unblocking. The first four bytes of each record, record segment, or block make up a

descriptor word containing control information. You must allow for these additional 4 bytes in both your input and output buffers.

Block Descriptor Word (BDW)

A variable-length block consists of a block descriptor word (BDW) followed by one or more logical records or record segments. The block descriptor word is a 4-byte field that describes the block. It specifies the 4 byte block length for the BDW plus the total length of all records or segments within the block.

There are two types of BDW. If bit 0 is zero, it is a nonextended BDW. Bits 1-15 contain the block length. Bits 16-31 are zeroes. The block length can be from 8 to 32 760 bytes. All access methods and device types support nonextended BDWs.

If bit 0 of the BDW is one, the BDW is an extended BDW and BDW bits 1-31 contain the block length. Extended BDWs are currently supported only on tape.

When writing, BSAM applications provide the BDW; for QSAM, the access method creates the BDW. BSAM accepts an extended BDW if large block interface (LBI) processing has been selected (DCBESLBI in the DCBE control block is set on) and the output device is a magnetic tape. If an extended BDW is encountered and you are not using LBI, or the output device is not magnetic tape, an ABEND 002 is issued. IBM recommends that the BSAM user not provide an extended BDW unless the block length is greater than 32 760 because an extended BDW would prevent SAM reading the data on lower-level DFSMS systems. (A coexistence PTF for DFSMS/MVS® 1.5 and earlier results in SAM issuing an ABEND 002 if it reads a block containing an extended BDW.) Other programs that read the data set may also not support an extended BDW. QSAM creates extended BDWs only for blocks whose length is greater than 32 760, otherwise the nonextended format is used. When you read with either BSAM or QSAM, the access method interrogates the BDW to determine its format.

See “Large Block Interface (LBI)” on page 323.

Record Descriptor Word (RDW)

A variable-length logical record consists of a record descriptor word (RDW) followed by the data. The record descriptor word is a 4 byte field describing the record. The first 2 bytes contain the length (LL) of the logical record (including the 4 byte RDW). The length can be from 4 to 32 760. All bits of the third and fourth bytes must be 0, because other values are used for spanned records.

For output, you must provide the RDW, except in data mode for spanned records (described under “Controlling Buffers” on page 346). For output in data mode, you must provide the total data length in the physical record length field (DCBPRECL) of the DCB.

For input, the operating system provides the RDW, except in data mode. In data mode, the system passes the record length to your program in the logical record length field (DCBLRECL) of the DCB.

The optional control character (*a*) can be specified as the fifth byte of each record. The first byte of data is a table reference character (*b*) if OPTCD=J has been specified. The RDW, the optional control character, and the optional table reference character are not punched or printed.

Spanned Format-VS Records (Sequential Access Method)

Figure 46 shows how the spanning feature of the queued and basic sequential access methods lets you create and process variable-length logical records that are larger than one physical block. It also lets you pack blocks with variable-length records by splitting the records into segments so that they can be written into more than one block.

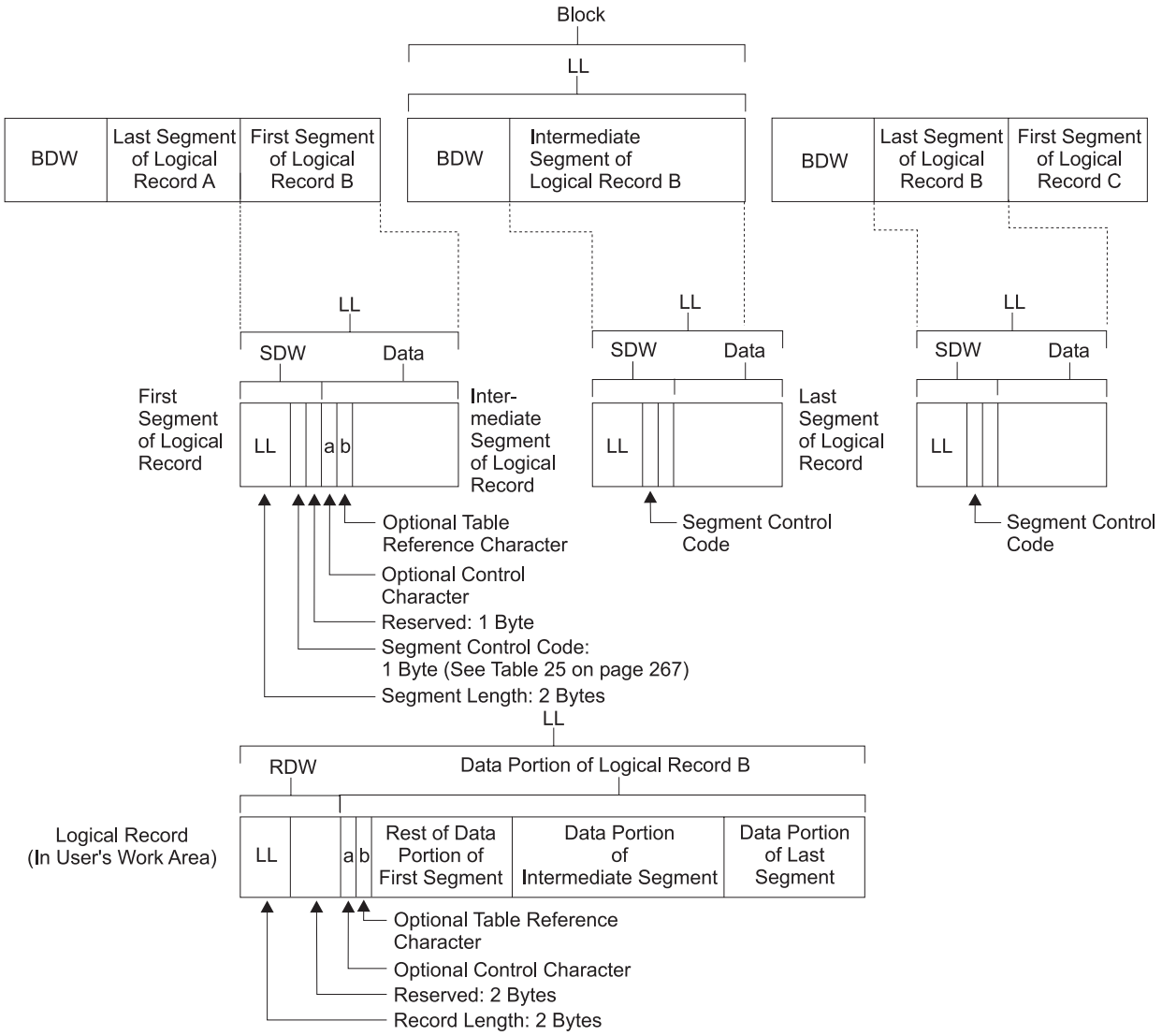


Figure 46. Spanned Format-VS Records (Sequential Access Method)

The format of the BDW is as described in Figure 45 on page 292.

When spanning is specified for blocked records, QSAM attempts to fill all blocks. For unblocked records, a record larger than the block size is split and written in two or more blocks. If your program is not using the large block interface, each block contains only one record or record segment. Thus, the block size can be set to the best block size for a given device or processing situation. It is not restricted by the maximum record length of a data set. A record can, therefore, span several blocks, and can even span volumes.

Spanned record blocks can have extended BDWs. See “Block Descriptor Word (BDW)” on page 293.

When you use unit record devices with spanned records, the system assumes that it is processing unblocked records and that the block size must be equivalent to the length of one print line or one card. The system writes records that span blocks one segment at a time.

Spanned variable-length records cannot be specified for a SYSIN data set.

Restrictions in Processing Spanned Records with QSAM

When spanned records span volumes, reading errors could occur when using QSAM, if a volume that begins with a middle or last segment is mounted first or if an FEOV macro is issued followed by another GET. QSAM cannot begin reading from the middle of the record. The errors include duplicate records, program checks in the user’s program, and invalid input from the spanned record data set.

A logical record spanning three or more volumes cannot be processed in update mode (as described in “Controlling Buffers” on page 346) by QSAM. For blocked records, a block can contain a combination of records and record segments, but not multiple segments of the same record unless the program is using LBI. When records are added to or deleted from a data set, or when the data set is processed again with different block size or record size parameters, the record segmenting changes.

When QSAM opens a spanned record data set in UPDAT mode, it uses the logical record interface (LRI) to assemble all segments of the spanned record into a single, logical input record, and to disassemble a single logical record into multiple segments for output data blocks. A record area must be provided by using the BUILDRCD macro or by specifying BFTEK=A in the DCB.

When you specify BFTEK=A, the open routine provides a record area equal to the LRECL specification, which should be the maximum length in bytes. (An LRECL=0 is not valid.)

Segment Descriptor Word

Each record segment consists of a segment descriptor word (SDW) followed by the data. The segment descriptor word, similar to the record descriptor word, is a 4 byte field that describes the segment. The first 2 bytes contain the length (LL) of the segment, including the 4 byte SDW. The length can be from 5 to 32 756 bytes or, when you are using WRITE with tape, from 18 to 32 756 bytes. The third byte of the SDW contains the segment control code that specifies the relative position of the segment in the logical record. The segment control code is in the rightmost 2 bits of the byte. The segment control codes are shown in Table 25.

Table 25. Segment Control Codes

Binary Code	Relative Position of Segment
00	Complete logical record
01	First segment of a multisegment record
10	Last segment of a multisegment record
11	Segment of a multisegment record other than the first or last segment

The remaining bits of the third byte and all of the fourth byte are reserved for possible future system use and must be 0.

The SDW for the first segment replaces the RDW for the record after the record is segmented. You or the operating system can build the SDW, depending on which access method is used.

- In the basic sequential access method, you must create and interpret the spanned records yourself.
- In the queued sequential access method move mode, complete logical records, including the RDW, are processed in your work area. GET consolidates segments into logical records and creates the RDW. PUT forms segments as required and creates the SDW for each segment.

  Data mode is similar to move mode, but allows reference only to the data portion of the logical record (that is, to one segment) in your work area. The logical record length is passed to you through the DCBLRECL field of the data control block.

  In locate mode, both GET and PUT process one segment at a time. However, in locate mode, if you provide your own record area using the BUILDRCD macro, or if you ask the system to provide a record area by specifying BFTEK=A, then GET, PUT, and PUTX process one logical record at a time.

Records Longer than 32 760 Bytes

A spanned record (RECFM=VS or RECFM=VBS) can contain logical records of any length, because it can contain any number of segments. While each segment must be less than 32 760, the segments concatenated together into the logical record can be longer than 32 760 bytes. A variable-length ISO/ANSI tape record (RECFM=D) can be longer than 32 760 bytes. Here are some techniques for processing records longer than 32 760 bytes.

1. If you use QSAM with BFTEK=A, but do not use the BUILDRCD macro to create the assembly area, you can create a record of up to 32 760 bytes long.
2. If you use QSAM locate mode and specify LRECL=X in your DCB macro, you can process logical records that exceed 32 760 bytes. Instead of assembling the record segments into one logical record, QSAM will give you one segment at a time. Then, you must concatenate the segments together into one logical record.
3. If you use BSAM and specify LRECL=X in your DCB macro, you can process logical records that exceed 32 760 bytes. You need to concatenate the segments together into one logical record.

You cannot use BFTEK=A or the BUILDRCD macro when the logical records exceed 32 760 bytes. (BFTEK=A is ignored when LRECL=X is specified.)

Null Segments

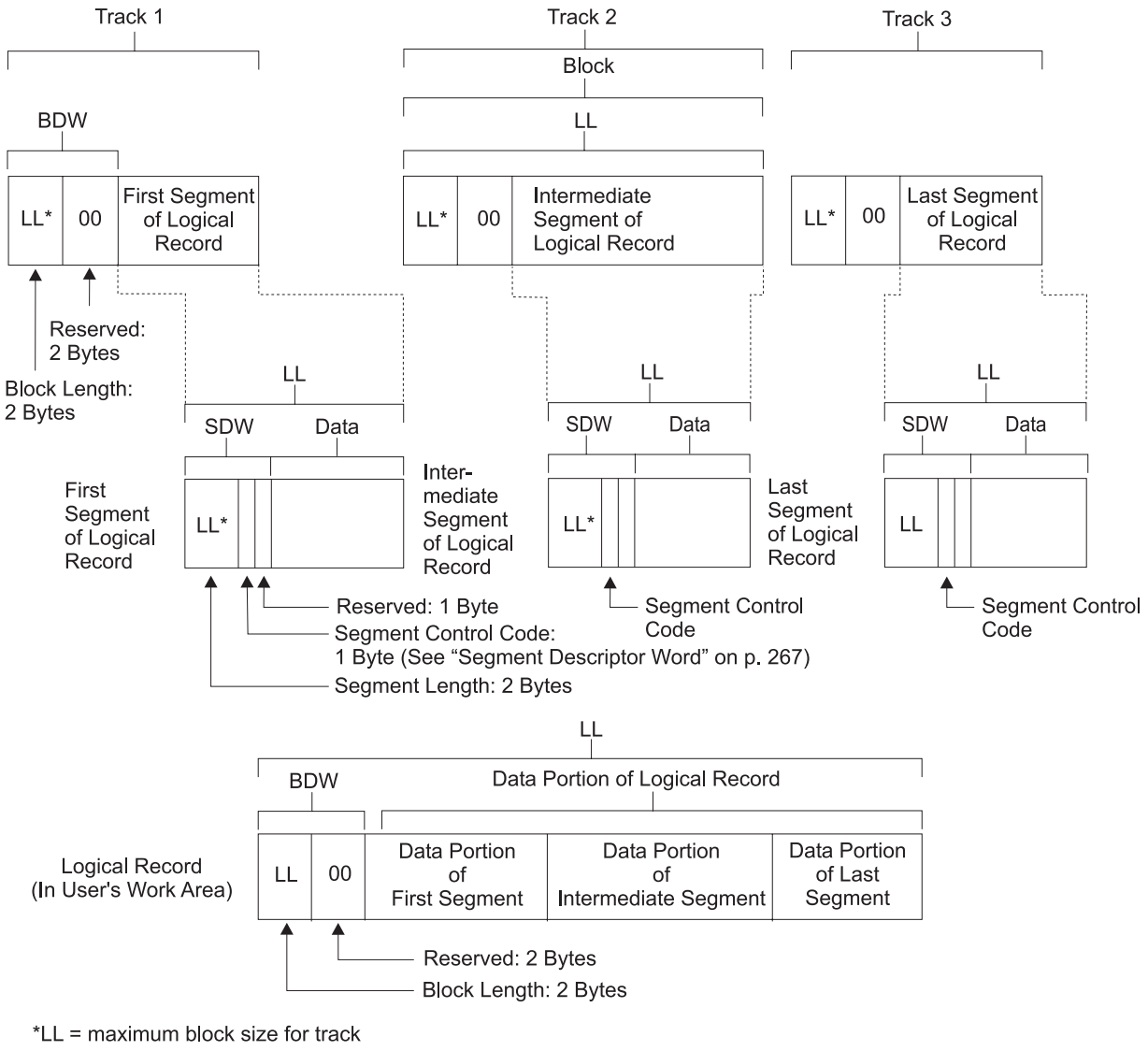
A 1 in bit position 0 of the SDW indicates a null segment. A null segment means that there are no more segments in the block. Bits 1-7 of the SDW and the remainder of the block must be binary zeros. A null segment is not an end-of-logical-record delimiter. (You do not have to be concerned about null segments unless you have created a data set using null segments.)

Null segments are not recreated in PDSEs. For more information, see “Processing PDSE Records” on page 434

Spanned Format-V Records (Basic Direct Access Method)

The spanning feature of BDAM lets you create and process variable-length unblocked logical records that span tracks. The feature also lets you pack tracks

with variable-length records by splitting the records into segments. Figure 47 shows how these segments can then be written onto more than one track.



*LL = maximum block size for track

Figure 47. Spanned Format-V Records for Direct Data Sets

When you specify spanned, unblocked record format for the basic direct access method, and when a complete logical record cannot fit on the track, the system tries to fill the track with a record segment. Thus, the maximum record length of a data set is not restricted by track capacity. Segmenting records permits a record to span several tracks, with each segment of the record on a different track. However, because the system does not permit a record to span volumes, all segments of a logical record in a direct data set are on the same volume.

Recommendation: Do not use the basic direct access method (BDAM).

Undefined-Length Record Format

Format-U permits processing of records that do not conform to the F- or V- format. Figure 48 shows how each block is treated as a record; therefore, any unblocking that is required must be performed by your program. The optional control character can be used in the first byte of each record. Because the system does not do length checking on format-U records, you can design your program to read less than a complete block into virtual storage. However, for extended format data sets, since the system writes maximum length records, you must provide an area at least as large as the block size of the data set. With BSAM the system attempts to read as much data as indicated by the current value of the BLKSIZE in the DCB or DCBE. When you are reading an extended format data set, make sure that the DCB or DCBE BLKSIZE field value is no more than the length of the area you are reading into. If you supply a short area because you know the next block is short, BSAM can overlay storage to the length limit set by the current BLKSIZE value.

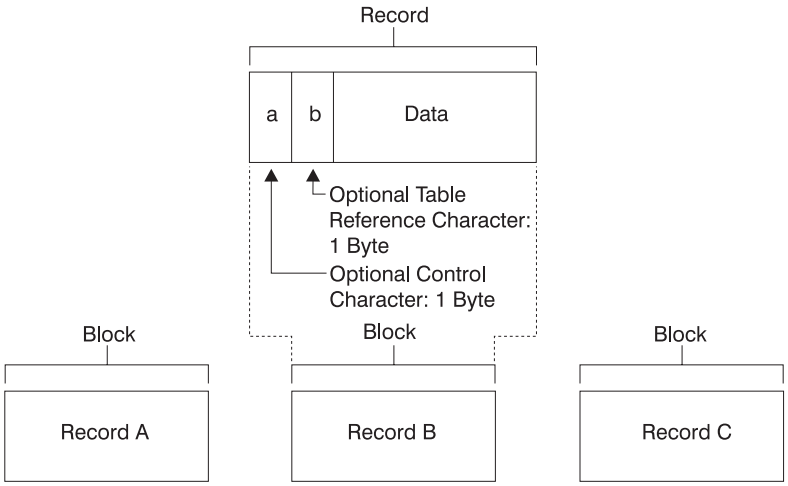


Figure 48. Undefined-Length Records

For format-U records, you must specify the record length when issuing the WRITE, PUT, or PUTX macro. No length checking is performed by the system, so no error indication will be given if the specified length does not match the buffer size or physical record size.

In update mode, you must issue a GET or READ macro before you issue a PUTX or WRITE macro to a data set on a direct access storage device. If you change the record length when issuing the PUTX or WRITE macro, the record will be padded with zeros or truncated to match the length of the record received when the GET or READ macro was issued. No error indication will be given.

ISO/ANSI Tapes

ISO/ANSI tape records are written in format-F, format-D, format-S, or format-U.

Character Data Conversion

Data management lets you convert from one character representation to another when using ISO/ANSI tapes. Conversion occurs according to one of the following techniques:

- **Coded Character Set Identifier (CCSID) Conversion.** CCSID conversion provides data management conversion to convert records between one CCSID which defines the character representation of the data in the records on tape to another CCSID which defines the character representation of the data in the records used by the application program. You can request that BSAM or QSAM perform this type of conversion for ISO/ANSI V4 tapes by supplying a CCSID in the CCSID parameter of a JOB statement, EXEC statement, and/or DD statement as well as through dynamic allocation or TSO ALLOCATE. CCSIDs are ignored if specified for other than ISO/ANSI V4 tapes.

  The CCSID which describes the data residing on the tape is taken from (in order of precedence):

  1. The CCSID supplied on the DD statement, or dynamic allocation, or TSO ALLOCATE.
  2. The CCSID field stored in the tape label.
  3. The default (to CCSID of 367 representing 7-bit ASCII) if a CCSID has been supplied for the application program.

  The CCSID that describes the data to use by the application program is taken from (in order of precedence):

  1. The CCSID supplied on the EXEC statement.
  2. The CCSID supplied on the JOB statement.
  3. The default (to CCSID of 500 representing International EBCDIC) if a CCSID has been supplied for the tape data.

  Data records can contain any character data as defined by the CCSID in which it was created.

  You can prevent access method conversion by supplying a special CCSID of 65535. In this case, data management transfers the data between the tape and the application program without conversion.

  See “Appendix F. Converting Character Sets” on page 593 for a list of supported CCSID combinations and “CCSID Decision Tables” on page 605 for a description of CCSID processing rules.

Restrictions: The following restrictions apply when CCSID conversion is used:

- Only SBCS to SBCS or DBCS to DBCS is supported. For more information about double byte character sets (DBCS), see Appendix B, ″Using the Double Byte Character Set″.
- When converting from one CCSID to another, changes in length for data records are not supported and will result in an error.
- All data management calls (OPEN, READ/WRITE, GET/PUT, CLOSE) must be made in the original key of the task (TCBPKF). Key switching is not supported and results in an error.
- All data management calls must be made in the task in which the DCB was opened. Subtasking is not supported and will result in an error.
- Supervisor state callers are not supported for any data management calls and results in an error.

• **Default character conversion.** Data management provides conversion from ASCII to EBCDIC on input, and EBCDIC to ASCII for output in any of the following cases (see “Tables for Default Conversion Codes” on page 610):

- ISO/ANSI V1 and V3 tapes
- ISO/ANSI V4 tapes without CCSID
- Unlabeled tapes with OPTCD=Q

Related reading: For information about conversion routines that the system supplies for this type of conversion, which converts to and from ASCII 7-bit code, see *z/OS DFSMS: Using Magnetic Tapes*.

When you convert from ASCII to EBCDIC, if a source character contains a bit in the high-order position, the 7-bit conversion does not produce an equivalent character. Instead, it produces a substitute character to note the loss in conversion. This means, for example, that the system cannot record random binary data (such as a dump) in ASCII 7-bit code.

The system cannot use CCSID conversion to read or write to an existing data set that was created using default character conversion, unless DISP=OLD.

When you use CCSIDs, the closest equivalent to default character conversion is between a CCSID of 367, which represents 7-bit ASCII, and a CCSID of 500, which represents International EBCDIC.

Format-F Records

For ISO/ANSI tapes, format-F records are the same as described in “Fixed-Length Record Formats” on page 290, except for control characters, block prefixes, and circumflex characters.

Control Characters. Control characters, when present, must be ISO/ANSI control characters. For more information about control characters see *z/OS DFSMS Macro Instructions for Data Sets*.

Block Prefixes. Record blocks can contain block prefixes. The block prefix can vary from 0 to 99 bytes, but the length must be constant for the data set being processed. For blocked records, the block prefix precedes the first logical record. For unblocked records, the block prefix precedes each logical record.

Using QSAM and BSAM to read records with block prefixes requires that you specify the BUFOFF parameter in the DCB. When using QSAM, you do not have access to the block prefix on input. When using BSAM, you must account for the block prefix on both input and output. When using either QSAM or BSAM, you must account for the length of the block prefix in the BLKSIZE and BUFL parameters of the DCB.

When you use BSAM on output records, the operating system does not recognize a block prefix. Therefore, if you want a block prefix, it must be part of your record. Note that you cannot include block prefixes in QSAM output records.

The block prefix can only contain EBCDIC characters that correspond to the 128, seven-bit ASCII characters. Thus, you must avoid using data types such as binary, packed decimal, and floating point that cannot always be converted into ASCII. This is also true when CCSIDs are used when writing to ISO/ANSI V4 tapes.

Related reading: For information about conversion routines supplied by the system for this type of conversion, which converts to ASCII 7-bit code, see *z/OS DFSMS: Using Magnetic Tapes*.

Figure 49 shows the format of fixed-length records for ISO/ANSI tapes and where control characters and block prefixes are positioned if they exist.

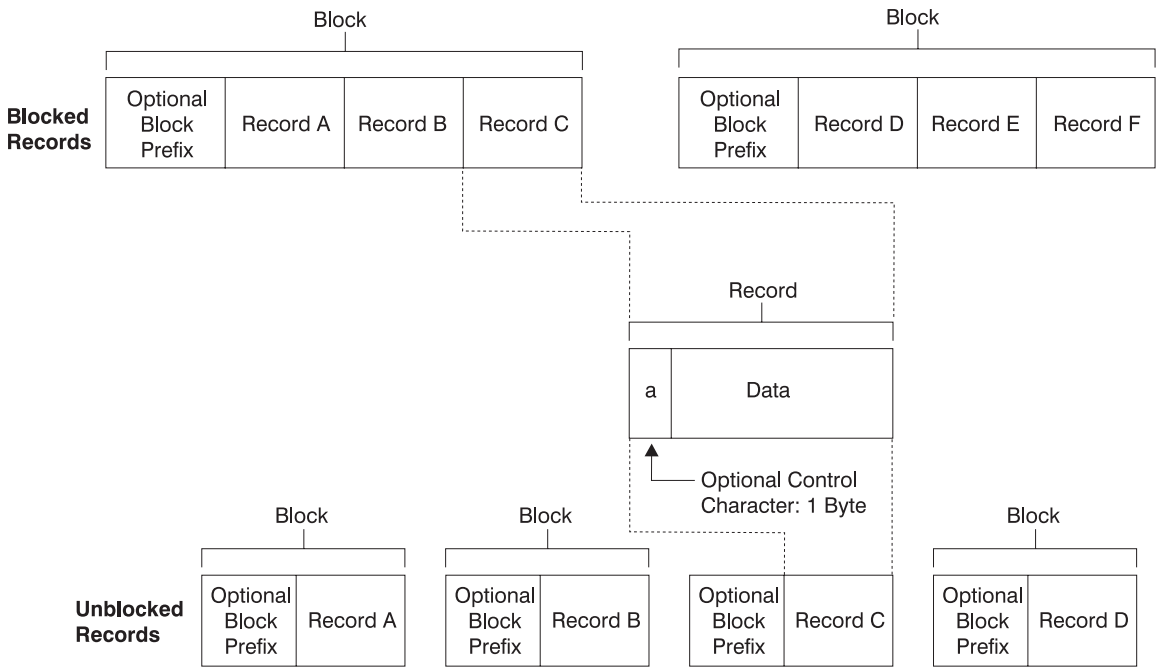


Figure 49. Fixed-Length Records for ISO/ANSI Tapes

Circumflex Characters. The GET routine tests each record (except the first) for all circumflex characters (X'5E'). If a record completely filled with circumflex characters is detected, QSAM ignores that record and the rest of the block. A fixed-length record must not consist of only circumflex characters. This restriction is necessary because circumflex characters are used to pad out a block of records when fewer than the maximum number of records are included in a block, and the block is not truncated.

Format-D Records

Format-D, format-DS, and format-DBS records are used for ISO/ANSI tape data sets. ISO/ANSI records are the same as format-V records, with three exceptions:

- Block prefix
- Block size
- Control characters.

Block Prefix. A record block can contain a block prefix. To specify a block prefix, code BUFOFF in the DCB macro. The block prefix can vary in length from 0 to 99 bytes, but its length must remain constant for all records in the data set being processed. For blocked records, the block prefix precedes the RDW for the first or only logical record in each block. For unblocked records, the block prefix precedes the RDW for each logical record.

To specify that the block prefix is to be treated as a BDW by data management for format-D or format-DS records on output, code BUFOFF=L as a DCB parameter. Your block prefix must be 4 bytes long, and it must contain the length of the block, including the block prefix. The maximum length of a format-D or format-DS, BUFOFF=L block is 9999 because the length (stated in binary numbers by the user) is converted to a 4 byte ASCII character decimal field on the ISO/ANSI tape when the block is written. It is converted back to a 2 byte length field in binary followed by two bytes of zeros when the block is read.

If you use QSAM to write records, data management fills in the block prefix for you. If you use BSAM to write records, you must fill in the block prefix yourself. If you are using chained scheduling to read blocked DB or DBS records, you cannot code BUFOFF=*absolute expression* in the DCB. Instead, BUFOFF=L is required, because the access method needs binary RDWs and valid block lengths to unblock the records.

When you use QSAM, you cannot read the block prefix into your record area on input. When using BSAM, you must account for the block prefix on both input and output. When using either QSAM or BSAM, you must account for the length of the block prefix in the BLKSIZE and BUFL parameters.

When using QSAM to access DB or DBS records, and BUFOFF=0 is specified, the value of BUFL, if specified, must be increased by 4. If BUFL is not specified, then BLKSIZE must be increased by 4. This permits a 4 byte QSAM internal processing area to be included when the system acquires the buffers. These 4 bytes do not become part of the user's block.

When you use BSAM on output records, the operating system does not recognize the block prefix. Therefore, if you want a block prefix, it must be part of your record.

The block prefix can contain only EBCDIC characters that correspond to the 128, seven-bit ASCII characters. Thus, you must avoid using data types (such as binary, packed decimal, and floating point), that cannot always be converted into ASCII. For DB and DBS records, the only time the block prefix can contain binary data is when you have coded BUFOFF=L, which tells data management that the prefix is a BDW. Unlike the block prefix, the RDW must always be binary. This is true whether conversion or no conversion is specified with CCSID for Version 4 tapes.

Block Size. Version 3 tapes have a maximum block size of 2048. This limit can be overridden by a label validation installation exit. For Version 4 tapes, the maximum size is 32 760.

If you specify a maximum data set block size of 18 or greater when creating variable-length blocks, then individual blocks can be shorter than 18 bytes. In those cases data management pads each one to 18 bytes when the blocks are written onto an ISO/ANSI tape. The padding character used is the ASCII circumflex character, which is X'5E'.

Control Characters. Control characters, if present, must be ISO/ANSI control characters. For more information about control characters see *z/OS DFSMS Macro Instructions for Data Sets*.

Figure 50 shows the format of nonspanned variable-length records for ISO/ANSI tapes, where the record descriptor word (RDW) is located, and where block prefixes and control characters must be placed when they are used.

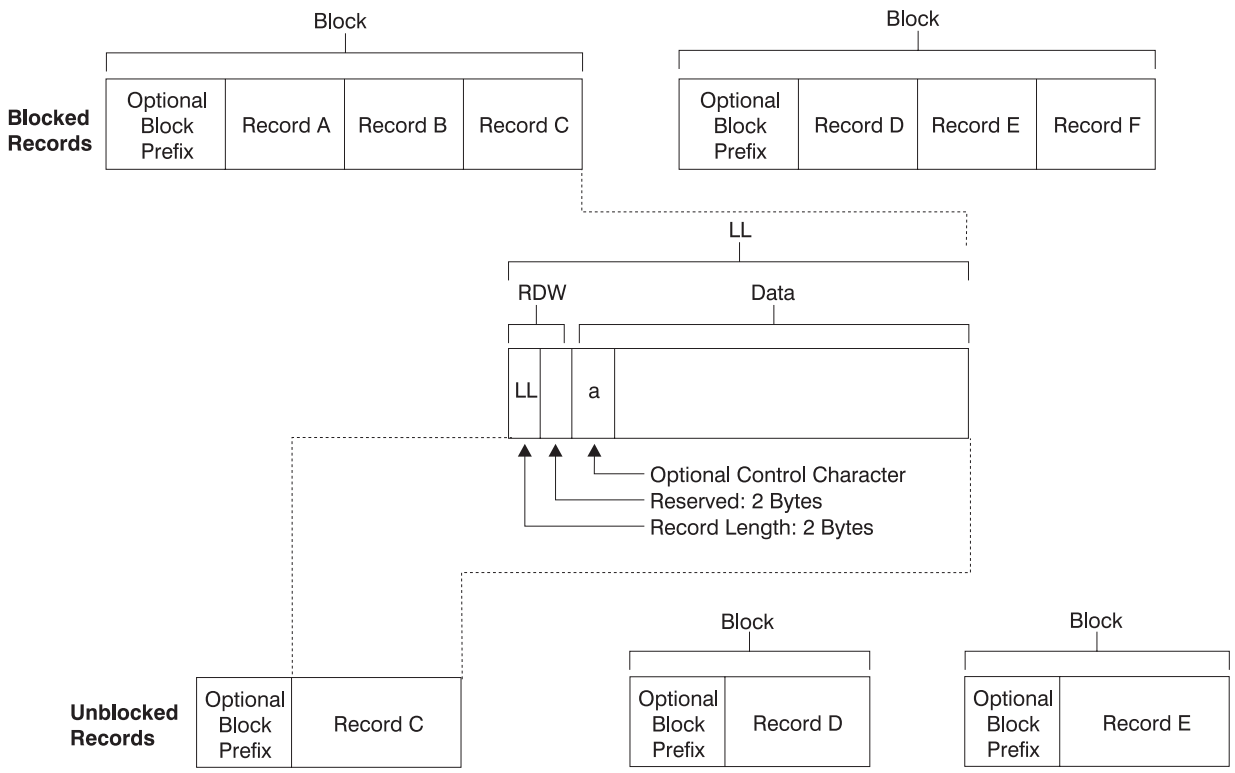


Figure 50. Nonspanned Format-D Records for ISO/ANSI Tapes As Seen by the Program

ISO/ANSI Format-DS and Format-DBS Records

For ISO/ANSI tapes, variable-length spanned records must be specified in the DCB RECFM parameter as DCB RECFM=DS or DBS. Format-DS and format-DBS records are similar to format-VS or format-VBS records. The exceptions are described in “Converting the Segment Descriptor Word” on page 305 and “Processing Records Longer than 32 760 Bytes” on page 306.

Figure 51 on page 305 shows what spanned variable-length records for ISO/ANSI tapes look like.

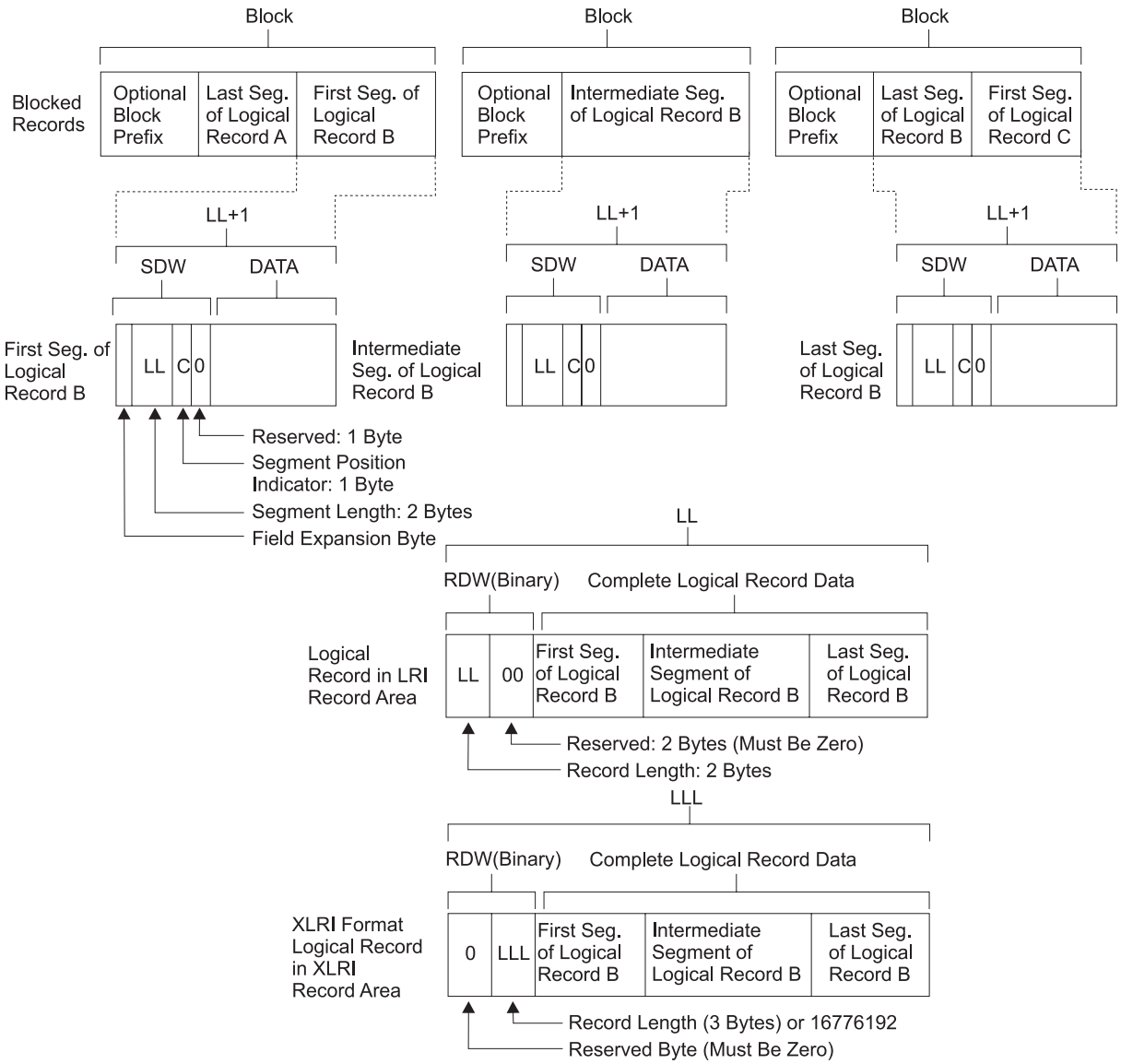


Figure 51. Spanned Variable-Length (Format-DS) Records for ISO/ANSI Tapes As Seen by the Program

Figure 51 shows the segment descriptor word (SDW), where the record descriptor word (RDW) is located, and where block prefixes must be placed when they are used. If you are not using IBM access methods see *z/OS DFSMS Macro Instructions for Data Sets* for a description of ISO/ANSI record control words and segment control words.

Converting the Segment Descriptor Word

There is an additional byte preceding each SDW for DS/DBS records. This additional byte is required for conversion of the SDW from IBM to ISO/ANSI format, because the ISO/ANSI SDW (called a segment control word) is five bytes long. Otherwise, the SDW for DS/DBS records is the same as the SDW for VS/VBS records. The SDW LL count excludes the additional byte.

QSAM or BSAM convert between ISO/ANSI segment control word (SCW) format and IBM segment descriptor word (SDW) format. On output, the binary SDW LL value (provided by you when using BSAM and by the access method when using QSAM), is increased by 1 for the extra byte and converted to four ASCII numeric

characters. Because the binary SDW LL value will result in four numeric characters, the binary value must not be greater than 9998. The fifth character is used to designate which segment type (complete logical record, first segment, last segment, or intermediate segment) is being processed.

On input, the four numeric characters designating the segment length are converted to two binary SDW LL bytes and decreased by one for the unused byte. The ISO/ANSI segment control character maps to the DS/DBS SDW control flags. This conversion leaves an unused byte at the beginning of each SDW. It is set to X'00'. See *z/OS DFSMS Macro Instructions for Data Sets* for more details on this process.

On the tape, the SDW bytes are ASCII numeric characters even if the other bytes in the record are not ASCII.

Processing Records Longer than 32 760 Bytes

A spanned record (RECFM=VS or RECFM=VBS) can contain logical records of any length, because it can contain any number of segments. While each segment must be less than 32 760, the segments concatenated together into the logical record can be longer than 32 760 bytes. A variable-length ISO/ANSI tape record (RECFM=D) can be longer than 32 760 bytes. Some techniques for processing records longer than 32 760 bytes follow.

Processing techniques for Format-D, Format-DS or Format-DBS Records (ISO/ANSI Tapes):

- If you use QSAM or BSAM and specify LRECL=X, you can process records longer than 32 760 bytes for ISO/ANSI tapes. Note that the maximum block length for ISO/ANSI tapes is 2048.
- If you use QSAM with XLRI and specify LRECL=nnnnnK or 0K, you can process records longer than 32 760 bytes for variable-length, spanned ISO/ANSI tapes.

DS/DBS records with a record length of over 32 760 bytes can be processed using XLRI. (XLRI is supported only in QSAM locate mode for ISO/ANSI tapes.) Using the LRECL=X for ISO/ANSI causes an 013-DC ABEND.

To use XLRI, specify LRECL=0K or LRECL=nK in the DCB macro. Specifying DCBLRECL with the K suffix sets the DCBBFTK bit that indicates that LRECL is coded in K units and that the DCB is to be processed in XLRI mode.

LRECL=0K in the DCB macro specifies that the LRECL value will come from the file label or JCL. When LRECL is from the label, the file must be opened as an input file. The label (HDR2) value for LRECL will be converted to kilobytes and rounded up when XLRI is in effect. When the ISO/ANSI label value for LRECL is 00000 to show that the maximum record length can be greater than 99999, you must use LRECL=nK in the JCL or in the DCB to specify the maximum record length.

You can express the LRECL value in JCL in absolute form or with the K notation. When the DCB specifies XLRI, the system converts absolute values to kilobytes by rounding up to an integral multiple of 1024. Absolute values are permissible only from 5 to 32 760.

To show the record area size in the DD statement, code LRECL=nK, or specify a data class that has the LRECL attribute you need. The value nK can range from 1K to 16 383K (expressed in 1024 byte multiples). However, depending on the buffer space available, the value you can specify in most systems will be much smaller than 16 383K bytes. This value is used to determine the size of the record area required to contain the largest logical record of the spanned format file.

When you use XLRI, the exact LRECL size is communicated in the three low-order bytes of the RDW in the record area. This special RDW format exists only in the record area to communicate the length of the logical record (including the 4 byte RDW) to be written or read. (See the XLRI format of the RDW in Figure 51 on page 305.) DCB LRECL shows the 1024 multiple size of the record area (rounded up to the next nearest kilobyte). The normal DS/DBS SDW format is used at all other times before conversion.

Processing DS/DBS Tapes with QSAM

When using QSAM, the same application used to process VS/VBS tape files can be used to process DS/DBS tape files. However, you must ensure that ISO/ANSI requirements such as block size limitation, tape device, and restriction to EBCDIC characters that correspond to the 128, seven-bit ASCII characters are met. The SCW/SDW conversion and buffer positioning is handled by the GET/PUT routines.

ISO/ANSI Version 4 tapes also permits special characters !*″%&’()+,-./:;<=>?_ and numeric characters 0 - 9.

Processing DS/DBS Tapes with BSAM

When using BSAM to process a DS/DBS tape file, you must allow for an additional byte at the beginning of each SDW. The SDW LL must exclude the additional byte. On input, you must ignore the unused byte preceding each SDW. On output, you must allocate the additional byte for each SDW.

Format-U Records

Data can only be in format-U for ISO/ANSI Version 1 tapes (ISO 1001-1969 and ANSI X3.27-1969). These records can be used for input only. They are the same as the format-U records described in “Undefined-Length Record Format” on page 298 except the control characters must be ISO/ANSI control characters, and block prefixes can be used.

Format-U records are not supported for Version 3 or Version 4 ISO/ANSI tapes. An attempt to process a format-U record from a Version 3 or Version 4 tape results in entering the label validation installation exit.

Record Format—Device Type Considerations

This section discusses which record formats are acceptable for specific devices.

DASD—Format-F, format-U, format-V
Magnetic tape—Format-D, format-F, format-U, format-V
Printer—Format-F, format-U, format-V
Card reader and punch—Format-F, format-U
SYSIN and SYSOUT—Format-F, format-U, format-V

For more information see “Chapter 25. Spooling and Scheduling Data Sets” on page 373.

Selecting Record Formats for Non-VSAM Data Sets

The device-dependent (DEVD) parameter of the DCB macro specifies the type of device where the data set's volume resides:

DA	Direct access storage devices
TA	Magnetic tape
PR	Printer
RD	Card reader
PC	Card punch

Note: Because the DEVD option affects only for the DCB macro expansion, you are guaranteed the maximum device flexibility by letting it default to DEVD=DA and not coding any device-dependent parameter.

Using Optional Control Characters

You can specify in the DD statement, the DCB macro, or the data set label that an optional control character is part of each record in the data set. The 1 byte character is used to show a carriage control function when the data set is printed or a stacker bin when the data set is punched. Although the character is a part of the record in storage, it is never printed or punched. Note that buffer areas must be large enough to accommodate the character.

If the immediate destination of the data set is a device, such as a disk or tape, which does not recognize the control character, the system assumes that the control character is the first byte of the data portion of the record. If the destination of the data set is a printer or punch and you have not indicated the presence of a control character, the system regards the control character as the first byte of data. If the destination of the data set is SYSOUT, the effect of the control characters is determined at the ultimate destination of the data set. See *z/OS DFSMS Macro Instructions for Data Sets* for a list of the control characters.

The presence of a control character is indicated by M or A in the RECFM field of the data control block. M denotes machine code; A denotes American National Standards Institute (ANSI) code. If either M or A is specified, the character must be present in every record; the printer space (PRTSP) or stacker select (STACK) field of the DCB is ignored.

The optional control character must be in the first byte of format-F and format-U records, and in the fifth byte of format-V records and format-D records where BUFOFF=L. If the immediate destination of the data set is a sequential DASD data set or an IBM standard or ISO/ANSI standard labelled tape, OPEN records the presence and type of control characters in the data set label. This is so that a program that copies the data set to a print, punch, or SYSOUT data set can propagate RECFM and therefore control the type of control character.

Using Direct Access Storage Devices (DASD)

Direct access storage devices accept records of format-F, format-V, or format-U. To read or write the records with keys, you must specify the key length (KEYLEN). In addition, the operating system has a standard track format for all direct access volumes. See “Track Format” on page 8 for a complete description of track format. Each track contains data information and certain control information, such as the following information:

- The address of the track
- The address of each record
- The length of each record
- Gaps between areas

Except for a PDSE or compressed format data set, the size of a block cannot exceed what the system can write on a track. For PDSEs and compressed format data sets, the access method simulates blocks, and you can select a value for BLKSIZE without regard to the track length. A compressed format data set is a type of extended format data set that is stored in a data format that can contain records that the access method compressed.

Using Magnetic Tape

Format-F, format-V, format-D, and format-U records are acceptable for magnetic tape. Format-V records are not acceptable on 7-track tape if the data conversion feature is not available. ASCII records are not acceptable on 7-track tape.

When you create a tape data set with variable-length record format-V or format-D, the control program pads any data block shorter than 18 bytes. For format-V records, it pads to the right with binary zeros so that the data block length equals 18 bytes. For format-D (ASCII) records, the padding consists of ASCII circumflex characters, which are equivalent to X'5E's.

Note that there is no minimum requirement for block size. However, in nonreturn-to-zero-inverted mode, if a data check occurs on a magnetic tape device, any record shorter than 12 bytes in a read operation will be treated as a noise record and lost. No check for noise will be made unless a data check occurs.

Table 26 shows how the tape density (DEN) specifies the recording density in bits per inch per track.

Table 26. Tape Density (DEN) Values

DEN	7-Track Tape	9-Track Tape
1	556 (NRZI)	N/A
2	800 (NRZI)	800 (NRZI)[1]
3	N/A	1600 (PE)[2]
4	N/A	6250 (GCR)[3]

Notes:

1. NRZI is for nonreturn-to-zero-inverted mode.
2. PE is for phase encoded mode.
3. GCR is for group coded recording mode.

When DEN is not specified, the highest density capable by the unit will be used. The DEN parameter has no effect on an 18-track or 36-track tape cartridge.

The track recording technique (TRTCH) for 7-track tape can be specified as follows.

C	Data conversion is to be used. Data conversion makes it possible to write 8 binary bits of data on 7 tracks. Otherwise, only 6 bits of an 8-bit byte are recorded. The length field of format-V records contains binary data and is not recorded correctly without data conversion.
E	Even parity is to be used. If E is omitted, odd parity is assumed.
T	BCDIC to EBCDIC conversion is required.

The track recording technique (TRTCH) for magnetic tape drives with Improved Data Recording Capability can be specified as:

COMP	Data is written in compacted format.
NOCOMP	Data is written in standard format.

The system programmer sets the 3480 default for COMP or NOCOMP in the DEVSUPxx member of SYS1.PARMLIB.

Using a Printer

Records of a data set that is written to a printer with BSAM or QSAM can contain control characters. See “Using Optional Control Characters” on page 308. Independently of whether the records contain control characters, they can contain table reference characters.

Table Reference Character

The table reference character is a numeric character that corresponds to the order in which you specified the character arrangement table names with the CHARS keyword. The system uses the table reference character for selection of a character arrangement table during printing.

A numeric table reference character (such as 0) selects the font to which the character corresponds. The characters’ number values represent the order in which you specified the font names with the CHARS parameter. In addition to using table reference characters that correspond to font names specified in the CHARS parameter, you can code table reference characters that correspond to font names specified in the PAGEDEF control structure. With CHARS, valid table reference characters vary and range between 0 and 3. With PAGEDEF, they range between 0 and 126. The system treats table reference characters with values greater than the limit as 0 (zero).

Indicate the presence of table reference characters by coding OPTCD=J in the DCB macro, in the DD statement, or in the dynamic allocation call.

The system processes table reference characters on printers such as the IBM 3800 and IBM 3900 that support the CHARS and PAGEDEF parameters on the DD statement. If the device is a printer that does not support CHARS or PAGEDEF, the system discards the table reference character. This is true both for printers that are allocated directly to the job step and for SYSOUT data sets. This makes it unnecessary for your program to know whether the printer supports table reference characters.

If the immediate destination of the data set for which OPTCD=J was specified is DASD, the system treats the table reference characters as part of the data. The system also records the OPTCD value in the data set label. If the immediate destination is tape, the system does not record the OPTCD value in the data set label.

Record Formats

The printer can accept format-F, format-V, and format-U records. The system does not print the first 4 bytes (record descriptor word or segment descriptor word) of format-V records or record segments. For format-V records, at least 1 byte of data must follow the record or segment descriptor word or the carriage control character. The system does not print the carriage control character, if you specify it in the RECFM parameter. The system does not position the printer to channel 1 for the first record unless you use a carriage control character to specify this position.

Because each line of print corresponds to one record, the record length should not exceed the length of one line on the printer. For variable-length spanned records, each line corresponds to one record segment; block size should not exceed the length of one line on the printer.

If you do not specify carriage control characters, you can specify printer spacing (PRTSP) as 0, 1, 2, or 3. If you do not specify PRTSP, the system assumes 1.

For all QSAM RECFM=FB printer data sets, the system adjusts the block size in the DCB to equal the logical record length. The system treats this data set as RECFM=F. If the system builds the buffers for this data set, the BUFL parameter determines the buffer length. If you do not specify the BUFL parameter, the system uses the adjusted block size for the buffer length.

To reuse the DCB with a block size larger than the logical record length, you must reset DCBBLKSI in the DCB and ensure that the buffers are large enough to contain the largest block size. To ensure the buffer size, specify the BUFL parameter before the first open of the data set. Or you can issue the FREEPOOL macro after each CLOSE macro, so that the system builds a new buffer pool of the correct size each time it opens the data set.

Using a Card Reader and Punch

Format-F and format-U records are acceptable to both the reader and the punch. Format-V records are acceptable to the punch only.The device control character, if specified in the RECFM parameter, is used to select the stacker; it is not punched. For control character information, see “Using Optional Control Characters” on page 308. The first 4 bytes (record descriptor word or segment descriptor word) of format-V records or record segments are not punched. For format-V records, at least 1 byte of data must follow the record or segment descriptor word or the carriage control character.

A record size of 80 bytes is called EBCDIC mode (E) and a record size of 160 bytes is called column binary mode (C). Each punched card corresponds to one physical record. Therefore, you should restrict the maximum record size to EBCDIC mode (80 bytes) or column binary mode (160 bytes). When column binary mode is used for the card punch, BLKSIZE must be 160 unless you are using PUT. Then you can specify BLKSIZE as 160 or a multiple of 160, and the system handles this as described under “PUT—Write a Record” on page 359. Specify the read/punch mode of operation (MODE) parameter as either card image column binary mode (C) or EBCDIC mode (E). If this information is omitted, E is assumed. The stacker

selection parameter (STACK) can be specified as either 1 or 2 to show which bin is to receive the card. If STACK is not specified, 1 is assumed.

For all QSAM RECFM=FB card punch data sets, the block size in the DCB is adjusted by the system to equal the logical record length. This data set is treated as RECFM=F. If the system builds the buffers for this data set, the buffer length is determined by the BUFL parameter. If the BUFL parameter was not specified, the adjusted block size is used for the buffer length.

If the DCB is to be reused with a block size larger than the logical record length, you must reset DCBBLKSI in the DCB and ensure that the buffers are large enough to contain the largest block size expected. You can ensure the buffer size by specifying the BUFL parameter before the first time the data set is opened, or by issuing the FREEPOOL macro after each CLOSE macro so the system will build a new buffer pool of the correct size each time the data set is opened.

Punch error correction on the IBM 2540 Card Read Punch is not performed.

The IBM 3525 Card Punch accepts only format-F records for print and associated data sets. Other record formats are permitted for the read data set, punch data set, and interpret punch data set.

Using a Paper Tape Reader

The system no longer supports paper tape readers (IBM 2671).

Chapter 22. Specifying and Initializing Data Control Blocks

This chapter covers the following topics.

Topic	Page
Processing Sequential and Partitioned Data Sets	314
Using OPEN to Prepare a Data Set for Processing	318
Selecting Data Set Options	322
Changing and Testing the DCB and DCBE	330
Using CLOSE to End the Processing of a Data Set	332
Opening and Closing Data Sets: Considerations	335
Positioning Volumes	338
Managing SAM Buffer Space	341
Constructing a Buffer Pool	342
Controlling Buffers	346
Choosing Buffering Techniques and GET/PUT Processing Modes	350
Using Buffering Macros with Queued Access Method	351
Using Buffering Macros with Basic Access Method	351

For each data set that you want to process, there must be a corresponding data control block (DCB) and data definition (DD) statement or its dynamic allocation equivalent. The characteristics of the data set and device-dependent information can be supplied by either source. As specified in *z/OS MVS JCL User's Guide* and *z/OS MVS JCL Reference*, the DD statement must also supply data set identification. Your program, SMS and various exit routines can supply device characteristics, space allocation requests, and related information. You establish the logical connection between a DCB and a DD statement by specifying the name of the DD statement in the DDNAME field of the DCB macro, or by completing the field yourself before opening the data set.

You can process a non-VSAM data set to read, update, or add data by following this procedure:

1. Create a data control block (DCB) to identify the data set to be opened. A DCB is required for each data set and is created in a processing program by a DCB macro.

   When the program is executed, the data set name and other important information (such as data set disposition) are specified in a JCL statement called the data definition (DD) statement, or in a call to dynamic allocation.
2. Optionally supply a data control block extension (DCBE). You can supply options and test data set characteristics that the system stores in the DCBE.
3. Connect your program to the data set you want to process, using the OPEN macro. The OPEN macro also positions volumes, writes data set labels and allocates virtual storage. There are various buffering macros and options that you can consider.

© Copyright IBM Corp. 1987, 2002

4. Request access to the data set. For example, if you are using BSAM to process a sequential data set, you can use the READ, WRITE, NOTE, or POINT macro.
5. Disconnect your program from the data set, using the CLOSE macro. The CLOSE macro also positions volumes, creates data set labels, completes writing queued output buffers, and frees virtual and auxiliary storage.

Primary sources of information to be placed in the data control block are a DCB macro, data definition (DD) statement, a dynamic allocation SVC 99 parameter list, a data class, and a data set label. A data class can be used to specify all of your data set's attributes except data set name and disposition. Also, you can provide or change some of the information during execution by storing the applicable data in the appropriate field of the DCB or DCBE.

Processing Sequential and Partitioned Data Sets

Data management is designed to provide a balance between ease of use, migration to new releases, coexistence with various levels of software and hardware, device independence, exploitation of hardware features, and performance. Sometimes these considerations can conflict. If your program exploits a particular model's features to maximize performance, it might not take full advantage of newer technology.

It is the intent of IBM that your programs that use documented programming interfaces and work on the current level of the system will run at least equally well on future levels of the system. However, IBM cannot guarantee that. Characteristics such as certain reason codes that are documented only in *z/OS DFSMSdfp Diagnosis Reference* are not part of the intended programming interface. Examples of potential problems are:

- Your program has a timing dependency such as a READ or WRITE macro completes before another event. In some cases READ or WRITE is synchronous with your program.
- Your program tests a field or control block that is not part of the intended programming interface. An example is status indicators not documented in Figure 107 on page 492.
- Your program relies on the system to enforce a restriction such as the maximum value of something. For example, the maximum block size on DASD used to be less than 32 760 bytes, the maximum NCP value for BSAM used to be 99 and the maximum block size on tape used to be 32 760.
- New releases might introduce new return and reason codes for system functions.

For these reasons, the operating system has many options. It is not the intent of IBM to require extensive education to use assembly language programming. The purpose of this section is to show how to read and write sequential data sets simply in High Level Assembler while maximizing ease of use, migration potential, the likelihood of coexistence, and device independence, while getting reasonable performance.

You can use the examples in this section to read or write sequential data sets and partitioned members. These include ordinary disk data sets, extended format data sets, compressed format data sets, PDS members, PDSE members, HFS files, USS FIFOs, spooled data sets (SYSIN and SYSOUT), real or VM simulated unit record devices, TSO/E terminals, magnetic tapes, dummy data sets, and most combinations of them in a concatenation.

Recommendations:

- Use QSAM because it is simpler. Use BSAM if you need to read or write nonsequentially or you need more control of I/O completion. With BSAM you can issue the NOTE, POINT, CNTRL, and BSP macros. These macros work differently on various device classes. See “Record Format—Device Type Considerations” on page 307 and “Achieving Device Independence” on page 387. Use BPAM if you need to access more than one member of a PDS or PDSE.
- Specify LRECL and RECFM in the DCB macro if your program’s logic depends on the record length and record format. If you omit either of them, your program is able to handle more types of data but you have to write more code. See “Chapter 21. Selecting Record Formats for Non-VSAM Data Sets” on page 289.
- Use format-F or format-V records, and specify blocking (RECFM=FB or VB). This allows longer blocks. Format-U generally is less efficient. Format-D works only on certain types of tape.
- Omit the block size in the DCB macro. Code BLKSIZE=0 in the DCBE macro to use the large block interface. When your program is reading, this allows it to adapt to the appropriate block size for the data set. If the data set has no label (such as for an unlabeled tape), the user can specify the block size in the DD statement or dynamic allocation. For some data set types (such as PDSEs and HFS files) there is no real block size; the system simulates any valid block size and there is a default.

  When your program is writing and you omit DCB BLKSIZE and code DCBE BLKSIZE=0, this enables the user to select the block size in the DD statement or dynamic allocation. The user should only do this if there is a reason to do so, such as a reading program cannot accept large blocks. If the user does not specify a block size, OPEN selects one that is valid for the LRECL and RECFM and is optimal for the device. Coding BLKSIZE=0 in the DCBE macro lets OPEN select a block size that exceeds 32 760 bytes if large block interface (LBI) processing is being used, thereby possibly shortening run time significantly. If OPEN might select a block size that is larger than the reading programs can handle, the user can code the BLKSZLIM keyword in the DD statement or the dynamic allocation equivalent or rely on the block size limit in the data class or in the DEVSUP*xx* parmlib member.

  If you want to provide your own default for BLKSIZE and not let OPEN do it, you can provide a DCB OPEN exit routine. See “DCB OPEN Exit” on page 509. The installation OPEN exit might override your program’s selection of DCB parameters.
- Omit BUFL (buffer length) because it relies on the value of the sum of BLKSIZE and KEYLEN and because it cannot exceed 32 760.
- Omit BUFNO (number of buffers) for QSAM, BSAM, and BPAM and NCP if you use BSAM or BPAM. Let OPEN select QSAM BUFNO. This is particularly important with striped data sets. The user can experiment with various values for QSAM BUFNO to see if it can improve run time.

  With BSAM and BPAM, code MULTSDN and MULTACC in the DCBE macro. See “Improving Performance for Sequential Data Sets” on page 390.

  With QSAM, BSAM, and BPAM this generally has no effect on the EXCP count that is reported in SMF type 14, 15, 21, and 30 records. On DASD, this counts blocks that are transferred and not the number of channel programs. This causes the counts to be repeatable and not to depend on random factors in the system.
- Omit BUFOFF because it works only with tapes with ISO/ANSI standard labels or no labels.

- If you choose BSAM or BPAM in 31-bit addressing mode, do not use the BUILD or GETPOOL macro and do not request OPEN to build a buffer pool. If you code a nonzero BUFNO value, you are requesting OPEN to build a buffer pool. Such a buffer pool resides below the line. Use your own code to allocate data areas above the line.
- Code A or M for RECFM or code OPTCD=J only if your program logic requires reading or writing control characters. These are not the EBCDIC or ASCII control characters such as carriage return, line feed, or new page.
- Omit KEYLEN, DEVD, DEN, TRTCH, MODE, STACK, and FUNC because they are device dependent. KEYLEN also makes the program run slower unless you code KEYLEN=0. The user can code most of them in the DD statement if needed.
- Omit BFALN, BFTEK, BUFCB, EROPT, and OPTCD because they probably are not useful, except OPTCD=J. OPTCD=J specifies that the records contain table reference characters. See “Table Reference Character” on page 310.
- LOCATE mode (MACRF=(GL,PL)) might be more efficient than move mode. This depends on your program’s logic. The move mode requires QSAM to move the data an extra time.
- If your program runs with 31-bit addressing mode (AMODE), code RMODE31=BUFF in the DCBE so that the QSAM buffers are above the 16 MB line. A nonreentrant, RMODE 24 program (residing below the 16 MB line) is simpler than a reentrant or RMODE 31 program because the DCB must reside below the line in storage that is separate for each open data set.
- Code a SYNAD (I/O error) routine to prevent the 001 ABEND that the system issues when a data set has an I/O error. In the SYNAD routine, issue the SYNADAF macro, write the message, and terminate the program. This writes a message and avoids a dump because the dump is not likely to be useful.
- Use extended-format data sets even if you are not using striping. They tend to be more efficient, and OPEN provides a more efficient default for BUFNO.

Figure 52 shows the simplest way to read a sequential data set.

```
         OPEN    (INDCB,INPUT)          Open to read
         LTR     R15,R15                Branch if DD name seems not
         BNZ    ...                      to be defined
* Loop to read all the records
LOOP     GET     INDCB                  Get address of a record in R1
         ...                            Process a record
         B       LOOP                   Branch to read next record
* I/O error routine for INDCB
IOERROR  SYNADAF ACSMETH=QSAM           Get message area
         MVI    6(R1),X'80'             Set WTO MCS flags
         MVC    8(16,R1),=CL16'I/O Error' Put phrase on binary fields
         MVC    128(4,R1),=X'00000020'   Set ROUTCDE=11 (WTP)
         WTO    MF=(E,4(R1))            Write message to user
         SYNADRLS                       Release SYNADAF area, fall through
* The GET macro branches here after all records have been read
EOD      CLOSE  (INDCB)                 Close the data set
         FREEPOOL INDCB                 Free the QSAM buffer pool
         ...                            Rest of program
INDCB    DCB    DDNAME=INPUT,MACRF=GL,RECFM=VB,    Must be format-V           *
                DCBE=INDCBE
INDCBE   DCBE   EODAD=EOD,SYNAD=IOERROR,BLKSIZE=0  Request LBI
```

Figure 52. Reading a Sequential Data Set

Figure 53 is the same as Figure 52 on page 316 but converted to be reentrant and reside above the 16 MB line:

```
         COPYPROG CSECT
         COPYPROG RMODE ANY
         COPYPROG AMODE 31
         GETMAIN R,LV=Arealen,LOC=(BELOW,64)
         LR      R11,R1
         USING   MYAREA,R11
         USING   IHADCB,InDCB
         USING   DCBE,INDCBE
         MVC     IHADCB(AreaLen),MYDCB    Copy DCB and DCBE
         LA      R0,DCBE                  Point DCB copy to
         ST      R0,DCBDCBE               DCBE copy
         OPEN    (IHADCB,),MF=(E,INOPEN)  Open to read
         LTR     R15,R15                  Branch if DDname seems not
         BNZ     ...                        to be defined
* Loop to read all the records
LOOP     GET     INDCB                    Get address of a record in R1
         ...                              Process a record
         B       LOOP                     Branch to read next record
* I/O error routine for INDCB
IOERROR  SYNADAF ACSMETH=QSAM             Get message area
         MVI    6(R1),X'80'               Set WTO MCS flags
         MVC    8(16,R1),=CL16'I/O Error'  Put phrase on binary fields
         MVC    128(4,R1),=X'00000020'    Set ROUTCDE=11 (WTP)
         WTO    MF=(E,4(R1))              Write message to user
         SYNADRLS                         Release SYNADAF area, fall through
* The GET macro branches here after all records have been read
EOD      CLOSE  MF=(E,INOPEN)           Close the data set
* FREEPOOL not needed due to RMODE31=BUFF
         ...                            Rest of program
MYDCB    DCB    DDNAME=INPUT,MACRF=GL,RECFM=VB,                              *
                DCBE=MYDCBE
MYDCBE   DCBE   EODAD=EOD,SYNAD=IOERROR,BLKSIZE=0,RMODE31=BUFF
         OPEN   (,INPUT),MF=L,MODE=24
AreaLen  EQU    *-MYDCB
         DCBD   DSORG=QS,DEVD=DA
         IHADCBE                        Could be above 16 MB line
MYAREA   DSECT
INDCB    DS     XL(DCBLNGQS)
INDCBE   DS     XL(DCBEEND-DCBE)
INOPEN   OPEN   (,),MF=L
```

Figure 53. Reentrant—Above the 16 MB Line

Using OPEN to Prepare a Data Set for Processing

Use the OPEN macro to complete a DCB for a data set, and to supply the specifications needed for I/O operations. Therefore, the appropriate data can be provided when your job is executed rather than when you write your program (see Figure 54 on page 319).

When the OPEN macro is executed, the OPEN routine:
- Completes the DCB
- Stores appropriate access method routine addresses in the DCB
- Initializes data sets by reading or writing labels and control information
- Builds the necessary system control blocks

The operating system stores information from a DD statement or dynamic allocation in the job file control block (JFCB). The open function uses the JFCB.

The DCB is filled in with information from the DCB macro, the JFCB, or an existing data set label. If more than one source specifies information for a particular field, only one source is used. A DD statement takes priority over a data set label, and a DCB macro over both.

You can change most DCB fields either before the data set is opened or when the operating system returns control to your program (at the DCB OPEN user exit). Some fields can be changed during processing. Do not try to change a DCB field, such as data set organization, from one that permitted the data set to be allocated to a system-managed volume, to one that makes the data set ineligible to be system-managed. For example, do not specify a data set organization in the DD statement as physical sequential and, after the data set has been allocated to a system-managed volume, try to open the data set with a DCB that specifies the data set as physical sequential unmovable. The types of data sets that cannot be system-managed are listed in “Chapter 2. Using the Storage Management Subsystem” on page 27.

Filling in the DCB

Figure 54 shows the process and the sequence of filling in the DCB from various sources.

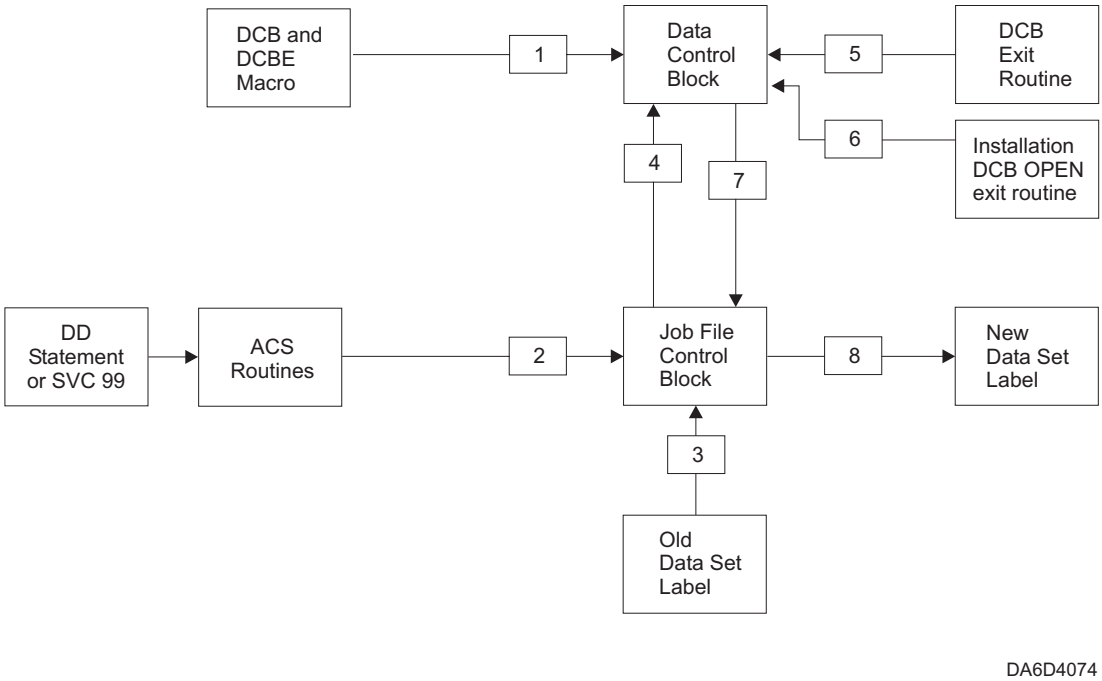


Figure 54. Sources and Sequence of Operations for Completing the DCB

The following items correspond to the boxed numbers in Figure 54.

1. The primary source is your program; that is, the DCB and DCBE macro or compiler. Usually, you should use only those DCB and DCBE parameters that are needed to ensure correct processing. The other parameters can be filled in when your program is to be executed.
2. A JCL DD statement or a call to dynamic allocation (SVC 99) creates a job file control block (JFCB). The DD or SVC 99 can supply a data class (DATACLAS) name. The installation data class ACS routine can supply or override the data class name. The resulting data class provides defaults for certain parameters that were omitted from the DD or SVC 99. Parameters from a data class do not override a DD or SVC 99. If the DD statement or call to dynamic allocation has a BLKSIZE value that exceeds 32 760, that value is in a system control block that is a logical extension to the JFCB, and the JFCB BLKSIZE field has a zero value.
3. When a DASD data set is opened (or a magnetic tape with standard labels is opened for INPUT, RDBACK, or INOUT or is being extended), any field in the JFCB not completed by a DD statement or data class is filled in from the data set label (if one exists). When you open a new DASD data set, the system might previously have calculated an optimal block size and stored it in the data set label. It does that if RECFM, LRECL, and DSORG are available. When opening a magnetic tape for output, the tape label is assumed not to exist or to apply to the current data set unless you specify DISP=MOD, or open for EXTEND or OUTINX and a volume serial number in the volume parameter of the DD statement (or the data set is cataloged).

OPEN does not perform a merge from a data set label to JFCB for a “like” sequential concatenation when making the transition between data sets. If you want a merge, turn on the unlike attribute bit (DCBOFPPC) in the DCB. The unlike attribute forces the system through OPEN for each data set in the concatenation, where a label to JFCB merge takes place. See “Concatenating Unlike Data Sets” on page 384.

4. From the JFCB, OPEN fills in any field not completed in the DCB or DCBE. This completes what is called the forward merge.
5. Certain fields in the DCB or DCBE can then be completed or changed by your own DCB user exit routine or JFCBE exit routine. The DCB and DCBE macro fields are described in *z/OS DFSMS Macro Instructions for Data Sets*. These exits are described in “DCB OPEN Exit” on page 509 and “JFCBE Exit” on page 513.
6. After OPEN calls the user’s optional DCB OPEN exit or JFCBE exit, it calls the installation’s optional OPEN exit routine. Either type of exit or both can make certain changes to the DCB and DCBE.

   The block size field (BLKSIZE) is in two bytes in the DCB if you are not using large block interface (LBI). Its maximum value is 32 760. The block size field is in four bytes in the DCBE if you are using LBI. After possibly calling these exits, OPEN tests if the block size field is zero or an exit changed LRECL or RECFM after the system calculated a block size when the DASD data set space was allocated. In either case, OPEN calculates an optimal block size according to the device type if the RECFM is not U.
7. All DCB fields are then unconditionally merged into corresponding JFCB fields if your data set is opened for output. This is the beginning of what is called the reverse merge. Merging the DCB fields is caused by specifying OUTPUT, OUTIN, EXTEND, or OUTINX in the OPEN macro.

   The DSORG field is merged only when it contains zeros in the JFCB. If your data set is opened for input (INPUT, INOUT, RDBACK, or UPDAT is specified in the OPEN macro), the DCB fields are not merged unless the corresponding JFCB fields contain zeros.
8. The open routines use the updated JFCB and associated control blocks to write the DASD data set labels if the data set was open for OUTPUT, OUTIN, OUTINX, or EXTEND. For standard labeled tapes, the open routines write labels only for the OUTPUT or OUTIN options when you are not extending the data set. You are extending if the OPEN option is OUTPUT or OUTIN with DISP=MOD or the OPEN option is OUTINX, EXTEND or INOUT. When extending a standard labeled tape data set, the EOV and CLOSE functions use the updated JFCB and associated control blocks to write trailer labels. If the data set is not closed when your program ends, the operating system closes it automatically.

When the data set is closed, the DCB is restored to the condition it had before the data set was opened (except that the buffer pool is not freed) unless you coded RMODE31=BUFF and OPEN accepted it.

Specifying the Forms of Macros, Buffering Requirements, and Addresses

The operating system requires several types of processing information to ensure proper control of your I/O operations. You must specify the forms of macros in the program, buffering requirements, and the addresses of your special processing routines during either the assembly or the execution of your program. The DCB parameters specifying buffer requirements are discussed in “Managing SAM Buffer Space” on page 341.

Because macros are expanded during the assembly of your program, you must supply the macro forms to be used in processing each data set in the associated DCB macro. You can supply buffering requirements and related information in the DCB and DCBE macro, the DD statement, or by storing the applicable data in the appropriate field of the DCB or DCBE before the end of your DCB exit routitne. If the addresses of special processing routines (EODAD, SYNAD, or user exits) are omitted from the DCB and DCBE macro, you must complete them in the DCB or DCBE before they are required.

Coding Processing Methods

You can process a data set as input, output, or update by coding the processing method in the OPEN macro. If the processing method parameter is omitted from the OPEN macro, INPUT is assumed. BISAM and QISAM scan mode ignore all the OPEN processing options. Use OUTPUT or EXTEND when using QISAM load mode with an indexed sequential data set.

INPUT—BDAM, BPAM, BSAM, QSAM
OUTPUT—BDAM, BPAM, BSAM, QSAM, QISAM (load mode)
EXTEND—BDAM, BPAM (PDSE only), BSAM, QSAM, QISAM (load mode)
UPDAT—BDAM, BPAM, BSAM, QSAM
RDBACK—BSAM, QSAM
INOUT—BSAM
OUTIN—BSAM
OUTINX—BSAM

If the data set resides on a direct access volume, you can code UPDAT in the processing method parameter to show that records can be updated.

RDBACK is supported only for magnetic tape. By coding RDBACK, you can specify that a magnetic tape volume containing format-F or format-U records is to be read backward. (Variable-length records cannot be read backward.)

Restriction: When a tape that is recorded in Improved Data Recording Capability (IDRC) mode, is read backward, it will have a severe performance degradation.

You can override the INOUT, OUTIN, UPDAT, or OUTINX at execution time by using the IN or OUT options of the LABEL parameter of the DD statement, as discussed in *z/OS MVS JCL Reference*. The IN option indicates that a BSAM data set opened for INOUT or a direct data set opened for UPDAT is to be read only. The OUT option indicates that a BSAM data set opened for OUTIN or OUTINX is to be written in only.

Restriction: Unless allowed by the label validation installation exit, OPEN for OUTPUT or OUTIN with DISP=MOD, INOUT, EXTEND, or OUTINX requests cannot be processed for ISO/ANSI Version 3 tapes or for non-IBM-formatted Version 4 tapes, because this kind of processing updates only the closing label of the file, causing a label symmetry conflict. An unmatching label should not frame the other end of the file. This restriction does not apply to IBM-formatted ISO/ANSI Version 4 tapes.

Related reading: For information about the label validation installation exit, see *z/OS DFSMS Installation Exits*.

Processing SYSIN, SYSOUT, and subsystem data sets. INOUT is treated as INPUT. OUTIN, EXTEND, or OUTINX is treated as OUTPUT. UPDAT and RDBACK cannot be used. SYSIN and SYSOUT data sets must be opened for INPUT and OUTPUT, respectively.

Processing PDSEs. For PDSEs, INOUT is treated as INPUT. OUTIN, EXTEND, and OUTINX are treated as OUTPUT.

Processing compressed-format data sets. Compressed-format data sets must not be opened for UPDAT.

In Figure 55 the data sets associated with three DCBs are to be opened simultaneously.

```
OPEN      (TEXTDCB,,CONVDCB,(OUTPUT),PRINTDCB,                    X
          (OUTPUT))
```

Figure 55. Opening Three Data Sets at the Same Time

Because no processing method parameter is specified for TEXTDCB, the system assumes INPUT. Both CONVDCB and PRINTDCB are opened for output. No volume positioning options are specified; thus, the disposition indicated by the DD statement DISP parameter is used.

Selecting Data Set Options

The following sections contain information about selecting data set options.

DCB and DCBE Parameters

After you have specified the data set characteristics in the DCB and DCBE macro, you can change them only by changing the DCB or DCBE during execution. See “Changing and Testing the DCB and DCBE” on page 330. The fields of the DCB discussed in the following sections are common to most data organizations and access methods. The DCBE is for BSAM, BPAM, and QSAM. For more information about the DCB and DCBE fields see *z/OS DFSMS Macro Instructions for Data Sets*.

Block Size (BLKSIZE)

BLKSIZE specifies the maximum length, in bytes, of a data block. If the records are format-F, the block size must be an integral multiple of the record length, except for SYSOUT data sets. (See “Chapter 25. Spooling and Scheduling Data Sets” on page 373.) If the records are format-V, the block size specified must be the maximum block size. If format-V records are unblocked, the block size must be 4 bytes greater than the record length (LRECL). If you do not use the large block

interface (LBI), the maximum block size is 32 760 except for ISO/ANSI Version 3 records, where the maximum block size is 2048. The 2048 byte limit can be overridden by a label validation installation exit (see *z/OS DFSMS Installation Exits*). If you use LBI, the maximum block size is 32 760 except on magnetic tape, where the maximum is larger.

The system can derive the best block size for DASD and tape and spooled data sets. The system does not derive a block size for BDAM, old, or unmovable data sets, or when the RECFM is U. See "System-Determined Block Size" on page 324 for more information on system-determined block sizes for DASD and tape data sets.

If you specify a block size other than zero, there is no minimum requirement for block size except that format-V blocks have a minimum block size of 8. However, if a data check occurs on a magnetic tape device, any block shorter than 12 bytes in a read operation, or 18 bytes in a write operation, is treated as a noise record and lost. No check for noise is made unless a data check occurs.

Large Block Interface (LBI): The large block interface (LBI) lets your program handle much larger blocks with BSAM or QSAM. On the current level of the system you can use LBI with BSAM, BPAM, and QSAM for any kind of data set except unit record or a TSO/E terminal. Currently blocks of more than 32 760 are supported only on tape.

You request LBI by coding a BLKSIZE value, even 0, in the DCBE macro or by turning on the DCBEULBI bit before completion of the DCB OPEN exit. Coding BLKSIZE causes the bit to be on. It is best if this bit is on before you issue the OPEN macro. That lets OPEN merge a large block size into the DCBE.

Your DCB OPEN exit can test bit DCBESLBI to learn if the access method supports LBI. If the bit is off, it might mean you are running on DFSMS/MVS 1.5 or earlier, which does not support LBI. If your program did not request *unlike* attributes processing (by turning on bit DCBOFPPC) before issuing OPEN, then DCBESLBI being on means that all the data sets in the concatenation support LBI. If your program requested *unlike* attributes processing before OPEN, then DCBESLBI being on each time that the system calls your DCB OPEN exit or JFCBE exit means only that the next data set supports LBI. After the exit, OPEN leaves DCBESLBI on only if DCBEULBI also is on. Your exit routine can change DCBEULBI. Never change DCBESLBI.

Another way to learn if the data set type supports LBI is to issue a DEVTYPE macro with INFO=AMCAP. See *z/OS DFSMSdfp Advanced Services*. After the DCB OPEN exit, the following items apply when DCBESLBI is on:

- OPEN is honoring your request for LBI.
- Do not use the BLKSIZE field in the DCB. The system uses it. Use the BLKSIZE field in the DCBE. For more information about DCBE field descriptions see *z/OS DFSMS Macro Instructions for Data Sets*.
- You can use extended BDWs with format-V records. Format-V blocks longer than 32 760 bytes require an extended BDW. See "Block Descriptor Word (BDW)" on page 293.
- When reading with BSAM or BPAM, your program determines the length of the block differently. See "Determining the Length of a Block when Reading with BSAM, BPAM, or BDAM" on page 393.

- When writing with BSAM or BPAM, your program sets the length of each block differently. See “Writing a Short Format-FB Block with BSAM or BPAM” on page 394.
- When writing undefined-length records with QSAM, you store the record length in the DCBE before issuing each PUT. See *z/OS DFSMS Macro Instructions for Data Sets*.
- After an I/O error, register 0 and the status area in the SYNAD routine are slightly different, and the beginning of the area returned by the SYNADAF macro is different. See Figure 107 on page 492 and *z/OS DFSMS Macro Instructions for Data Sets*.
- If the block size exceeds 32 760, you cannot use the BUILD, GETPOOL, or BUILDRCD macro or the BUFL parameter.
- Your program cannot request exchange buffering (BFTEK=E), OPTCD=H (VSE embedded checkpoints) or open with the UPDAT option.
- With LBI, fixed-length unblocked records greater than 32 760 bytes are not supported by QSAM.

System-Determined Block Size: If you do not specify a block size for the creation of a data set, the system attempts to determine the block size. Using a system-determined block size has the following benefits:

- The program can write to DASD, tape, or SYSOUT without you or the program calculating the optimal block size. DASD track capacity calculations are complicated. Optimal block sizes differ for various models of DASD and tape.
- If the data set later is moved to a different DASD type, such as by DFSMShsm, the system recalculates an appropriate block size and reblocks the data.

The system determines the block size for a data set as follows:

1. OPEN calculates a block size.
2. OPEN compares the calculated block size to a block size limit, which affects only data sets on tape because the minimum value of the limit is 32 760.
3. OPEN attempts to decrease the calculated block size to be less than or equal to the limit.

The block size limit is the first nonzero value from the following items:

1. BLKSZLIM value in the DD statement or dynamic allocation.
2. Block size limit in the data class. The SMS data class ACS routine can assign a data class to the data set. You can request a data class name with the DATACLAS keyword in the DD statement or the dynamic-allocation equivalent. The data set does not have to be SMS managed.
3. TAPEBLKSZLIM value in the DEVSUPxx member of SYS1.PARMLIB. A system programmer sets this value, which is in the data facilities area (DFA) (see *z/OS DFSMSdfp Advanced Services*).
4. The minimum block-size limit, 32 760.

Your program can obtain the BLKSZLIM value that is in effect by issuing the RDJFCB macro with the X'13' code (see *z/OS DFSMSdfp Advanced Services*).

Because larger blocks generally cause data transfer to be faster, why would you want to limit it? Some possible reasons follow:

- A user will take the tape to an operating system or older z/OS system or application program that does not support the large size that you want. The other operating system might be a backup system that is used only for disaster recovery. An OS/390 system before Version 2 Release 10 does not support the large block interface that is needed for blocks longer than 32 760.
- You want to copy the tape to a different type of tape or to DASD without reblocking it, and the maximum block size for the destination is less than you want. An example is the IBM 3480 Magnetic Tape Subsystem, whose maximum block size is 65 535. The optimal block size for an IBM 3590 is 224 KB or 256 KB, depending on the level of the hardware. To copy from an optimized 3590 to a 3480 or 3490, you must reblock the data.
- A program that reads or writes the data set and runs in 24-bit addressing mode might not have enough buffer space for very large blocks.

Table 27 describes block size support.

Table 27. Optimum and Maximum Block Size Supported

Device Type	Optimum	Maximum
DASD	Half track (usually)	32 760
Reel tape	32 760	32 760
3480, 3490	65 535	65 535
3590	262 144 (256 KB) except on some older models on which it is 229 376 (224 KB)	262 144 (256 KB)
DUMMY	16	5 000 000

DASD Data Sets: When you create (allocate space for) a new DASD data set, the system derives the optimum block size and saves it in the data set label if all of the following are true:

- Block size is not available or specified from any source. BLKSIZE=0 can be specified.
- You specify LRECL or it is in the data class. The data set does not have to be SMS managed.
- You specify RECFM or it is in the data class. It must be fixed or variable.
- You specify DSORG as PS or PO or you omit DSORG and it is PS or PO in the data class.

Your DCB OPEN exit can examine the calculated block size in the DCB or DCBE if no source other than the system supplied the block size.

Data Control Block (DCB)

When a program opens a DASD data set for writing the first time since it was created, OPEN derives the optimum block size after calling the optional DCB OPEN exit and all the following are true:

- The block size in the DCB (or DCBE with LBI) is zero.
- The system determined the block size when the data set was created, and RECFM or LRECL in the DCB is different from the data set label.
- LRECL is in the DCB.
- RECFM is in the DCB and it is fixed or variable.
- The access method is BSAM, BPAM, or QSAM.

For sequential or PDSs, the system-determined block size returned is optimal in terms of DASD space utilization. For PDSE's, the system-determined block size is optimal in terms of I/O buffer size because PDSE physical block size on the DASD is a fixed size determined by PDSE.

For a compressed format data set, the system does not consider track length. The access method simulates blocks whose length is independent of the real physical block size. The system-determined block size is optimal in terms of I/O buffer size. The system chooses a value for the BLKSIZE parameter as it would for an IBM standard labeled tape as in Table 28 on page 327 and always limits it to 32 760. This value is stored in the DCB or DCBE and DS1BLKL in the DSCB. However, regardless of the block size found in the DCB and DSCB, the actual size of the physical blocks written to DASD is calculated by the system to be optimal for the device.

The system does not determine the block size for the following types of data sets:

- Unmovable data sets
- Data sets with a record format of U
- Old data sets
- Direct data sets

Unmovable data sets cannot be system managed. There are exceptions, however, in cases where the checkpoint/restart function has set the unmovable attribute for data sets that are already system managed. This setting prevents data sets opened previously by a checkpointed application from being moved until you no longer want to perform a restart on that application.

Tape Data Sets: The system can determine the optimum block size for tape data sets. The system sets the block size at OPEN on return from the DCB OPEN exit and installation DCB OPEN exit if:

- The block size in DCBBLKSI is zero (or DCBEBLKSI if using LBI).
- The record length is not zero.
- The record format is fixed or variable.
- The tape data set is open for OUTPUT or OUTIN.
- The access method is BSAM or QSAM.

Rule: For programming languages, the program must specify the file is blocked in order to get tape system-determined block size. For example, with COBOL, the program should specify BLOCK CONTAINS 0 RECORDS.

The system-determined block size depends on the record format of the tape data set. Table 28 shows the block sizes that are set for tape data sets.

Table 28. Rules for Setting Block Sizes for Tape Data Sets or Compressed Format Data Sets

RECFM	Block Size Set
F or FS	LRECL
FB or FBS (Label type=AL Version 3)	Highest possible multiple of LRECL that is ≤ 2048 if LRECL ≤ 2048 Highest possible multiple of LRECL that is ≤ 32 760 if LRECL > 2048
FB or FBS (Label type=AL Version 4 or not AL)	Not tape or not LBI: highest possible multiple of LRECL that is ≤ 32 760 LBI on tape: Highest possible multiple of LRECL that is ≤ the device's optimal block size
V (not AL)	LRECL + 4 (LRECL must be less than or equal to 32 756)
VS (not AL)	LRECL + 4 if LRECL ≤ 32 756 32 760 if LRECL > 32 756
VB or VBS (not AL)	Not tape or not LBI: 32 760 LBI on tape: Device's optimal block size
D (Label type=AL)	LRECL + 4 (LRECL must be ≤ 32 756)
DBS or DS (Label type=AL Version 3)	2048 (the maximum block size allowed unless an installation exit allows it)
D or DS (Label type NL or NSL or label type=AL Version 4)	LRECL + 4 (LRECL must be ≤ 32 756)
DB or DBS (Label type NL, or NSL, or AL Version 4)	32 760
DB not spanned (Label type=AL Version 3)	2048 if LRECL ≤ 2044 DCBBLKSI = 32 760 if LRECL > 2044 (you have the option, for AL Version 3, to accept this block size in the label validation installation exit)

Label Types:
AL = ISO/ANSI labels
NL = no labels
NSL = nonstandard labels
SL = IBM standard labels
Not AL = NL, NSL, or SL labels

RECFM Allowances:
- RECFM=D is not allowed for SL tapes
- RECFM=V is not allowed for AL tapes

Data Set Organization (DSORG)

DSORG specifies the organization of the data set as physical sequential (PS), indexed sequential (IS), partitioned (PO), or direct (DA). If the data set is processed using absolute rather than relative addresses, you must mark it as unmovable by adding a U to the DSORG parameter (for example, by coding DSORG=PSU). You must specify the data set organization in the DCB macro. In addition:

- When creating or processing an indexed sequential organization data set or creating a direct data set, you must also specify DSORG in the DD statement.
- When creating a direct data set, the DSORG in the DCB macro must specify PS or PSU and the DD statement must specify DA or DAU.
- PS is for sequential and extended format DSNTYPE.
- PO is the data set organization for both PDSEs and PDSs. DSNTYPE is used to distinguish between PDSEs and PDSs.

Unmovable and IS data sets cannot be system managed.

Key Length (KEYLEN)

KEYLEN specifies the length (0 to 255) in bytes of an optional key that precedes each block on direct access storage devices. The value of KEYLEN is not included in BLKSIZE or LRECL, but must be included in BUFL if buffer length is specified. Thus, BUFL=KEYLEN+BLKSIZE. See “Using KEYLEN with PDSEs” on page 435 for information about using the KEYLEN parameter with PDSEs.

Rule: Do not specify nonzero key length when opening a PDSE or extended format data set for output.

IBM recommends not coding KEYLEN or coding KEYLEN=0. A nonzero value generally will make your program run slower.

Record Length (LRECL)

LRECL specifies the length, in bytes, of each record in the data set. If the records are of variable length or undefined length, the maximum record length must be specified. For input, the field has no effect for undefined-length (format-U) records. The value of LRECL and when you specify it depends on the format of the records:

- For fixed-length unblocked records, LRECL must equal BLKSIZE.
- For PDSEs or compressed-format data sets with fixed-length blocked records, LRECL must be specified when the data set is opened for output.
- For the extended logical record interface (XLRI) for ISO/ANSI variable spanned records, LRECL must be specified as LRECL=0K or LRECL=nK.

Record Format (RECFM)

RECFM specifies the characteristics of the records in the data set as fixed-length (F), variable-length (V), ASCII variable-length (D), or undefined-length (U). Blocked records are specified as FB, VB, or DB. Spanned records are specified as VS, VBS, DS, or DBS. You can also specify the records as fixed-length standard by using FS or FBS. You can request track overflow for records other than standard format by adding a T to the RECFM parameter (for example, by coding FBT). Track overflow is ignored for PDSEs.

The type of print control can be specified to be in ANSI format-A, or in machine code format-M. See “Using Optional Control Characters” on page 308 and *z/OS DFSMS Macro Instructions for Data Sets* for information about control characters.

Write Validity Check Option (OPTCD=W)

You can specify the write validity check option in the DCB parameter of the DD statement, the dynamic allocation text units, or the DCB macro. After a block is transferred from main to auxiliary storage, the system reads the stored block (without data transfer) and, by testing for a data check from the I/O device, verifies that the block was written correctly. Be aware that the write validity check process requires an additional revolution of the device for each block. If the system detects any errors, it starts its standard error recovery procedure.

For buffered tape devices, the write validity check option delays the device end interrupt until the data is physically on tape. When you use the write validity check option, you get none of the performance benefits of buffering and the average data transfer rate is much less.

Rule: OPTCD=W is ignored for PDSEs and for extended format data sets.

DD Statement Parameters

Each of the data set description fields of the DCB, except for direct data sets, can be specified when your job is to be executed. Also, data set identification and disposition, and device characteristics, can be specified at that time. To allocate a data set, you must specify the data set name and disposition in the DD statement. In the DD statement, you can specify a data class, storage class, and management class, and other JCL keywords.You can specify the classes using the JCL keywords DATACLAS, STORCLAS, and MGMTCLAS. If you do not specify a data class, storage class, or management class, the ACS routines assign classes based on the defaults defined by your storage administrator. Storage class and management class can be assigned only to data sets that are to be system managed.

ACS Routines. Your storage administrator uses the ACS routines to determine which data sets are to be system managed. The valid classes that can either be specified in your DD statement or assigned by the ACS routines are defined in the SMS configuration by your storage administrator. The ACS routines analyze your JCL, and if you specify a class that you are not authorized to use or a class that does not exist, your allocation fails. For more information about specifying data class, storage class, and management class in your DD statement see *z/OS MVS JCL User's Guide*.

Data Class. Data class can be specified for both system-managed and non-system-managed data sets. It can be specified for both DASD and tape data sets. You can use data class together with the JCL keyword LIKE for tape data sets. This simplifies migration to and from system-managed storage. When you allocate a data set, the ACS routines assign a data class to the data set, either the data class you specify in your DD statement, or the data class defined as the default by your storage administrator. The data set is allocated using the information contained in the assigned data class. See your storage administrator for information on the data classes available to your installation and *z/OS DFSMSdfp Storage Administration Reference* for more information about allocating system-managed data sets and using SMS classes.

Data Control Block (DCB)

You can override any of the information contained in a data class by specifying the values you want in your DD statement or dynamic allocation. A data class can contain any of the following information.

Data Set Characteristics	JCL Keywords Used To Override
Data set organization	DSORG
Data set type	DSNTYPE
Key length	KEYLEN
Key offset	KEYOFF
Record format	RECFM
Record length	LRECL
Block size	BLKSIZE
Block size limit	BLKSZLIM
Record organization	RECORG
Retention period	RETPD
Space allocation	SPACE, AVGREC

Related reading: For more information on the JCL keywords that override data class information, see *z/OS MVS JCL User's Guide* and *z/OS MVS JCL Reference*.

In a DD statement or dynamic-allocation call, you cannot specify directly through the DSNTYPE value that the data set is to be an extended-format data set.

The easiest data set allocation is one that uses the data class, storage class, and management class defaults defined by your storage administrator. The following example shows how to allocate a system-managed data set:

```
//ddname DD DSNAME=NEW.PLI,DISP=(NEW,KEEP)
```

You cannot specify the keyword DSNTYPE with the keyword RECORG in the JCL DD statement. They are mutually exclusive.

Changing and Testing the DCB and DCBE

With certain restrictions you can complete or change the DCB or DCBE during execution of your program. You can also determine data set characteristics from information supplied by the data set labels. You can make changes or additions before you open a data set, after you close it, during the DCB OPEN exit routine, or while the data set is open. See "DCB OPEN Exit" on page 509 and "Filling in the DCB" on page 319 for information about using the DCB OPEN exit routines. Also see *z/OS DFSMS Macro Instructions for Data Sets* for information about changing various DCB fields. (Naturally, you must supply the information before it is needed.)

You should not attempt to change the data set characteristics of a system-managed data set to characteristics that make it ineligible to be system managed. For example, do not specify a data set organization in the DD statement as PS and, after the data set has been allocated to a system-managed volume, change the DCB to specify DSORG=PSU. That causes abnormal end of your program.

Using the DCBD Macro

Use the data control block DSECT (DCBD) macro to identify the DCB field names symbolically. If you load a base register with a DCB address, you can refer to any field symbolically. You can code the DCBD macro once to describe all DCBs.

The DCBD macro generates a dummy control section (DSECT) named IHADCB. Each field name symbol consists of DCB followed by the first 5 letters of the keyword subparameter for the DCB macro. For example, the symbolic name of the block size parameter field is DCBBLKSI. (For other DCB field names see *z/OS DFSMS Macro Instructions for Data Sets*.)

The attributes of each DCB field are defined in the dummy control section. Use the DCB macro's assembler listing to determine the length attribute and the alignment of each DCB field.

Changing an Address in the DCB

Figure 56 shows how to change a field in the DCB.

```
            ...
            OPEN     (TEXTDCB,INOUT),MODE=31
            ...
EOFEXIT     CLOSE    (TEXTDCB,REREAD),MODE=31,TYPE=T
            LA       10,TEXTDCB
            USING    IHADCB,10
            MVC      DCBSYNAD+1(3),=AL3(OUTERROR)
            B        OUTPUT
INERROR     STM      14,12,SYNADSA+12
            ...
OUTERROR    STM      14,12,SYNADSA+12
            ...
TEXTDCB     DCB      DSORG=PS,MACRF=(R,W),DDNAME=TEXTTAPE,   C
                     EODAD=EOFEXIT,SYNAD=INERROR
            DCBD     DSORG=PS
            ...
```

Figure 56. Changing a Field in the DCB

The data set defined by the data control block TEXTDCB is opened for both input and output. When the application program no longer needs it for input, the EODAD routine closes the data set temporarily to reposition the volume for output. The EODAD routine then uses the dummy control section IHADCB to change the error exit address (SYNAD) from INERROR to OUTERROR.

The EODAD routine loads the address TEXTDCB into register 10, the base register for IHADCB. Then it moves the address OUTERROR into the DCBSYNAD field of the DCB. Even though DCBSYNAD is a fullword field and contains important information in the high-order byte,change only the 3 low-order bytes in the field.

All unused address fields in the DCB, except DCBEXLST, are set to 1 when the DCB macro is expanded. Many system routines interpret a value of 1 in an address field as meaning no address was specified, so use it to dynamically reset any field you do not need.

Using the IHADCBE Macro

Use the IHADCBE mapping macro to identify DCB extension field names symbolically. If you load a base register with a DCBE address, you can refer to any field symbolically. You can code the IHADCBE macro once to describe all DCBEs.

The IHADCBE macro generates a dummy control section (DSECT) named DCBE. For the symbols generated see *z/OS DFSMS Macro Instructions for Data Sets*.

All address fields in the DCBE are 4 bytes. All undefined addresses are set to 0.

Using CLOSE to End the Processing of a Data Set

The CLOSE macro is used to end processing of a data set and release it from a DCB. The volume positioning (tapes only) that is to result from closing the data set can also be specified. See “Positioning Volumes” on page 338 for the definition of volume positioning. Volume positioning options are the same as those that can be specified for end-of-volume conditions in the OPEN macro or the DD statement. An additional volume positioning option, REWIND, is available and can be specified by the CLOSE macro for magnetic tape volumes. REWIND positions the tape at the load point regardless of the direction of processing.

Issuing the CHECK Macro

Before issuing the CLOSE macro, a CHECK macro must be issued for all DECBs that have outstanding I/O from WRITE macros. When CLOSE TYPE=T is specified, a CHECK macro must be issued for all DECBs that have outstanding I/O from either WRITE or READ macros except when issued from EODAD.

In Figure 57 the data sets associated with three DCBs are to be closed simultaneously. Because no volume positioning parameters (LEAVE, REWIND) are specified, the positioning indicated by the DD statement DISP parameter is used.

```
CLOSE      (TEXTDCB,,CONVDCB,,PRINTDCB)
```

Figure 57. Closing Three Data Sets at the Same Time

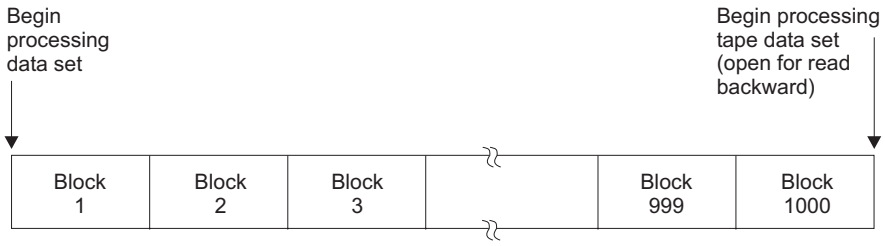
Closing a Data Set Temporarily

You can code CLOSE TYPE=T to temporarily close sequential data sets on magnetic tape and direct access volumes processed with BSAM. When you use TYPE=T, the DCB used to process the data set maintains its open status. You do not have to issue another OPEN macro to continue processing the same data set. CLOSE TYPE=T cannot be used in a SYNAD routine.

The TYPE=T parameter causes the system control program to process labels, modify some of the fields in the system control blocks for that data set, and reposition the volume (or current volume for multivolume data sets) in much the same way that the normal CLOSE macro does. When you code TYPE=T, you can specify that the volume is either to be positioned at the end of data (the LEAVE option) or to be repositioned at the beginning of data (the REREAD option). Magnetic tape volumes are repositioned either immediately before the first data record or immediately after the last data record. The presence of tape labels has no effect on repositioning.

When a DCB is shared among multiple tasks, only the task that opened the data set can close it unless TYPE=T is specified.

Figure 58, which assumes a sample data set containing 1000 blocks, shows the relationship between each positioning option and the point where you resume processing the data set after issuing the temporary close.



If you CLOSE CODE TYPE = T and specify	After temporary close, you will resume processing
LEAVE	Immediately after block 1000
LEAVE (with tape data set open for read backward)	Immediately before block 1
REREAD	Immediately before block 1
REREAD (with tape data set open for read backward)	Immediately after block 1000

Figure 58. Record Processed when LEAVE or REREAD is Specified for CLOSE TYPE=T

Using CLOSE TYPE=T with Sequential Data Sets

For data sets processed with BSAM, you can use CLOSE TYPE=T with the following restrictions:

- The DCB for the data set you are processing on a direct access device must specify either DSORG=PS or DSORG=PSU for input processing, and either DSORG=PS, DSORG=PSU, DSORG=PO, or DSORG=POU for output processing. (You cannot specify the REREAD option if DSORG=PO or DSORG=POU is specified. The REREAD restriction prohibits the use of temporary close following or during the building of a BDAM data set that is allocated by specifying BSAM MACRF=WL.)
- The DCB must not be open input to a member of a PDS.
- If you open the data set for input and issue CLOSE TYPE=T with the LEAVE option, the volume will be repositioned only if the data set specifies DSORG=PS or DSORG=PO.

Releasing Space

The close function attempts to release unused tracks or cylinders for a data set if all of the following are true:

- The SMS management class specifies YI or CI for the partial release attribute, or you specified RLSE for the SPACE parameter in the DD statement or RELEASE in the TSO ALLOCATE command.
- You did not specify TYPE=T on the CLOSE macro.
- The DCB was opened with the OUTPUT, OUTIN, OUTINX, INOUT or EXTEND option and the last operation before CLOSE was WRITE (and CHECK), STOW or PUT.
- No other DCB for this data set in the address space was open.
- No other address space in any system is allocated to the data set.
- The data set is sequential or partitioned.
- Certain functions of dynamic allocation are not currently executing in the address space.

For a multivolume data set that is not in extended format, CLOSE releases space only on the current volume.

Space also can be released when DFSMShsm is performing space management or when an authorized program issues the PARTREL macro.

Space is released on a track boundary if the extent containing the last record was allocated in units of tracks or in units of average record or block lengths with ROUND not specified. Space is released on a cylinder boundary if the extent containing the last record was allocated in units of cylinders or in units of average block lengths with ROUND specified. However, a cylinder boundary extent could be released on a track boundary if:

- The DD statement used to access the data set contains a space parameter specifying units of tracks or units of average block lengths with ROUND not specified, or
- No space parameter is supplied in the DD statement and no secondary space value has been saved in the data set label for the data set.

Changing a cylinder boundary extent to a track boundary extent generally causes loss of the possible performance benefit of a cylinder boundary. On the latest disk drives there is no performance benefit of cylinder boundaries.

Managing Buffer Pools When Closing Data Sets

After closing the data set, you should issue a FREEPOOL macro to release the virtual storage used for the buffer pool unless you specified RMODE31=BUFF on the DCBE macro with QSAM or it is BSAM and BUFNO was not supplied from any source. If you plan to process other data sets, use FREEPOOL to regain the buffer pool storage space. If you expect to reopen a data set using the same DCB, use FREEPOOL unless the buffer pool created the first time the data set was opened will meet your needs when you reopen the data set. FREEPOOL is discussed in more detail in “Constructing a Buffer Pool” on page 342.

After the data set has been closed, the DCB can be used for another data set. If you do not close the data set before a task completes, the operating system tries to close it automatically. If the DCB is not available to the system at that time, the operating system abnormally ends the task, and data results can be unpredictable. The operating system, however, cannot automatically close any DCBs in dynamic storage (outside your program) or after the normal end of a program that was brought into virtual storage by the loader. Therefore, reentrant or loaded programs must include CLOSE macros for all open data sets.

Opening and Closing Data Sets: Considerations

This sections discusses the various OPEN and CLOSE considerations.

Parameter Lists with 31-Bit Addresses

You can code OPEN and CLOSE with MODE=31 to specify a long form parameter list that can contain 31-bit addresses. The default, MODE=24, specifies a short form parameter list with 24-bit addresses. If TYPE=J is specified, you must use the short form parameter list.

The short form parameter list must reside below 16 MB, but the calling program can be above 16 MB.

The long form parameter list can reside above or below 16 MB. VSAM and VTAM® access control blocks (ACBs) can reside above 16 MB. Although you can code MODE=31 on the OPEN or CLOSE call for a DCB, the DCB must reside below 16 MB. Therefore, the leading byte of the 4-byte DCB address must contain zeros. If the byte contains something other than zeros, an error message is issued. If an OPEN was attempted, the data set is not opened. If a CLOSE was attempted, the data set is not closed.

You need to keep the mode that is specified in the MF=L and MF=E versions of the OPEN macro consistent. The same is true for the CLOSE macro. If MODE=31 is specified in the MF=L version of the OPEN or CLOSE macro, MODE=31 must also be coded in the corresponding MF=E version of the macro. Unpredictable results occur if the mode that is specified is not consistent.

Open and Close of Multiple Data Sets at the Same Time

An OPEN or CLOSE macro can be used to begin or end processing of more than one data set. Simultaneous opening or closing is faster than issuing separate macros. However, additional storage space is required for each data set specified. The examples in Figure 55 on page 322 and Figure 57 on page 332 show how to code simultaneous open and close operations.

Factors to Consider When Allocating Direct Access Data Sets

When the system allocates a new SMS data set with DSORG=PS or no DSORG, the access methods treat the data set as being null, that is, having no data. A program can safely read the data set before data has been written in it. This means the first GET or first CHECK for a READ causes the EODAD routine to be called.

For data sets other than system managed with DSORG=PS or null, the program will receive unpredictable results such as reading residual data from a prior user, getting an I/O error, or getting an ABEND. Reading residual data can cause your program to appear to run correctly, but you can get unexpected output from the residual data. You can use one of the following methods to make the data set appear null:

1. At allocation time, specify a primary allocation value of zero; such as `SPACE=(TRK,(0,10))` or `SPACE=(CYL,(0,50))`. This technique does not work with a VIO data set because creation includes the secondary space amount.
2. After allocation time, put an end-of-file mark at the beginning of the data set, by running a program that opens the data set for output and closes it without writing anything.

After you delete your data set containing confidential data, you can be certain another user cannot read your residual data if you use the erase feature described in “Erasing DASD Data” on page 62.

Factors to Consider When Opening and Closing Data Sets

When you open and close data sets, consider the following factors:

- Data sets on DASD can be shared by two or more tasks or two or more DCBs. See “Chapter 24. Sharing Non-VSAM Data Sets” on page 365 and “Sharing PDSEs” on page 460 for more information.
- Volume disposition specified in the OPEN or CLOSE macro can be overridden by the system if necessary. However, you need not be concerned; the system automatically requests the mounting and demounting of volumes, depending on the availability of devices at a particular time. For more information about volume disposition see *z/OS MVS JCL User’s Guide*.

Open/Close/EOV Errors

There are two classes of errors that can occur during open, close, and end-of-volume processing: determinate and indeterminate errors. Determinate errors are errors associated with an ABEND issued by OPEN, CLOSE, or EOV. For example, a condition associated with the 213 completion code with a return code of 04 might be detected during open processing, indicating that the data set label could not be found for a data set being opened. In general, the OPEN, CLOSE and other system functions attempt to react to errors with return codes and determinate abends; however, in some cases, the result is indeterminate errors, such as program checks. In such cases, you should examine the last action taken by your program. Pay particular attention to bad addresses supplied by your program or overlaid storage.

If a determinate error occurs during the processing resulting from a concurrent OPEN or CLOSE macro, the system attempts to forcibly close the DCBs associated with a given OPEN or CLOSE macro. You can also immediately end the task abnormally by coding a DCB ABEND user exit routine that shows the immediate termination option. For more information on the DCB ABEND exit see “DCB ABEND Exit” on page 505. You can also request the DELAY option. In that case, when all open or close processing is completed, abnormal end processing is started. Abnormal end involves forcing all DCBs associated with a given OPEN or CLOSE macro to close status, thereby freeing all storage devices and other system resources related to the DCBs.

If an indeterminate error (such as a program check) occurs during open, close, or EOV processing, no attempt is made by the system control program to complete concurrent open or close processing. The DCBs associated with the OPEN or CLOSE macro are forced to close status if possible, and the resources related to each DCB are freed.

To determine the status of any DCB after an error, check the OPEN (or CLOSE) return code in register 15 or test DCBOFOPN. See *z/OS DFSMS Macro Instructions for Data Sets*.

During task termination, the system issues a CLOSE macro for each data set that is still open. If the task is terminating abnormally due to a determinate system ABEND for an output QSAM data set, the close routines that would normally finish processing buffers are bypassed. Any outstanding I/O requests are purged. Thus, your last data records might be lost for a QSAM output data set.

You should close an indexed sequential data set before task termination because, if an I/O error is detected, the ISAM close routines cannot return the application program registers to the SYNAD routine, causing unpredictable results.

Installation Exits

Four installation exit routines are provided for abnormal end with ISO/ANSI Version 3 or Version 4 tapes.

- The label validation exit is entered during OPEN/EOV if an invalid label condition is detected and label validation has not been suppressed. Invalid conditions include incorrect alphanumeric fields, nonstandard values (for example, RECFM=U, block size greater than 2048, or a zero generation number), invalid label sequence, nonsymmetrical labels, invalid expiration date sequence, and duplicate data set names. However, Version 4 tapes allow block size greater than 2048, invalid expiration date sequence, and duplicate data set names.
- The validation suppression exit is entered during OPEN/EOV if volume security checking has been suppressed, if the volume label accessibility field contains an ASCII space character, or if RACF accepts a volume and the accessibility field does not contain an uppercase A through Z .
- The volume access exit is entered during OPEN/EOV if a volume is not RACF protected and the accessibility field in the volume label contains an ASCII uppercase A through Z .
- The file access exit is entered after locating a requested data set if the accessibility field in the HDR1 label contains an ASCII uppercase A through Z.

ISO/ANSI Version 4 tapes also permits special characters !*"%&'()+,-./:;<=>?_ and numeric 0-9.

Related reading: For additional information about ISO/ANSI Version 3 or Version 4 installation exits see *z/OS DFSMS Installation Exits*.

Positioning Volumes

Volume positioning is releasing the DASD or tape volume or rotating the tape volume so that the read-write head is at a particular point on the tape. The following sections discuss the steps in volume positioning: releasing the volume, processing end-of-volume, positioning the volume.

Releasing Data Sets and Volumes

You are offered the option of being able to release data sets, and the volumes the data sets reside on when your task is no longer using them. If you are not sharing data sets, these data sets would otherwise remain unavailable for use by other tasks until the job step that opened them ends.

There are two ways to code the CLOSE macro that can result in releasing a data set and the volume on which it resides at the time the data set is closed:

1. Together with the FREE=CLOSE parameter of the DD statement, you can code:

   ```
   CLOSE   (DCB1,DISP) or
   CLOSE   (DCB1,REWIND)
   ```

   See *z/OS MVS JCL Reference* for information about using and coding the FREE=CLOSE parameter of the DD statement.
2. If you do not code FREE=CLOSE on the DD statement, you can code:

   ```
   CLOSE   (DCB1,FREE)
   ```

In either case, tape data sets and volumes are freed for use by another job step. Data sets on direct access storage devices are freed and the volumes on which they reside are freed if no other data sets on the volume are open. Additional information on volume disposition is provided in *z/OS MVS JCL User's Guide*.

If you issue a CLOSE macro with the TYPE=T parameter, the system does not release the data set or volume. They can be released using a subsequent CLOSE without TYPE=T or by the unallocation of the data set.

See *z/OS DFSMS Macro Instructions for Data Sets* for more information about coding restrictions on the CLOSE macro.

Processing End-of-Volume

The access methods pass control to the data management end-of-volume routine when another volume or concatenated data set is present and any of the following conditions is detected:

- Tape mark (input tape volume).
- File mark or end of last extent (input direct access volume).
- End-of-data indicator (input device other than magnetic tape or direct access volume). An example of this would be the last card read on a card reader.
- End of reel or cartridge (output tape volume).
- End of last allocated extent (output direct access volume).
- Application program issued an FEOV macro.

An exception is that the system calls your optional DCB OPEN exit routine instead of your optional EOV exit routine if all of the following are true:

- You are reading a concatenation.
- You read the end of a data set other than the last or issued an FEOV macro on its last volume.
- You turned on the DCB “unlike” attributes bit. See “Concatenating Unlike Data Sets” on page 384.

If the LABEL parameter of the associated DD statement shows standard labels, the end-of-volume routine checks or creates standard trailer labels. If SUL or AUL is specified, control is passed to the appropriate user label routine if it is specified in your exit list.

If multiple volume data sets are specified in your DD statement, automatic volume switching is accomplished by the end-of-volume routine. When an end-of-volume condition exists on an output data set, additional space is allocated as indicated in your DD statement. If no more volumes are specified or if more than specified are required, the storage is obtained from any available volume on a device of the same type. If no such volume is available, the system issues an ABEND.

If you perform multiple opens and closes without writing any user data in the area of the end-of-tape reflective marker, then header and trailer labels can be written past the marker. Access methods detect the marker. Because the creation of empty data sets does not involve access methods, the end-of-tape marker is not detected, which can cause the tape to run off the end of the reel.

Positioning During End-of-Volume

When a tape end-of-volume condition is detected and the system does not need the drive for another tape, the system positions the volume according to the disposition specified in the DD statement unless the volume disposition is specified in the OPEN macro. Volume positioning instructions for a sequential data set on magnetic tape can be specified as LEAVE or REREAD.

Using the OPEN Macro to Position Tape Volumes

If the tape was last read forward, LEAVE and REREAD have the following effects.

LEAVE— Positions a labeled tape to the point following the tape mark that follows the data set trailer label group. Positions an unlabeled volume to the point following the tape mark that follows the last block of the data set.

REREAD—Positions a labeled tape to the point preceding the data set header label group. Positions an unlabeled tape to the point preceding the first block of the data set.

If the tape was last read backward, LEAVE and REREAD have the following effects.

LEAVE—Positions a labeled tape to the point preceding the data set header label group, and positions an unlabeled tape to the point preceding the first block of the data set.

REREAD—Positions a labeled tape to the point following the tape mark that follows the data set trailer label group. Positions an unlabeled tape to the point following the tape mark that follows the last block of the data set.

Using the DISP Parameter to Position Volumes

If however you want to position the current volume according to the option specified in the DISP parameter of the DD statement, you code DISP in the OPEN macro.

DISP specifies that a tape volume is to be disposed of in the manner implied by the DD statement associated with the data set. Direct access volume positioning and disposition are not affected by this parameter of the OPEN macro. There are several dispositions that can be specified in the DISP parameter of the DD statement; DISP can be PASS, DELETE, KEEP, CATLG, or UNCATLG.

The resultant action when an end-of-volume condition arises depends on (1) how many tape units are allocated to the data set, and (2) how many volumes are specified for the data set in the DD statement. The UNIT and VOLUME parameters of the DD statement associated with the data set determine the number of tape units allocated and the number of volumes specified. If the number of volumes is greater than the number of units allocated, the current volume will be rewound and unloaded. If the number of volumes is less than or equal to the number of units, the current volume is merely rewound.

For magnetic tape volumes that are not being unloaded, positioning varies according to the direction of the last input operation and the existence of tape labels. When a JCL disposition of PASS or RETAIN® is specified, the result is the same as the OPEN or CLOSE LEAVE option. The CLOSE disposition option takes precedence over the OPEN option and the OPEN and CLOSE disposition options take precedence over the JCL.

Forcing End-of-Volume

The FEOV macro directs the operating system to start the end-of-volume processing before the physical end of the current volume is reached. If another volume has been specified for the data set or a data set is concatenated after the current data set, volume switching takes place automatically. The REWIND and LEAVE volume positioning options are available.

If an FEOV macro is issued for a spanned multivolume data set that is being read using QSAM, errors can occur when the next GET macro is issued. Make sure that each volume begins with the first (or only) segment of a logical record. Input routines cannot begin reading in the middle of a logical record.

The FEOV macro can only be used when you are using BSAM or QSAM. FEOV is ignored if issued for a SYSOUT data set. If you issue FEOV for a spooled input data set, control passes to your end-of-data (EODAD) routine or your program is positioned to read the next data set in the concatenation.

Managing SAM Buffer Space

The operating system provides several methods of buffer acquisition and control. Each buffer (virtual storage area used for intermediate storage of I/O data) usually corresponds in length to the size of a block in the data set being processed.

You can assign more than one buffer to a data set by associating the buffer with a buffer pool. A buffer pool must be constructed in a virtual storage area allocated for a given number of buffers of a given length.

The number of buffers you assign to a data set should be a trade-off against the frequency with which you refer to each buffer. A buffer that is not referred to for a fairly long period could be paged out. If much of this were allowed, throughput could decrease.

Using QSAM, buffer segments and buffers within the buffer pool are controlled automatically by the system. However, you can notify the system that you are finished processing the data in a buffer by issuing a release (RELSE) macro for input, or a truncate (TRUNC) macro for output. This simple buffering technique can be used to process a sequential or an indexed sequential data set. IBM recommends not using the RELSE or QSAM TRUNC macros because they can cause your program to become dependent on the size of each block.

When using QSAM to process tape blocks larger than 32 760 bytes, you must let the system build the buffer pool automatically during OPEN. The macros GETPOOL, BUILD, and BUILDRCD do not support the large block size or buffer size. If, during QSAM OPEN, or a BSAM OPEN with a nonzero BUFNO the system finds that the DCB has a buffer pool, and that the buffer length is smaller than the data set block size, an ABEND 013 is issued.

For QSAM, IBM recommends that you let the system build the buffer pool automatically during OPEN and omit the BUFL parameter. This simplifies your program. It permits concatenation of data sets in any order of block size. If you code RMODE31=BUFF on the DCBE macro, the system attempts to get buffers above the line.

When you use BSAM or BPAM, OPEN builds a buffer pool only if you code a nonzero value for BUFNO. OPEN issues ABEND 013-4C if BUFL is nonzero and is less than BLKSIZE in the DCB or DCBE, depending on whether you are using LBI. If the system builds the buffer pool for a BSAM user, the buffer pool resides below the 16 MB line.

If you use the basic access methods, you can use buffers as work areas rather than as intermediate storage areas. You can control the buffers in a buffer pool directly by using the GETBUF and FREEBUF macros.

For BSAM, IBM recommends that you allocate data areas or buffers through GETMAIN, STORAGE, or CPOOL macros and not through BUILD, GETPOOL, or by the system during OPEN. Allocated areas can be above the line. Areas that you allocate can be better integrated with your other areas.

Constructing a Buffer Pool

Buffer pool construction can be accomplished using any of the following techniques:
- Statically in an area that you provide, using the BUILD macro
- Explicitly in subpool 0, using the GETPOOL macro
- Automatically, by the system, when the data set is opened

Recommendation: For QSAM, use the automatic technique so that the system can rebuild the pool automatically when using concatenated data sets.

For the basic access methods, these techniques cannot build buffers above the 16 MB line or build buffers longer than 32 760 bytes.

If QSAM is used, the buffers are automatically returned to the pool when the data set is closed. If you did not use the BUILD macro and the buffer pool is not above the 16 MB line due to RMODE31=BUFF on the DCBE macro, you should use the FREEPOOL macro to return the virtual storage area to the system. If you code RMODE31=BUFF on a DCBE macro, then FREEPOOL has no effect and is optional. The system automatically frees the buffer pool.

For both data areas and buffers that have virtual addresses greater than 16 MB or less than 16 MB, the real address can exceed 2 GB. IBM recommends that when you obtain storage for buffers or data areas with GETMAIN or STORAGE that you specify that the real addresses can be above the 2 GB bar. This has no effect on your application program unless it deals with real storage addresses, which is uncommon.

In some rare applications, fullword or doubleword alignment of a block within a buffer is significant. You can specify in the DCB that buffers are to start on either a doubleword boundary or on a fullword boundary that is not also a doubleword boundary (by coding BFALN=D or F). If doubleword alignment is specified for format-V records, the fifth byte of the first record in the block is so aligned. For that reason, fullword alignment must be requested to align the first byte of the variable-length record on a doubleword boundary. The alignment of the records following the first in the block depends on the length of the previous records.

Buffer alignment provides alignment for only the buffer. If records from ASCII magnetic tape are read and the records use the block prefix, the boundary alignment of logical records within the buffer depends on the length of the block prefix. If the length is 4, logical records are on fullword boundaries. If the length is 8, logical records are on doubleword boundaries.

If you use the BUILD macro to construct the buffer pool, alignment depends on the alignment of the first byte of the reserved storage area.

When you code RMODE31=BUFF for QSAM, the theoretical upper limit for the size of the buffer pool is 2 GB. This imposes a limit on the buffer size, and thus on block size, of 2 GB divided by the number of buffers. If the system is to build the buffer pool, and the computed buffer pool size exceeds 2 GB, an ABEND 013 is issued. In practice, you can expect maximum buffer pool size to be less than 2 GB because of maximum device block sizes.

Building a Buffer Pool

When you know both the number and the size of the buffers required for a given data set before program assembly, you can reserve an area of the appropriate size to be used as a buffer pool. Any type of area can be used–for example, a predefined storage area or an area of coding no longer needed.

A BUILD macro, issued during execution of your program, uses the reserved storage area to build a buffer pool. The address of the buffer pool must be the same as that specified for the buffer pool control block (BUFCB) in your DCB. The BUFCB parameter cannot refer to an area that resides above the 16 MB line. The buffer pool control block is an 8 byte field preceding the buffers in the buffer pool. The number (BUFNO) and length (BUFL) of the buffers must also be specified. The length of BUFL must be at least the block size.

When the data set using the buffer pool is closed, you can reuse the area as required. You can also reissue the BUILD macro to reconstruct the area into a new buffer pool to be used by another data set.

You can assign the buffer pool to two or more data sets that require buffers of the same length. To do this, you must construct an area large enough to accommodate the total number of buffers required at any one time during execution. That is, if each of two data sets requires 5 buffers (BUFNO=5), the BUILD macro should specify 10 buffers. The area must also be large enough to contain the 8 byte buffer pool control block.

You can issue the BUILD macro in 31-bit mode, but the buffer area cannot reside above the line and be associated with a DCB. In any case, real addresses can point above the 2 GB bar.

Building a Buffer Pool and a Record Area

The BUILDRCD macro, like the BUILD macro, causes a buffer pool to be constructed in an area of virtual storage you provide. Also, BUILDRCD makes it possible for you to access variable-length, spanned records as complete logical records, rather than as segments.

You must be processing with QSAM in the locate mode and you must be processing either VS/VBS or DS/DBS records, if you want to access the variable-length, spanned records as logical records. If you issue the BUILDRCD macro before the data set is opened, or during your DCB exit routine, you automatically get logical records rather than segments of spanned records.

Only one logical record storage area is built, no matter how many buffers are specified; therefore, you cannot share the buffer pool with other data sets that might be open at the same time.

You can issue the BUILDRCD macro in 31-bit mode, but the buffer area cannot reside above the line and be associated with a DCB.

Getting a Buffer Pool

If a specified area is not reserved for use as a buffer pool, or if you want to delay specifying the number and length of the buffers until execution of your program, you can use the GETPOOL macro. This macro allows you to vary the size and number of buffers according to the needs of the data set being processed. The storage resides below the line in subpool zero.

The GETPOOL macro causes the system to allocate a virtual storage area to a buffer pool. The system builds a buffer pool control block and stores its address in the data set's DCB. If you choose to issue the GETPOOL macro, issue it either before opening the data set or during your DCB's OPEN exit routine.

When using GETPOOL with QSAM, specify a buffer length (BUFL) at least as large as the block size or omit the BUFL parameter.

Constructing a Buffer Pool Automatically

If you have requested a buffer pool and have not used a BUILD or GETPOOL macro for the DCB by the end of your DCB exit routine, the system automatically allocates virtual storage space for a buffer pool. The buffer pool control block is also assigned and the pool is associated with a specific DCB. For BSAM, a buffer pool is requested by specifying BUFNO. For QSAM, BUFNO can be specified or permitted to default in the following ways:

n—Extended format but not compressed format and not LBI (n = 2 × blocks per track × number of stripes)

1—Compressed format data set, PDSE, SYSIN, SYSOUT, SUBSYS, USS file

2—Block size > 32 760

3—2540 Card Reader or Punch

5—All others

If you are using the basic access method to process an indexed sequential or direct data set, you must specify dynamic buffer control. Otherwise, the system does not construct the buffer pool automatically.

If all of your GET, PUT, PUTX, RELSE, and TRUNC macros for a particular DCB are issued in 31-bit mode, then you should consider supplying a DCBE macro with RMODE31=BUFF.

Because a buffer pool obtained automatically is not freed automatically when you issue a CLOSE macro unless the system recognized your specification of RMODE31=BUFF on the DCBE macro, you should also issue a FREEPOOL or FREEMAIN macro (see "Freeing a Buffer Pool" on page 345).

Freeing a Buffer Pool

Any buffer pool assigned to a DCB either automatically by the OPEN macro (except when dynamic buffer control is used or the system is honoring RMODE31=BUFF on the DCBE macro), or explicitly by the GETPOOL macro should be released before your program is completed. The FREEPOOL macro should be issued to release the virtual storage area when the buffers are no longer needed. When you are using the queued access technique, you must close the data set first. If you are not using the queued access method, it is still advisable to close the data set first.

If the OPEN macro was issued while running in problem state, protect key of zero, a buffer pool that was obtained by OPEN should be released by issuing the FREEMAIN macro instead of the FREEPOOL macro. This is necessary because the buffer pool acquired under these conditions will be in storage assigned to subpool 252 (in user key storage).

Constructing a Buffer Pool: Examples

Figure 59 and Figure 60 on page 346 show several possible methods of constructing a buffer pool. They do not consider the method of processing or controlling the buffers in the pool.

In Figure 59, a static storage area named INPOOL is allocated during program assembly.

```
            ...                          Processing
            BUILD     INPOOL,10,52       Structure a buffer pool
            OPEN      (INDCB,,OUTDCB,(OUTPUT))
            ...                          Processing
ENDJOB      CLOSE     (INDCB,,OUTDCB)
            ...                          Processing
            RETURN                       Return to system control
INDCB       DCB       BUFNO=5,BUFCB=INPOOL,EODAD=ENDJOB,---
OUTDCB      DCB       BUFNO=5,BUFCB=INPOOL,---
            CNOP      0,8                Force boundary alignment
INPOOL      DS        CL528              Buffer pool
            ...
```

Figure 59. Constructing a Buffer Pool from a Static Storage Area

The BUILD macro, issued during execution, arranges the buffer pool into 10 buffers, each 52 bytes long. Five buffers are assigned to INDCB and five to OUTDCB, as specified in the DCB macro for each. The two data sets share the buffer pool because both specify INPOOL as the buffer pool control block. Notice that an additional 8 bytes have been allocated for the buffer pool to contain the buffer pool control block.

In Figure 60 on page 346, two buffer pools are constructed explicitly by the GETPOOL macros.

```
          ...
          GETPOOL     INDCB,10,52             Construct a 10-buffer pool
          GETPOOL     OUTDCB,5,112            Construct a 5-buffer pool
          OPEN        (INDCB,,OUTDCB,(OUTPUT))
          ...
ENDJOB    CLOSE       (INDCB,,OUTDCB)
          FREEPOOL    INDCB                   Release buffer pools after all
*                                              I/O is complete
          FREEPOOL    OUTDCB
          ...
          RETURN                              Return to system control
INDCB     DCB         DSORG=PS,BFALN=F,LRECL=52,RECFM=F,EODAD=ENDJOB,---
OUTDCB    DCB         DSORG=IS,BFALN=D,LRECL=52,KEYLEN=10,BLKSIZE=104,       C
          ...     RKP=0,RECFM=FB,---
```

Figure 60. Constructing a Buffer Pool Using GETPOOL and FREEPOOL

Ten input buffers are provided, each 52 bytes long, to contain one fixed-length record. Five output buffers are provided, each 112 bytes long, to contain 2 blocked records plus an 8 byte count field (required by ISAM). Notice that both data sets are closed before the buffer pools are released by the FREEPOOL macros. The same procedure should be used if the buffer pools were constructed automatically by the OPEN macro.

Controlling Buffers

You can use several techniques to control which buffers are used by your program. The advantages of each depend to a great extent on the type of job you are doing. The queued access methods permits simple buffering. The basic access methods permits either direct or dynamic buffer control.

Although only simple buffering can be used to process an indexed sequential data set, buffer segments and buffers within a buffer pool are controlled automatically by the operating system.

Queued Access Method

The queued access methods provide three processing modes (move, data, and locate mode) that determine the extent of data movement in virtual storage. Move, data, and locate mode processing can be specified for either the GET or PUT macro. (Substitute mode is no longer supported; the system defaults to move mode.) The movement of a record is determined by the following modes.

Move Mode. The system moves the record from a system input buffer to your work area, or from your work area to an output buffer.

Data Mode (QSAM Format-V Spanned Records Only). Data mode works the same as the move mode, except only the data portion of the record is moved.

Locate Mode. The system does not move the record. Instead, the access method macro places the address of the next input or output buffer in register 1. For QSAM format-V spanned records, if you have specified logical records by specifying BFTEK=A or by issuing the BUILDRCD macro, the address returned in register 1 points to a record area where the spanned record is assembled or segmented.

PUT-Locate Mode. The PUT-locate routine uses the value in the DCBLRECL field to determine if another record will fit into your buffer. Therefore, when you write a

short record, you can get the largest number of records per block by modifying the DCBLRECL field before you issue a PUT-locate to get a buffer segment for the short record. Perform the following steps:

1. Record the length of the next (short) record into DCBLRECL.
2. Issue PUT-locate.
3. Move the short record into the buffer segment.

GET-Locate Mode. Two processing modes of the PUTX macro can be used with a GET-locate macro. The update mode returns an updated record to the data set from which it was read. The output mode transfers an updated record to an output data set. There is no actual movement of data in virtual storage. See *z/OS DFSMS Macro Instructions for Data Sets* for information about the processing mode specified by the parameter of the PUTX macro.

Basic Access Method

If you use a basic access method and want the system to assist in buffer control, you can control buffers directly by using the GETBUF macro to retrieve a buffer constructed. A buffer can then be returned to the pool by the FREEBUF macro. Because GETBUF does not support a buffer pool that is above the 16 MB line, IBM suggests that you write your own routine to allocate buffers above the line.

QSAM in an Application

The term *simple buffering* refers to the relationship of segments within the buffer. All segments in a simple buffer are together in storage and are always associated with the same data set. Each record must be physically moved from an input buffer segment to an output buffer segment. The record can be processed within either segment or in a work area.

If you use simple buffering, records of any format can be processed. New records can be inserted and old records deleted as required to create a new data set. The following examples of using QSAM use buffers that could have been constructed in any way previously described.

GET-locate, PUT-move, PUTX-output. Processed in an input buffer and then moved to an output buffer.

GET-move, PUT-locate. Moved from an input buffer to an output buffer where it can be processed.

GET-move, PUT-move. Moved from an input buffer to a work area where it can be processed and then moved to an output buffer.

GET-locate, PUT-locate. Processed in an input buffer, copied to an output buffer, and possibly processed some more.

GET-locate, PUTX-update. Processed in an input buffer and returned to the same data set.

GET-locate, PUT-move/PUTX-output. The GET macro (step A, Figure 61) locates the next input record to be processed.

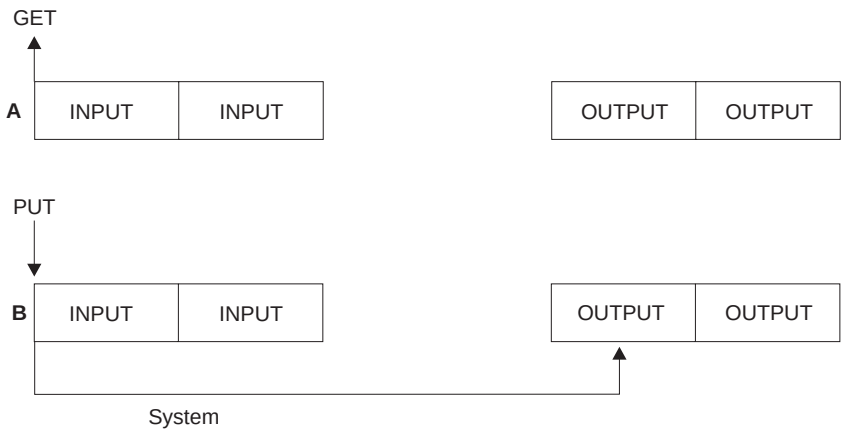


Figure 61. Simple Buffering with MACRF=GL and MACRF=PM

GET returns a record address in register 1. This address remains valid until the next GET or CLOSE for the DCB. Your program passes the address to the PUT macro in register 0. PUT copies the record synchronously.

The PUTX-output macro can be used in place of the PUT-move macro.

GET-move, PUT-locate. The PUT macro locates the address of the next available output buffer. PUT returns its address in register 1 and your program passes it to the GET macro in register 0.

On the GET macro you specify the address of the output buffer into which the system moves the next input record.

A filled output buffer is not written until the next PUT macro is issued. PUT returns a buffer address before GET moves a record. This means that when GET branches to the end-of-data routine because all data has been read, the output buffer still needs a record. Your program should replace the unpredictable output buffer content with another record, which you might set to blanks or zeros. The next PUT or CLOSE macro writes the record.

GET-move, PUT-move. The GET macro (step A, Figure 62 on page 349) specifies the address of the work area into which the system moves the next record from the input buffer.

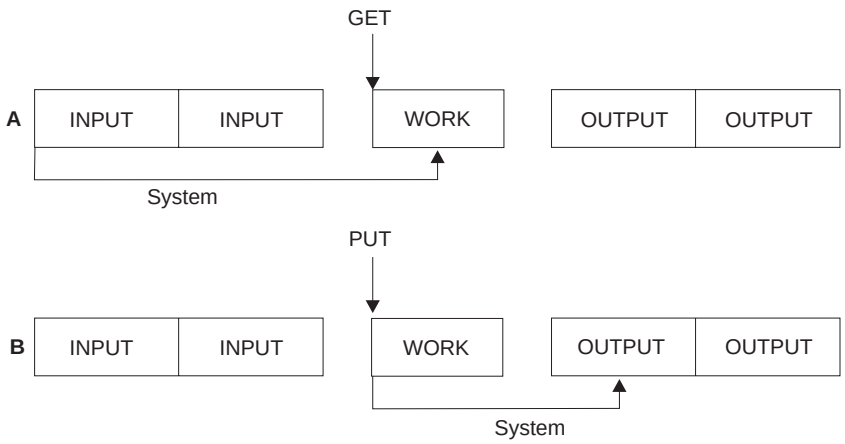


Figure 62. Simple Buffering with MACRF=GM and MACRF=PM

The PUT macro (step B, Figure 62) specifies the address of the work area from which the system moves the record into the next output buffer.

GET-locate, PUT-locate. The GET macro (step A, Figure 63) locates the address of the next available input buffer. GET returns the address in register 1.

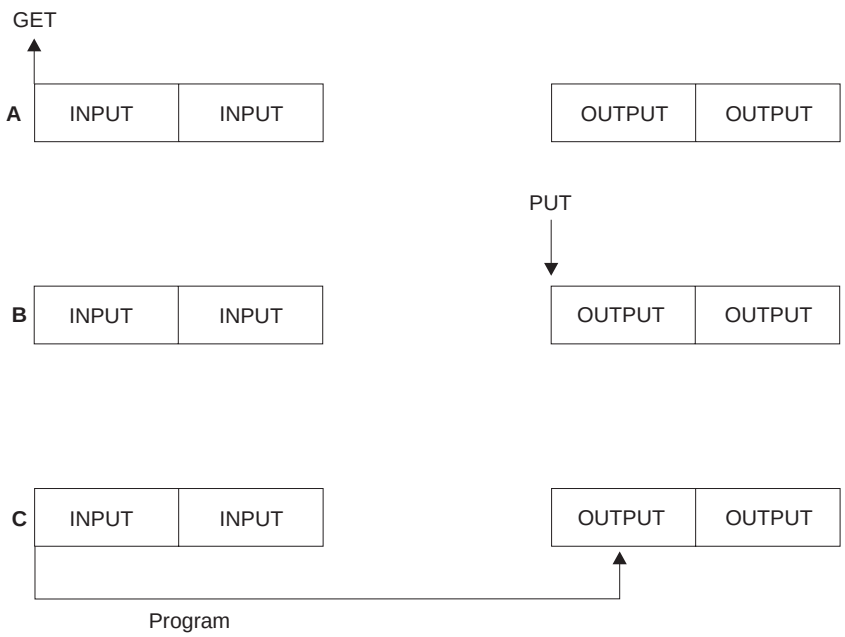


Figure 63. Simple Buffering with MACRF=GL and MACRF=PL

The PUT macro (step B, Figure 63) locates the address of the next available output buffer. PUT returns its address in register 1. You must then move the record from the input buffer to the output buffer (step C, Figure 63). Your program can process each record either before or after the move operation.

A filled output buffer is not written until the next PUT, TRUNC or CLOSE macro is issued.

Data Control Block (DCB)

Be careful not to issue an extra PUT before issuing CLOSE or FEOV. Otherwise, when the CLOSE or FEOV macro tries to write your last record, the extra PUT will write a meaningless record or produce a sequence error.

UPDAT mode. When a data set is opened with UPDAT specified (Figure 64), only GET-locate and PUTX-update are supported.

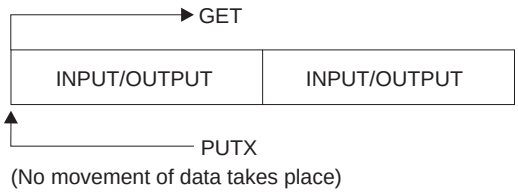


Figure 64. Simple Buffering with MACRF=GL and MACRF=PM-UPDAT Mode

The GET macro locates the next input record to be processed and returns its address in register 1. You can update the record and issue a PUTX macro that will cause the block to be written back in its original location in the data set after all the logical records in that block have been processed.

If you modify the contents of a buffer but do not issue a PUTX macro for that record, the system can still write the modified record block to the data set. This happens with blocked records when you issue a PUTX macro for one or more other records in the buffer.

Exchange Buffering

Exchange buffering is no longer supported. Its request is ignored by the system and move mode is used instead.

Choosing Buffering Techniques and GET/PUT Processing Modes

As you can see from the previous examples, the most efficient code is achieved by using automatic buffer pool construction, and GET-locate and PUTX-output with simple buffering. Table 29 summarizes the combinations of buffering techniques and processing modes you might use:

Table 29. Buffering Technique and GET/PUT Processing Modes

		Simple Input Buffering			
Actions	**GET-move PUT-locate**	**GET-move PUT-move**	**GET-locate PUT-locate**	**GET-locate PUT-move**	**GET-locate (logical record), PUT-locate**
Program must move record			X		X
System moves record	X	X		X	
System moves record segment					X
Work area required		X			
PUTX-output can be used				X	

Using Buffering Macros with Queued Access Method

This section describes how to use the RELSE and TRUNC macros.

RELSE—Release an Input Buffer

When using QSAM to read blocked records, you can direct the system to ignore the remaining records in the input buffer being processed. The next GET macro retrieves a record from another buffer. If format-V spanned records are being used, the next logical record obtained can begin on any segment in any subsequent block.

When you are using locate mode, the record address returned from the most recent GET macro remains valid until you issue the next GET. Issuing a RELSE macro does change the effect of a previous PUTX macro.

TRUNC—Truncate an Output Buffer

When using QSAM to write blocked records, you can issue the TRUNC macro to direct the system to write a short block. The first record in the next buffer is the next record processed by a PUT-output or a PUTX-output mode.

If the locate mode is being used, the system assumes a record has been placed in the buffer segment pointed to by the last PUT macro.

The last block of a data set is truncated by the CLOSE routine. A data set that contains format-F records with truncated blocks generally cannot be read as efficiently as a standard format-F data set.

A TRUNC macro issued against a PDSE does not create a short block because the block boundaries are not saved on output. On input, the system uses the block size specified in the DCB or DCBE for reading the PDSE. Logical records are packed into the user buffer without respect to the block size specified when the PDSE member was created.

In order to help the storage administrator find programs that issue a QSAM TRUNC macro for PDSEs, the SMF type 15 record (see *z/OS MVS System Management Facilities (SMF)*) contains an indicator that the program did it.

Recommendation: Avoid using the QSAM TRUNC macro. Many data set copying and backup programs reblock the records. This means they do not preserve the block boundaries that your program can have set.

Using Buffering Macros with Basic Access Method

This section describes how to use the GETBUF and FREEBUF macros.

GETBUF—Get a Buffer from a Pool

The GETBUF macro can be used with the basic access method to request a buffer from a buffer pool constructed by the BUILD, GETPOOL, or OPEN macro. That buffer pool resides below the 16 MB line even if you issue BUILD, GETPOOL, or OPEN in 31-bit mode. The address of the buffer is returned by the system in a register you specify when you issue the macro. If no buffer is available, the register contains a 0 instead of an address.

FREEBUF—Return a Buffer to a Pool

The FREEBUF macro is used with the basic access method to return a buffer to the buffer pool from which it was obtained by a GETBUF macro. Although the buffers need not be returned in the order in which they were obtained, they must be returned if you want to make them available for later GETBUF macros.

Chapter 23. Accessing Records

This chapter covers the following topics.

Topic	Page
Accessing Data with READ and WRITE	353
Accessing Data with GET and PUT	359
Analyzing I/O Errors	362

Accessing Data with READ and WRITE

The basic sequential access method (BSAM) and basic partitioned access method (BPAM) provide the READ, WRITE and TRUNC macros for transmitting data between virtual and auxiliary storage. These macros can be issued in 24- or 31-bit addressing mode. Macros are provided to help you manage buffers, or to do overlapped I/O.

The READ and WRITE macros process blocks, not records. Thus, you must block and unblock records. Buffers, allocated by either you or the operating system, are filled or emptied individually each time a READ or WRITE macro is issued. The READ and WRITE macros only start I/O operations. To ensure the operation is completed successfully, you must issue a CHECK, WAIT, or EVENTS macro to test the data event control block (DECB). The only exception is that, when the SYNAD or EODAD routine is entered, do not issue a CHECK, WAIT, or EVENTS macro for outstanding READ or WRITE requests.

Using the Data Event Control Block (DECB)

A data event control block is a 20- to 32-byte area reserved by each READ or WRITE macro. It contains the ECB, control information, and pointers to control blocks. It must reside below the 16 MB line. The DECB is described in *z/OS DFSMS Macro Instructions for Data Sets* and “Data Event Control Block” on page 487. The ECB is described in “Event Control Block” on page 489.

The DECB is examined by the CHECK routine when the I/O operation is completed to determine if an uncorrectable error or exceptional condition exists. If it does, CHECK passes control to your SYNAD routine. If you have no SYNAD routine, the task is abnormally ended.

Grouping Related Control Blocks in a Paging Environment

Code related control blocks (the DCB and DECB) and data areas (buffers and key areas) so they reside in the same area of your program. This reduces the number of paging operations required to read from and write to your data set.

Rule: DCB and DECBs must reside below 16 MB, but their central storage addresses can be above the 2 GB bar.

Using Overlapped I/O with BSAM

When using BSAM with overlapped I/O (multiple I/O requests outstanding at one time), you must use more than one DECB. Specify a different DECB for each I/O

© Copyright IBM Corp. 1987, 2002

request. For example, if you specify NCP=3 in your DCB for the data set and you are reading records from the data set, you can code the following macros in your program:

```
...
READ DECB1,...
READ DECB2,...
READ DECB3,...
CHECK DECB1
CHECK DECB2
CHECK DECB3
...
```

This is a more efficient technique:

```
READ DECB1,...
READ DECB2,...
READ DECB3,...
CHECK DECB1             Beginning of loop
READ DECB1,...
CHECK DECB2
READ DECB2,...
CHECK DECB3
READ DECB3,...
(Repeat the previous six macros until a
CHECK macro causes entry to EODAD routine.)
```

This is a generalized technique that supports any value for NCP:

1. Supply a DCBE with a nonzero MULTSDN and a nonzero MULTACC. For optimization you can select the MULTACC value to be half of the MULTSDN value.
2. Issue an OPEN macro. OPEN calculates an NCP value. If using LBI to process a tape data set and the block size is greater than 32 768, the minimum calculated NCP value is 2 and the maximum value is 16.
3. Allocate storage for a number of data areas equal to the NCP value and for an equal number of DECBs. They do not have to be contiguous. The DECBs must be below the 16 MB line, but the data areas can be above the 16 MB line. Central storage addresses can be above 2 GB. After each DECB you can add a word pointing to the next DECB and point the last one to the first DECB. This simplifies finding DECBs.
4. For each DECB area, copy a single DECB to it, and issue a READ or WRITE macro to it, supplying an appropriate data area address.

   The source of the DECB copy can be above or below the 16 MB line but the destination must be below the 16 MB line.
5. Repeat the following steps for each DECB until a CHECK macro for a READ causes entry to the end-of-data exit (EODAD) routine or until you have nothing more to write:
   a. Issue the CHECK macro (for oldest outstanding READ or WRITE)
   b. Process data in block if you are reading, or create a block in data area if you are writing.
   c. Issue the READ or WRITE macro to the DECB.
   d. Load a pointer to the next DECB (to get oldest outstanding READ or WRITE).
6. If you are writing, then issue a CHECK macro for each remaining outstanding DECB in the order of the WRITEs. If you are reading, do not issue another CHECK.
7. Issue a CLOSE macro and free storage.

Figure 84 on page 422 shows this technique, except for the FIND macro and DSORG=PO in the DCB macro. To process a sequential data set, code DSORG=PS. You can easily adapt this technique to use WRITE or READ.

Reading a Block

The READ macro retrieves a data block from an input data set and places it in a designated area of virtual storage. To permit overlap of the input operation with processing, the system returns control to your program before the read operation is completed. You must test the DECB created for the read operation for successful completion before the block is processed or the DECB is reused.

If an indexed sequential data set is being read, the block is brought into virtual storage and the address of the record is returned to you in the DECB.

When you use the READ macro for BSAM to read a direct data set with spanned records and keys, and you specify BFTEK=R in your DCB, the data management routines displace record segments after the first in a record by key length. This is called *offset reading*. With offset reading you can expect the block descriptor word and the segment descriptor word at the same locations in your buffer or buffers, even if you read the first segment of a record (preceded in the buffer by its key), or a subsequent segment (which does not have a key).

You can specify variations of the READ macro according to the organization of the data set being processed and the type of processing to be done by the system as follows.

Sequential and Partitioned

SF Read the data set sequentially.

SB Read the data set backward (magnetic tape, format-F, and format-U only). When RECFM=FBS, data sets with the last block truncated cannot be read backward.

Indexed Sequential

K Read the data set.

KU Read for update. The system maintains the device address of the record; thus, when a WRITE macro returns the record, no index search is required.

Direct

D Use the direct access method.

I Locate the block using a block identification.

K Locate the block using a key.

F Provide device position feedback.

X Maintain exclusive control of the block.

R Provide next address feedback.

U Next address can be a capacity record or logical record, whichever occurred first.

Writing a Block

The WRITE macro places a data block in an output data set from a designated area of virtual storage. The WRITE macro can also be used to return an updated data block to a data set. To permit overlap of output operations with processing, the system returns control to your program before the write operation is completed. You must test the DECB that is created for the write operation for successful completion before you reuse the DECB. For ASCII tape data sets, do not issue more than one WRITE on the same block, because the WRITE macro causes the data in the record area to be converted from EBCDIC to ASCII. Or, if CCSIDs are specified for ISO/ANSI V4 tapes, from the CCSID specified for the application program to the CCSID of the data records on tape.

As with the READ macro, you can specify variations of the WRITE macro according to the organization of the data set and type of processing to be done by the system as follows.

Sequential

SF Write the data set sequentially.

Indexed Sequential

K Write a block containing an updated record, or replace a record with a fixed, unblocked record having the same key. The record to be replaced need not have been read into virtual storage.

KN Write a new record or change the length of a variable-length record.

Direct

SD Write a dummy fixed-length record. (BDAM load mode)

SZ Write a capacity record (R0). The system supplies the data, writes the capacity record, and advances to the next track. (BDAM load mode)

SFR Write the data set sequentially with next-address feedback. (BDAM load mode, variable spanned)

D Use the direct access method.

I Search argument identifies a block.

K Search argument is a key.

A Add a new block.

F Provide record location data (feedback).

X Release exclusive control.

Ensuring I/O Initiation with the TRUNC Macro

The TRUNC macro is not required for BSAM or BPAM. It is necessary only if you have supplied a DCBE with a nonzero MULTACC value and you are about to issue WAIT or EVENTS for a DECB instead of issuing a CHECK macro. See “Waiting for Completion of a Read or Write Operation” on page 357 and “DASD and Tape Performance” on page 391.

Testing Completion of a Read or Write Operation

When processing a data set, you can test for completion of a READ or WRITE request by issuing a CHECK macro. The system tests for errors and exceptional

conditions in the data event control block (DECB). Successive CHECK macros issued for the same data set must be issued in the same order as the associated READ and WRITE macros.

The check routine passes control to the appropriate exit routines specified in the DCB or DCBE for error analysis (SYNAD) or, for sequential or PDSs, end-of-data (EODAD). It also automatically starts the end-of-volume procedures (volume switching or extending output data sets).

If you specify OPTCD=Q in the DCB, CHECK causes input data to be converted from ASCII to EBCDIC or, if CCSIDs are specified for ISO/ANSI V4 tapes, from the CCSID of the data records on tape to the CCSID specified for the application program.

If the system calls your SYNAD or EODAD routine, then all other I/O requests for that DCB have been terminated, although they have not necessarily been posted. There is no need to test them for completion or issue CHECK for them.

Waiting for Completion of a Read or Write Operation

When processing a data set, you can test for completion of any READ or WRITE request by issuing a WAIT or EVENTS macro. You can choose to do this so you can wait for multiple unrelated events. You can process whichever events have completed. The I/O operation is overlapped with processing, but the DECB is not checked for errors or exceptional conditions, nor are end-of-volume procedures initiated. Your program must perform these operations.

If you use overlapped BSAM or BPAM READ or WRITE macros, your program can run faster if you use the MULTACC parameter on the DCBE macro. If you do that and use WAIT or EVENTS for the DCB, then you must also use the TRUNC macro. See TRUNC information in “Ensuring I/O Initiation with the TRUNC Macro” on page 356 and “DASD and Tape Performance” on page 391.

For BDAM and BISAM, a WAIT macro must be issued for each READ or WRITE macro if MACRF=C is not coded in the associated DCB. When MACRF=C is coded, a CHECK macro must be issued for each READ or WRITE macro. Because the CHECK macro incorporates the function of the WAIT macro, a WAIT is normally unnecessary. The EVENTS macro or the ECBLIST form of the WAIT macro can be useful, though, in selecting which of several outstanding events should be checked first. Each operation must then be checked or tested separately.

Example:

1. You have opened an input DCB for BSAM with NCP=2, and an output DCB for BISAM with NCP=1 and without specifying MACRF=C.
2. You have issued two BSAM READ macros and one BISAM WRITE macro.
3. You now issue the WAIT macro with ECBLIST pointing to the BISAM DECB and the first BSAM DECB. (Because BSAM requests are serialized, the first request must execute before the second.)
4. When you regain control, inspect the DECBs to see which has completed (second bit on).
   a. If it was BISAM, issue another WRITE macro.
   b. If it was BSAM, issue a CHECK macro and then another READ macro.

Handling Exceptional Conditions on Tape

In this section an exceptional condition is any READ or WRITE macro that did not have normal completion.

Most programs do not care about how much data in the data set is on each volume and if there is a failure, they do not care what the failure was. A person is more likely to want to know the cause of the failure.

In some cases you might want to take special actions before some of the system's normal processing of an exceptional condition. One such exceptional condition is reading a tape mark and another such exceptional condition is writing at the end of the tape.

With BSAM, your program can detect when it has reached the end of a magnetic tape and do some processing before BSAM's normal processing to go to another volume. To do that, do the following:

1. Instead of issuing the CHECK macro, issue the WAIT or EVENTS macro. Use the ECB, which is the first word in the DECB. The first byte is called the post code. As a minor performance enhancement, you can skip all three macros if the second bit of the post code already is 1.
2. Inspect the post code. Do one of the following:
   a. Post code is X'7F': The READ or WRITE is successful. If you are reading and either the tape label type is AL or OPTCD=Q is in effect, then you must issue the CHECK macro in order to convert between ASCII and EBDCIC. Otherwise, the CHECK is optional and you can continue normal processing as if your program had issued the CHECK macro.
   b. Post code is not X'7F': You cannot issue another READ or WRITE successfully unless you take one of the following actions. All later READs or WRITEs that you issued for the DCB have post codes that you cannot predict, but they are guaranteed not to have started. If your only reason to issue WAIT or EVENTS is to wait for multiple events, then issue CHECK to handle the exceptional condition. If part or all of your purpose was to handle a certain exceptional condition, such as a full volume, take one of the following actions:
      - If the post code is X'41' and the status indicators show unit exception and an error status, you read a tape mark or wrote at the end of the volume. Coincidentally, however, you got an input or output error that prevented reading or writing to the block. You can still take one of the preceding actions. Issuance of CHECK causes entry to your SYNAD routine or issuance of an ABEND. You can issue CLOSE to bypass the SYNAD routine, but CLOSE might detect another input or output error and issue an ABEND.
      - If the post code and the status indicators are different from the preceding ones, the system probably did not read or write to the data block. Issue CHECK or CLOSE. CHECK causes entry to your SYNAD routine or the issuance of an ABEND. The system discards all other WRITEs with an unpredictable post code or no post code.

Accessing Data with GET and PUT

The queued access method provides GET and PUT macros for transmitting data within virtual storage. The GET and PUT macros can be issued in 24- or 31-bit addressing mode. The QSAM GET and PUT macros automatically block and unblock the records that are stored and retrieved. The queued access method anticipates required buffering and overlaps I/O operations with instruction stream processing.

Because the operating system controls buffer processing, you can use as many I/O buffers as needed without reissuing GET or PUT macros to fill or empty buffers. Usually, more than one input block is in storage at a time, so I/O operations do not delay record processing.

Because the operating system overlaps I/O with processing, you need not test for completion, errors, or exceptional conditions. After a GET or PUT macro is issued, control is not returned to your program until an input area is filled or an output area is available. Exits to error analysis (SYNAD) and end-of-volume or end-of-data (EODAD) routines are automatically taken when necessary.

GET—Retrieve a Record

The GET macro obtains a record from an input data set. It operates in a logical-sequential and device-independent manner. The GET macro schedules the filling of input buffers, unblocks records, and directs input error recovery procedures. For spanned-record data sets, it also merges record segments into logical records.

After all records have been processed and the GET macro detects an end-of-data indication, the system automatically checks labels on sequential data sets and passes control to your end-of-data exit (EODAD) routine. If an end-of-volume condition is detected for a sequential data set, the system automatically switches volumes if the data set extends across several volumes, or if concatenated data sets are being processed.

If you specify OPTCD=Q in the DCB or DD statement, or if the LABEL parameter on the DD statement specifies ISO/ANSI labels, the GET macro converts input data from ASCII to EBCDIC. If CCSIDs are specified for ISO/ANSI V4 tapes, it converts input data from the CCSID of the data records on tape to the CCSID specified for the application program. This parameter is supported only for a magnetic tape that does not have IBM standard labels.

PUT—Write a Record

The PUT macro writes a record into an output data set. Like the GET macro, it operates in a logical-sequential and device-independent manner. As required, the PUT macro blocks records, schedules the emptying of output buffers, and handles output error correction procedures. For sequential data sets, it also starts automatic volume switching and label creation, and also segments records for spanning.

If you specify OPTCD=Q in the DCB or DD statement, or if the LABEL parameter on the DD statement specifies ISO/ANSI labels, the PUT macro causes output to be converted from EBCDIC to ASCII. If CCSIDs are specified for ISO/ANSI V4 tapes, it causes output to be converted from the CCSID specified for the application program to the CCSID of the data records on tape. This parameter is supported only for a magnetic tape that does not have IBM standard labels. If the tape has ISO/ANSI labels (LABEL=(,AL)), the system assumes OPTCD=Q.

If the PUT macro is directed to a card punch or printer, the system automatically adjusts the number of records or record segments per block of format-F or format-V blocks to 1. Thus, you can specify a record length (LRECL) and block size (BLKSIZE) to provide an optimum block size if the records are temporarily placed on magnetic tape or a direct access volume.

For spanned variable-length records, the block size must be equivalent to the length of one card or one print line. Record size might be greater than block size in this case.

PUTX—Write an Updated Record

Use the PUTX macro to update a data set or to write a new output data set using records from an input data set as a base. PUTX updates, replaces, or inserts records from existing data sets, but does not create records.

When you use the PUTX macro to update, each record is returned to the data set referred to by a previous locate mode GET macro. The buffer containing the updated record is flagged and written back to the same location on the direct access storage device where it was read. The block is not written until a GET macro is issued for the next buffer, except when a spanned record is to be updated. In that case, the block is written with the next GET macro.

When you use the PUTX macro to write a new output data set, you can add new records by using the PUT macro. As required, the PUTX macro blocks records, schedules the writing of output buffers, and handles output error correction procedures.

PDAB—Parallel Input Processing (QSAM Only)

QSAM parallel input processing can be used to process two or more input data sets concurrently, such as sorting or merging several data sets at the same time. QSAM parallel input processing eliminates the need for issuing a separate GET macro to each DCB processed. The GET routine for parallel input processing selects a DCB with a ready record and then transfers control to the normal GET routine. If there is no DCB with a ready record, the GET routine issues a multiple WAIT macro.

Parallel input processing provides a logical input record from a queue of data sets with equal priority. The function supports QSAM with input processing, simple buffering, locate or move mode, and fixed-, variable-, or undefined-length records. Spanned records, track-overflow records, dummy data sets, and SYSIN data sets are not supported.

Parallel input processing can be interrupted at any time to retrieve records from a specific data set, or to issue control instructions to a specific data set. When the retrieval process has been completed, parallel input processing can be resumed.

Data sets can be added to or deleted from the data set queue at any time. You should note, however, that, as each data set reaches an end-of-data condition, the data set must be removed from the queue with the CLOSE macro before a subsequent GET macro is issued for the queue. Otherwise, the task could be ended abnormally.

Using Parallel Data Access Blocks (PDAB)

You specify a request for parallel input processing by including the address of a parallel data access block (PDAB) in the DCB exit list. For more information on the DCB exit list, see “DCB Exit List” on page 501.

Use the PDAB macro to create and format a work area that identifies the maximum number of DCBs that can be processed at any one time. If you exceed the maximum number of entries specified in the PDAB macro when adding a DCB to the queue with the OPEN macro, the data set will not be available for parallel input processing. However, it will be available for sequential processing.

When issuing a parallel GET macro, register 1 must always point to a PDAB. You can load the register or let the GET macro do it for you. When control is returned to you, register 1 contains the address of a logical record from one of the data sets in the queue. Registers 2 - 13 contain their original contents at the time the GET macro was issued. Registers 14, 15, and 0 are changed.

Through the PDAB, you can find the data set from which the record was retrieved. A fullword address in the PDAB (PDADCBEP) points to the address of the DCB. It should be noted that this pointer could be invalid from the time a CLOSE macro is issued to the issuing of the next parallel GET macro.

In Figure 65, not more than three data sets (MAXDCB=3 in the PDAB macro) are open for parallel processing at a time.

```
          ...
          OPEN  (DATASET1,(INPUT),DATASET2,(INPUT),DATASET3,         X
                (INPUT),DATASET4,(OUTPUT))
          TM    DATASET1+DCBQSWS-IHADCB,DCBPOPEN  Opened for
*                                       parallel processing
          BZ    SEQRTN                  Branch on no to
*                                       sequential routine
          TM    DATASET2+DCBQSWS-IHADCB,DCBPOPEN
          BZ    SEQRTN
          TM    DATASET3+DCBQSWS-IHADCB,DCBPOPEN
          BZ    SEQRTN
GETRTN    GET   DCBQUEUE,TYPE=P
          LR    10,1                    Save record pointer
          ...
          ...                           Record updated in place
          ...
          PUT   DATASET4,(10)
          B     GETRTN
EODRTN    L     2,DCBQUEUE+PDADCBEP-IHAPDAB
          L     2,0(0,2)
          CLOSE   ((2))
          CLC   ZEROS(2),DCBQUEUE+PDANODCB-IHAPDAB  Any DCBs left?
          BL    GETRTN                  Branch if yes
          ...
DATASET1  DCB   DDNAME=DDNAME1,DSORG=PS,MACRF=GL,RECFM=FB,          X
                LRECL=80,EODAD=EODRTN,EXLST=SET3XLST
DATASET2  DCB   DDNAME=DDNAME2,DSORG=PS,MACRF=GL,RECFM=FB,          X
                LRECL=80,EODAD=EODRTN,EXLST=SET3XLST
DATASET3  DCB   DDNAME=DDNAME3,DSORG=PS,MACRF=GL,RECFM=FB,          X
                LRECL=80,EODAD=EODRTN,EXLST=SET3XLST
DATASET4  DCB   DDNAME=DDNAME4,DSORG=PS,MACRF=PM,RECFM=FB,          X
                LRECL=80
DCBQUEUE  PDAB  MAXDCB=3
SET3XLST  DC    0F'0',X'92',AL3(DCBQUEUE)
ZEROS     DC    X'0000'
          DCBD  DSORG=QS
          PDABD
          ...
```

Figure 65. Parallel Processing of Three Data Sets

The number of bytes required for PDAB is equal to 24 + 8n, where n is the value of the keyword, MAXDCB.

If data definition statements and data sets are supplied, DATASET1, DATASET2, and DATASET3 are opened for parallel input processing as specified in the input processing OPEN macro. Other attributes of each data set are QSAM (MACRF=G), simple buffering by default, locate or move mode (MACRF=L or M), fixed-length records (RECFM=F), and exit list entry for a PDAB (X'92'). Note that both locate and move modes can be used in the same data set queue. The mapping macros, DCBD and PDABD, are used to refer to the DCBs and the PDAB respectively.

Testing for Parallel Processing

Following the OPEN macro, tests are made to determine whether the DCBs were opened for parallel processing. If not, the sequential processing routine is given control.

In Figure 65 on page 361 when one or more data sets are opened for parallel processing, the GET routine retrieves a record, saves the pointer in register 10, processes the record, and writes it to DATASET4. This process continues until an end-of-data condition is detected on one of the input data sets. The end-of-data routine locates the completed input data set and removes it from the queue with the CLOSE macro. A test is then made to determine whether any data sets remain on the queue. Processing continues in this manner until the queue is empty.

Analyzing I/O Errors

The basic and queued access methods both provide special macros for analyzing I/O errors. These macros can be used in SYNAD routines or in error analysis routines.

SYNADAF—Perform SYNAD Analysis Function

The SYNADAF macro analyzes the status, sense, and exceptional condition code data that is available to your error analysis routine. It produces an error message that your routine can write into any appropriate data set. The message is in the form of unblocked variable-length records, but you can write them as fixed-length records by omitting the block length and record length fields that precede the message texts.

The SYNADAF message can come in two parts, with each message being an unblocked variable-length record. If the data set being analyzed is not a PDSE, extended format data set, or UNIX file; only the first message is filled in. If the data set is a PDSE, extended format data set or UNIX file, both messages are filled in. An 'S' in the last byte of the first message means a second message exists. This second message is located 8 bytes past the end of the first message.

The text of the first message is 120 characters long, and begins with a field of 36, 42, or 20 blanks. You can use the blank field to add your own remarks to the message. The text of the second message is 128 characters long and ends with a field of 76 blanks that are reserved for later use. This second message begins in the fifth byte in the message buffer.

Example: A typical message for a tape data set with the blank field omitted follows:

```
,TESTJOBb,STEP2bbb,283,TA,MASTERbb,READb,DATA CHECKbbbbb,0000015,BSAMb
```

In the preceding example, 'b' means a blank.

That message shows that a data check occurred during reading of the 15th block of a data set being processed with BSAM. The data set was identified by a DD statement named MASTER, and was on a magnetic tape volume on unit 283. The name of the job was TESTJOB; the name of the job step was STEP2.

Example: Two typical messages for a PDSE with the blank fields omitted follow:

```
,PDSEJOBb,STEP2bbb,0283,D,PDSEDDbb,READb,DATA CHECKbbbbb,
00000000100002,BSAMS

,003,000005,0000000002,00000000,00000000,00 ... (76 blanks)
```

That message shows that a data check occurred during reading of a block referred to by a BBCCHHR of X'00000000100002' of a PDSE being processed by BSAM. The data set was identified by a DD statement named PDSEDD, and was on a DASD on unit 283. The name of the job was PDSEJOB. The name of the job step was STEP2. The 'S' following the access method 'BSAM' means that a second message has been filled in. The second message identifies the record in which the error occurred. The concatenation number of the data set is 3 (the third data set in a concatenation), the TTR of the member is X'000005', and the relative record number is 2. The SMS return and reason codes are zero, meaning that no error occurred in SMS.

If the error analysis routine is entered because of an input error, the first 6 or 16 bytes of the first message (at offset 8) contain binary information. If no data was transmitted, or if the access method is QISAM, these first bytes are blanks. If the error did not prevent data transmission, these first bytes contain the address of the input buffer and the number of bytes read. You can use this information to process records from the block. For example, you can print each record after printing the error message. Before printing the message, however, you should replace the binary information with EBCDIC characters.

The SYNADAF macro provides its own save area and makes this area available to your error analysis routine. When used at the entry point of a SYNAD routine, it fulfills the routine's responsibility for providing a save area. See *z/OS DFSMS Macro Instructions for Data Sets* for more information on the SYNADAF macro.

SYNADRLS—Release SYNADAF Message and Save Areas

The SYNADRLS macro releases the message and save areas provided by the SYNADAF macro. You must issue this macro before returning from the error analysis routine.

Device Support Facilities (ICKDSF): Diagnosing I/O Problems

Use Device Support Facilities (ICKDSF) Release 9 or later to determine if there are problems with the disk drive or a problem reading or writing data stored on the volume. Device Support Facilities also performs service checking of a volume. The INSPECT command for the Device Support Facilities program can assign alternate tracks. See *Device Support Facilities User's Guide and Reference*.

Chapter 24. Sharing Non-VSAM Data Sets

This chapter covers the following topics.

Topic	Page
PDSEs	368
Direct Data Sets (BDAM)	368
Factors to Consider When Opening and Closing Data Sets	369
Control of Checkpoint Data Sets on Shared DASD Volumes	369
System Use of Search Direct for Input Operations	371

You can share non-VSAM data sets among:

- Different jobs in a single operating system
- Multiple DCBs in a task or different subtasks
- One DCB in a task or different subtasks
- Different instances of the operating system. To share between different systems safely, you need global resource serialization (GRS) or an equivalent product. Failure to use GRS or an equivalent can result in both data set and VTOC corruption. See *z/OS MVS Planning: Global Resource Serialization* for more information.

There are two conditions under which a data set on a direct access device can be shared by two or more tasks:

- Two or more DCBs are opened and used concurrently by the tasks to refer to the same, shared data set (multiple DCBs).
- Only one DCB is opened and used concurrently by multiple tasks in a single job step (a single, shared DCB).

Except for PDSEs, the system does not protect data integrity when multiple DCBs are open for output and the DCBs access a data set within the same job step. The system ensures that only one program in the sysplex can open a PDS with the OUTPUT option, even if you specify DISP=SHR. If a second program issues OPEN with the OUTPUT option, for the PDS with DISP=SHR, while a DCB is still open with the OUTPUT option, the second program gets a 213-30 ABEND. This does not apply to two programs in one address space with DISP=OLD or MOD, which would cause overlaid data. This 213-30 enforcement mechanism does not apply when you issue OPEN with the UPDAT option. Therefore programs that issue OPEN with UPDAT and DISP=SHR can corrupt the PDS directory. Use DISP=OLD to avoid the possibility of an abend during the processing of a PDS for output or of corrupting the directory when it is open for update.

The DCBE must not be shared by multiple DCBs that are open. After the DCB is successfully closed, you may open a different DCB pointing to the same DCBE.

The operating system provides job control language (JCL) statements and macros that help you ensure the integrity of the data sets you want to share among the tasks that process them. Figure 66 on page 366 and Figure 67 on page 367 show which JCL and macros you should use, depending on the access method your task is using and the mode of access (input, output, or update). Figure 66 describes the

© Copyright IBM Corp. 1987, 2002

processing procedures you should use if more than one DCB has been opened to the shared data set. The DCBs can be used by tasks in the same or different job steps.

The purpose of the RLSE value for the space keyword in the DD statement is to cause CLOSE to free unused space when the data set becomes closed. The system does not perform this function if the DD has DISP=SHR or more than one DCB is open to the data set.

MULTIPLE DCBs

ACCESS MODE	ACCESS METHOD				
	BSAM,BPAM	QSAM	BDAM	QISAM	BISAM
Input	DISP = SHR	DISP = SHR	DISP = SHR	DISP = SHR	DISP = SHR
Output	No facility (not PDSE) DISP = SHR (PDSE)	No facility (not PDSE) DISP = SHR (PDSE)	DISP = SHR	No facility	DISP = SHR and ENQ on Data Set
Update	DISP = SHR user must ENQ on block	DISP = SHR and guarantee discrete blocks	DISP = SHR BDAM will ENQ on block	DISP = SHR and ENQ on data set and guarantee discrete blocks	DISP = SHR and ENQ on data set and guarantee discrete blocks

Figure 66. JCL, Macros, and Procedures Required to Share a Data Set Using Multiple DCBs

DISP=SHR. Each job step sharing an existing data set must code SHR as the subparameter of the DISP parameter on the DD statement for the shared data set to let the steps execute concurrently. For more information about ensuring data set integrity see *z/OS MVS JCL User's Guide*. For more information about sharing PDSEs see "Sharing PDSEs" on page 460. If the tasks are in the same job step, DISP=SHR is not required.

No facility. There are no facilities in the operating system for sharing a data set under these conditions.

ENQ on data set. Besides coding DISP=SHR on the DD statement for the data set that is to be shared, each task must issue ENQ and DEQ macros naming the data set or block as the resource for which exclusive control is required. The ENQ must be issued before the GET (READ); the DEQ macro should be issued after the PUTX or CHECK macro that ends the operation. For more information about using the ENQ and DEQ macros see *z/OS MVS Programming: Assembler Services Reference ABE-HSP*.

Guarantee discrete blocks. When you are using the access methods that provide blocking and unblocking of records (QSAM, QISAM, and BISAM), it is necessary that every task updating the data set ensure that it is not updating a block that contains a record being updated by any other task. There are no facilities in the operating system for ensuring that discrete blocks are being processed by different tasks.

ENQ on block. If you are updating a shared data set (specified by coding DISP=SHR on the DD statement) using BSAM or BPAM, your task and all other tasks must serialize processing of each block of records by issuing an ENQ macro before the READ macro and a DEQ macro after the CHECK macro that follows the WRITE macro you issued to update the record. If you are using BDAM, it provides for enqueuing on a block using the READ exclusive option that is requested by coding MACRF=X in the DCB and an X in the type operand of the READ and WRITE macros. For an example of the use of the BDAM macros see “Exclusive Control for Updating” on page 540.

Figure 67 describes the macros you can use to serialize processing of a shared data set when a single DCB is being shared by several tasks in a job step.

A SINGLE SHARED DCB

ACCESS MODE	ACCESS METHOD				
	BSAM, BPAM, BDAM Create	QSAM	BDAM	QISAM	BISAM
Input	ENQ	ENQ	No action required	ENQ	ENQ
Output	ENQ	ENQ	No action required	ENQ and key sequence	ENQ
Update	ENQ	ENQ	No action	ENQ	ENQ

Figure 67. Macros and Procedures Required to Share a Data Set Using a Single DCB

ENQ. When a data set is being shared by two or more tasks in the same job step (all that use the same DCB), each task processing the data set must issue an ENQ macro on a predefined resource name before issuing the macro or macros that begin the I/O operation. Each task must also release exclusive control by issuing the DEQ macro at the next sequential instruction following the I/O operation. If, however, you are processing an indexed sequential data set sequentially using the SETL and ESETL macros, you must issue the ENQ macro before the SETL macro and the DEQ macro after the ESETL macro. Note also that if two tasks are writing different members of a PDS, each task should issue the ENQ macro before the FIND macro and issue the DEQ macro after the STOW macro that completes processing of the member. See *z/OS MVS Programming: Assembler Services Reference ABE-HSP* for more information about the ENQ and DEQ macros. For an example of the use of ENQ and DEQ macros with BISAM see Figure 125 on page 565.

No action required. See “Sharing DCBs” on page 544.

ENQ on block. When updating a shared direct data set, every task must use the BDAM exclusive control option that is requested by coding MACRF=X in the DCB macro and an X in the type operand of the READ and WRITE macros. See “Exclusive Control for Updating” on page 540 for an example of the use of BDAM macros. Note that all tasks sharing a data set must share subpool 0. See the ATTACH macro description in *z/OS MVS Programming: Assembler Services Reference ABE-HSP*.

Key sequence. Tasks sharing a QISAM load mode DCB must ensure that the records to be written are presented in ascending key sequence; otherwise, a sequence check will result in (1) control being passed to the SYNAD routine identified by the DCB, or (2) if there is no SYNAD routine, termination of the task.

The DISP=SHR specification on the DD statement is not required.

Data sets can also be shared both ways at the same time. More than one DCB can be opened for a shared data set, while more than one task can be sharing one of the DCBs. Under this condition, the serialization techniques specified for indexed sequential and direct data sets in Figure 66 on page 366 satisfy the requirement. For sequential and PDSs, the techniques specified in Figure 66 and Figure 67 must be used.

Open and Close of Data Sets Shared by More than One Task. When more than one task is sharing a data set, the following restrictions must be recognized. Failure to comply with these restrictions endangers the integrity of the shared data set.

- All tasks sharing a DCB must be in the job step that opened the DCB. See “Chapter 24. Sharing Non-VSAM Data Sets” on page 365.
- Any task that shares a DCB and starts any input or output operations using that DCB must ensure that all those operations are complete before terminating the task. A CLOSE macro issued for the DCB ends all input and output operations.
- A DCB can be closed only by the task that opened it.

Shared Direct Access Storage Devices. At some installations, direct access storage devices are shared by two or more independent computing systems. Tasks executed on these systems can share data sets stored on the device. Accessing a shared data set or the same storage area on shared devices by multiple independent systems requires careful planning. Without proper intersystem communication, data integrity could be endangered. See *z/OS MVS Programming: Authorized Assembler Services Guide*.

To ensure data integrity in a shared DASD environment, your system must have global resource serialization (GRS) active or a functionally equivalent global serialization method. See *z/OS MVS Planning: Global Resource Serialization* for details on GRS.

PDSEs

See “Sharing PDSEs” on page 460 for information about sharing PDSEs.

Direct Data Sets (BDAM)

See “Sharing DCBs” on page 544 for more information on sharing direct data sets using BDAM.

Factors to Consider When Opening and Closing Data Sets

Consider the following factors when opening and closing data sets:

- Two or more DCBs should never be concurrently open for output to the same data set, except with the basic indexed sequential access method (BISAM) or with PDSEs. Two or more DCBs can be open concurrently for output to the same PDSE or indexed sequential data set.
- For all data sets except for PDSEs, if, concurrently, one DCB is open for input or update, and one for output to the same data set on direct access storage devices, the input or update DCB might be unable to read what the output DCB wrote if the output DCB extended the data set. For PDSEs, the system dynamically determines that the data set has been extended.
- When an extended format data set is opened for reading, OPEN determines the number of blocks in the data set as of the previous CLOSE for writing. Any data added after the open for reading will not be found unless you supply a DCBE with PASTEOD=YES for the reading DCB. It should be set before reading. For QSAM you must set PASTEOD=YES before completion of the DCB OPEN exit routine to ensure that the system recognizes it.
- If you want to use the same DD statement for two or more DCBs, you cannot specify parameters for fields in the first DCB and then be assured of obtaining the default parameters for the same fields in any other DCB using the same DD statement. This is true for both input and output, and is especially important when you are using more than one access method. Any action on one DCB that alters the JFCB affects the other DCBs and thus can cause unpredictable results. Therefore, unless the parameters of all DCBs using one DD statement are the same, you should use separate DD statements.
- Associated data sets for the IBM 3525 Card Punch can be opened in any order, but all data sets must be opened before any processing can begin. Associated data sets can be closed in any order, but, after a data set has been closed, I/O operations cannot be performed on any of the associated data sets.
- The OPEN macro gets user control blocks and user storage in the protection key in which the OPEN macro is issued. Therefore, any task that processes the DCB (such as Open, Close, or EOV) must be in the same protection key.

Control of Checkpoint Data Sets on Shared DASD Volumes

When an application program has a checkpoint, the system records information about the status of that program in a checkpoint data set. Checkpoint data sets contain system data. In order to ensure integrity of this data, checkpoint data sets are, by default, permitted only on nonshared DASD and tape volumes. If a user could read a checkpoint data set (even one the user owns) then the user might be able to see information the user is not authorized to read. If a user could modify a checkpoint data set (including one the user owns) the user might be able to use it to bypass all security and integrity checks in the system.

Sharing Non-VSAM Data Sets

On systems that assure data set integrity across multiple systems, you may be authorized to create checkpoints on shared DASD through the RACF facility class “IHJ.CHKPT.volser”, where “volser” is the volume serial of the volume to contain the checkpoint data set. Data set integrity across multiple systems is provided when enqueues on the major name “SYSDSN”, minor name “data set name” are treated as global resources (propagated across all systems in the complex) using multisystem global resource serialization (GRS) or an equivalent function.

If a checkpoint data set is on shared DASD, DFSMS issues the SAF RACROUTE macro requesting authorization against a facility class profile of IHJ.CHKPT.volser during checkpoint (“volser” is the volume serial number where the checkpoint data set resides).

If the system programmer cannot insure data set integrity on any shared DASD volumes, the system programmer need not take any further action (for instance, do not define any profile to RACF which would cover IHJ.CHKPT.volser). You cannot take checkpoints on shared DASD volumes.

If data set integrity is assured on all shared DASD volumes and the system programmer wants to let checkpointing on any of these volumes, build a facility class generic profile with a name of IHJ.CHKPT.* with UACC of READ.

If data set integrity cannot be assured on some of the volumes, build discrete profiles for each of these volumes with profile names of IHJ.CHKPT.volser with UACC of NONE. These “volume-specific” profiles are in addition to the generic profiles described above to permit checkpoints on shared DASD volumes for which data set integrity is assured.

If the system programmer wants to let some, but not all, users to create checkpoints on the volumes, build the generic profiles with UACC of NONE and permit READ access only to those specific users or groups of users.

Information in a checkpoint data set includes the location on the disk or tape where the application is currently reading or writing each open data set. If a data set that is open at the time of the checkpoint is moved to another location before the restart, you cannot restart the application from the checkpoint because the location-dependent information recorded by checkpoint/restart is no longer valid.

There are several system functions (for example, DFSMShsm or DFSMSdss) that might automatically move a data set without the owner specifically requesting it. To ensure that all checkpointed data sets remain available for restart, the checkpoint function sets the unmovable attribute for each SMS-managed sequential data set that is open during the checkpoint. An exception is the data set containing the actual recorded checkpoint information (the checkpoint data set), which does not require the unmovable attribute.

You can move checkpointed data sets when you no longer need them to perform a restart. DFSMShsm and DFSMSdss FORCECP(days) enable you to use operations such as migrate, copy, or defrag to move an SMS-managed sequential data set based on a number of days since the last access. DFSMShsm recall, and DFSMSdss restore and copy, are operations that turn off the unmovable attribute for the target data set.

See *z/OS Security Server RACF Command Language Reference* for information about RACF commands and *z/OS Security Server RACF Security Administrator's Guide* for information about using and planning for RACF options.

If you do not have RACF or an equivalent product, the system programmer can write an MVS router exit that is invoked by SAF and can be used to achieve the above functions. See *z/OS MVS Programming: Authorized Assembler Services Guide* for information about writing this exit.

System Use of Search Direct for Input Operations

To hasten the input operations required for a data set on DASD, the operating system uses a technique called search direct in its channel programs. Search direct reads in the requested record and the count field of the next record. This lets the operation get the next record directly, along with the count field of the record that follows it. Search direct (OPTCD=Z) is an obsolete DCB macro and DD statement option. Now the system generally uses this technique for sequential reading.

When sharing data sets, you must consider the restrictions of search direct. Search direct can cause unpredictable results when multiple DCBs are open and the data sets are being shared, and one of the applications is adding records. You might get the wrong record. Also, you might receive unpredictable results if your application has a dependency that is incompatible with the use of search direct.

Chapter 25. Spooling and Scheduling Data Sets

This chapter contains information about the following topics.

Topic	Page
Job Entry Subsystem	373
SYSIN Data Set	374
SYSOUT Data Set	374

Spooling includes two basic functions:

- Input streams are read from the input device and stored on an intermediate storage device in a format convenient for later processing by the system and by the user's program.
- Output streams are similarly stored on an intermediate device until a convenient time for printing, punching, processing by a TSO/E user or sending over a network to another system.

With spooling, unit record devices are used at full speed if enough buffers are available. They are used only for the time needed to read, print, or punch the data. Without spooling, the device is occupied for the entire time it takes the job to process. Also, because data is stored instead of being transmitted directly, output can be queued in any order and scheduled by class and by priority within each class.

Scheduling provides the highest degree of system availability through the orderly use of system resources that are the objects of contention.

Job Entry Subsystem

The job entry subsystem (JES) spools and schedules input and output data streams. It controls all blocking and deblocking of your data to make the best use of system operation. The BSAM NCP value has an effect on the access method, but a value greater than 1 does not improve performance. NCP is supported for compatibility with other data sets. The block size (BLKSIZE) and number of buffers (BUFNO) specified in your program have no relationship with what is actually used by the job entry subsystem. Therefore, you can select the blocking factor that best fits your application program with no effect on the spooling efficiency of the system. For QSAM applications, move mode is as efficient as locate mode.

SYSIN and SYSOUT data sets cannot be system managed. SYSIN and SYSOUT must be either BSAM or QSAM data sets and you open and close them in the same manner as any other data set processed on a unit record device. Because SYSIN and SYSOUT data sets are spooled on intermediate devices, you should avoid using device-dependent macros (such as FEOV, CNTRL, PRTOV, or BSP) in processing these data sets. See "Achieving Device Independence" on page 387. You can use PRTOV, but it will have no effect. For more information about SYSIN and SYSOUT parameters see *z/OS MVS JCL User's Guide* and *z/OS MVS JCL Reference*. Your SYNAD routine is entered if an error occurs during data transmission to or from an intermediate storage device. Again, because the specific device is indeterminate, your SYNAD routine code should be device independent. If

© Copyright IBM Corp. 1987, 2002

you specify the DCB open exit routine in an exit list, it will be entered in the usual manner. See “DCB Exit List” on page 501 for the DCB exit list format and “DCB OPEN Exit” on page 509.

SYSIN Data Set

You enter data into the system input stream by preceding it with a DD * or a DD DATA JCL statement. This is called a SYSIN data set. The DD name is not necessarily SYSIN.

A SYSIN data set cannot be opened by more than one DCB at the same time; that would result in an S013 ABEND.

If no record format is specified for the SYSIN data set, a record format of fixed is supplied. Spanned records (RECFM=VS or VBS) cannot be specified for SYSIN.

The minimum record length for SYSIN is 80 bytes. For undefined records, the entire 80-byte image is treated as a record. Therefore, a read of less than 80 bytes results in the transfer of the entire 80-byte image to the record area specified in the READ macro. For fixed and variable-length records, an ABEND results if the LRECL is less than 80 bytes.

The logical record length value of SYSIN (JFCLRECL field in the JFCB) is filled in with the logical record length value of the input data set. This logical record length value is increased by 4 if the record format is variable (RECFM=V or VB).

The logical record length can be a size other than the size of the input device, if the SYSIN input stream is supplied by an internal reader. JES supplies a value in the JFCLRECL field of the JFCB if that field is found to be zero.

The block size value (the JFCBLKSI field in the JFCB) is filled in with the block size value of the input data set. This block size value is increased by 4 if the record format is variable (RECFM=V or VB). JES supplies a value in the JFCBLKSI field of the JFCB if that field is found to be 0.

SYSOUT Data Set

Your output data can be printed or punched from an output stream that is called the SYSOUT data set. Code the SYSOUT parameter in your DD statement and designate the appropriate output class. For example, SYSOUT=A requests output class A. Your installation establishes the class-device relationship; a list of devices assigned to each output class will enable you to select the appropriate one.

JES permits multiple opens to the same SYSOUT data set, and the records are interspersed. However, you need to ensure that your application serializes the data set. For more information about serialization see “Chapter 24. Sharing Non-VSAM Data Sets” on page 365.

From open to close of a particular data control block you should not change the DCB indicators of the presence or type of control characters. When directed to disk or tape, all the DCB’s for a particular data set should have the same type of control characters. For a SYSOUT data set, the DCBs can have either type of control character or none. The result depends on the ultimate destination of the data set. For local printers and punches, each record is processed according to its control character.

When you use QSAM with fixed-length blocked records or BSAM, the DCB block size parameter does not have to be a multiple of logical record length (LRECL) if the block size is specified in the SYSOUT DD statement. Under these conditions, if block size is greater than, but not a multiple of, LRECL, the block size is reduced to the nearest lower multiple of LRECL when the data set is opened.

You can specify blocking for SYSOUT data sets, even though your LRECL is not known to the system until execution. Therefore, the SYSOUT DD statement of the go step of a compile-load-go procedure can specify a block size without the block size being a multiple of LRECL.

You should omit the DEVD parameter in the DCB macro, or you should code DEVD=DA.

You can use the SETPRT macro to affect the attributes and scheduling of a SYSOUT data set.

Your program is responsible for printing format, pagination, header control, and stacker select. You can supply control characters for SYSOUT data sets in the normal manner by specifying ANSI or machine characters in the DCB. Standard controls are provided by default if they are not explicitly specified. The length of output records must not exceed the allowable maximum length for the ultimate device. Cards can be punched in EBCDIC mode only.

You can supply table reference characters (TRC's) for SYSOUT data sets by specifying OPTCD=J in the DCB. When the data set is printed, if the printer does not support TRC's then the system discards them.

See ″Processing SYSIN, SYSOUT, and System Data Sets″ under “Coding Processing Methods” on page 321.

Chapter 26. Processing Sequential Data Sets

This chapter covers the following topics.

Topic	Page
Creating a Sequential Data Set	377
Retrieving a Sequential Data Set	378
Concatenating Data Sets Sequentially	379
Modifying Sequential Data Sets	386
Achieving Device Independence	387
Improving Performance for Sequential Data Sets	390
Determining the Length of a Block when Reading with BSAM, BPAM, or BDAM	393
Writing a Short Format-FB Block with BSAM or BPAM	394
Using Hiperbatch	395
Processing Non-VSAM Extended-Format Data Sets	395

You must use sequential data sets for all magnetic tape devices, punched cards, and printed output. A data set residing on DASD, regardless of organization, can be processed sequentially. You can also process sequentially data sets created using BISAM or QISAM. Because the entire data set (prime, index, and overflow areas) will be processed, you should determine what type of records are involved.

Creating a Sequential Data Set

Use either the queued access method or the basic sequential access method to store and retrieve the records of a sequential data set. To create a sequential data set on magnetic tape or DASD, take the following actions:

1. Code DSORG=PS or PSU in the DCB macro.
2. Do one of the following:
   - Code a DD statement to describe the data set. See *z/OS MVS JCL Reference*. If SMS is implemented on your system, you can specify a data class in the DD statement or have the ACS routines assign a data class.
   - Create the data set using the TSO or access method services ALLOCATE command. See *z/OS DFSMS Access Method Services*. If SMS is implemented on your system, you can specify the DATACLAS parameter or have the ACS routine assign a data class.
   - Call dynamic allocation (SVC 99) from your program. See *z/OS MVS Programming: Authorized Assembler Services Guide*. If SMS is implemented on your system, you can specify the data class text unit or have the ACS routines assign a data class.
3. Use a data class that is an optional method to simplify and standardize data attributes. You can take advantage of a data class for data sets that are system managed or not system managed.
4. Process the data set with an OPEN macro (the data set is opened for OUTPUT, OUTIN, OUTINX, or EXTEND), a series of PUT or WRITE and CHECK macros, and then a CLOSE macro.

© Copyright IBM Corp. 1987, 2002

Processing a Sequential Data Set

The example in Figure 68 shows that the GET-move and PUT-move require two movements of the data records.

```
         OPEN   (INDATA,,OUTDATA,(OUTPUT))
NEXTREC  GET    INDATA,WORKAREA              Move mode
         AP     NUMBER,=P'1'
         UNPK   COUNT,NUMBER                 Record count adds 6
         OI     COUNT+5,X'F0'                Set zone bits
         PUT    OUTDATA,COUNT                bytes to each record
         B      NEXTREC
ENDJOB   CLOSE  (INDATA,,OUTDATA)
         ...
COUNT    DS     CL6
WORKAREA DS     CL50
NUMBER   DC     PL4'0'
SAVE14   DS     F
INDATA   DCB    DDNAME=INPUTDD,DSORG=PS,MACRF=(GM),EODAD=ENDJOB,       X
               LRECL=50,RECFM=FB
OUTDATA  DCB    DDNAME=OUTPUTDD,DSORG=PS,MACRF=(PM),                  X
               LRECL=56,RECFM=FB
         ...
```

Figure 68. Creating a Sequential Data Set—Move Mode, Simple Buffering

If the record length (LRECL) does not change during processing, but only one move is necessary, you can process the record in the input buffer segment. A GET-locate provides a pointer to the current segment. See “QSAM in an Application” on page 347 for more information.

Retrieving a Sequential Data Set

To retrieve a sequential data set from magnetic tape, direct access storage devices, or other types of devices, take the following actions:

1. Code DSORG=PS or PSU in the DCB macro.
2. Tell the system where your data set is located (by coding a DD statement, or by calling dynamic allocation (TSO ALLOCATE command or SVC 99)).
3. Process the data set with an OPEN macro (data set is opened for input, INOUT, RDBACK, or UPDAT), a series of GET or READ macros, and then a CLOSE macro.

The example in Figure 69 on page 379 is similar to that in Figure 68. However, because there is no change in the record length, the records can be processed in the input buffer. Only one move of each data record is required. See “QSAM in an Application” on page 347 for more information.

```
          ....
          OPEN       (INDATA,,OUTDATA,(OUTPUT),ERRORDCB,(OUTPUT))
NEXTREC   GET        INDATA                 Locate mode
          LR         2,1                    Save pointer
          AP         NUMBER,=P'1'
          UNPK       0(6,2),NUMBER          Process in input area
          PUT        OUTDATA                Locate mode
          MVC        0(50,1),0(2)           Move record to output buffer
          B          NEXTREC
ENDJOB    CLOSE      (INDATA,,OUTDATA,,ERRORDCB)
          ...
NUMBER    DC         PL4'0'
INDATA    DCB        DDNAME=INPUTDD,DSORG=PS,MACRF=(GL),EODAD=ENDJOB
OUTDATA   DCB        DDNAME=OUTPUTDD,DSORG=PS,MACRF=(PL)
ERRORDCB  DCB        DDNAME=SYSOUTDD,DSORG=PS,MACRF=(PM),RECFM=V,         C
                     BLKSIZE=128,LRECL=124
SAVE2     DS         F
          ...
```

Figure 69. Retrieving a Sequential Data Set—Locate Mode, Simple Buffering

Concatenating Data Sets Sequentially

Two or more data sets can be retrieved by the system and processed successively as a single sequential data set. This is called *sequential concatenation*. With sequential concatenation, the application program treats a collection of data sets as one sequential data set. The number of data sets that you can concatenate with sequential concatenation is variable. It is governed by the maximum size of the TIOT option. The system programmer controls the TIOT size with the option ALLOCxx member of SYS1.PARMLIB. See *z/OS MVS Initialization and Tuning Reference*. When data sets are sequentially concatenated, the system is open to only one of the data sets at a time. Concatenated data sets cannot be read backward. During sequential concatenation, the system treats a member of a PDSE or PDS as if it were a sequential data set. The system treats a striped extended format data set as if it were a single-volume data set.

Rule: You cannot concatenate VSAM data sets.

End-of-Data-Set (EODAD) Processing. When the change from one data set to another is made, label exits are taken as required; automatic volume switching is also performed for multiple volume data sets. When your program reads past the end of a data set, control passes to your end-of-data-set (EODAD) routine only if the last data set in the concatenation has been processed.

Consecutive Data Sets on a Tape Volume. To save time when processing two consecutive sequential data sets on a single tape volume, specify LEAVE in your OPEN macro, or DISP=(OLD,PASS) in the DD statement, even if you otherwise would code DISP=(OLD,KEEP).

Concatenating Like Data Sets

Concatenation can be thought of as processing a sequence of *like* or *unlike* data sets. The *like* data sets in the sequence are those that can be processed correctly without notifying the system. All other data sets are treated as *unlike* data sets. For example, you must concatenate data sets with different record formats as *like* unless the data meets the requirements of a different record format. For example, if all the format-V records are the same length, you can specify format-F when reading. If you specify format-U, you can read any format.

System-managed and non-system-managed data sets can be concatenated as *like* data sets. However, you must ensure the concatenated data sets meet all *like* concatenation requirements (see below) or unpredictable results can occur (for example, OPEN ABENDs).

Your program indicates whether the system is to treat the data sets as *like* or *unlike* by setting the bit DCBOFPPC. See “Concatenating Unlike Data Sets” on page 384. The DCB macro assembles this bit as 0, which indicates *like* data sets.

Rules for a Sequential Like Data Set

To be a *like* data set, a sequential data set must meet all the following conditions:

- All the data sets in a concatenation should have compatible record formats. They are all processed with the record format of the first data set in the concatenation (see “Persistence of DCB and DCBE Fields” on page 381). For example a data set with unblocked records can be treated as having short blocked records. A data set with fixed-blocked-standard records (format-FBS) can be treated as having just fixed-blocked records (format-FB), but the reverse cannot work.

  Having compatible record formats does not ensure that *like* processing is successful. For example, if the record format of the first data set in the concatenation is fixed (format-F) and a concatenated data set has fixed-blocked records (format-FB), then unpredictable results, such an I/O errors or open ABENDs, can occur, but the reverse should work.

  The results of concatenating data sets of different characteristics can also depend on the actual data record size and on whether the data sets are system managed. For example, you can process two concatenated data sets successfully if the first data set is format-F with a BLKSIZE and LRECL of 80, the second data set is format-FB with a BLKSIZE of 800 and an LRECL of 80, the second data set is not system managed, and the actual data size of all the blocks in the second data set is 80 bytes. However, if the actual data size of a block is greater than 80 bytes, an I/O error occurs when the system reads that record from the second data set. Alternatively, if SMS manages the second data set, the system processes data from the first data set. An open failure (ABEND 013-60) occurs when EOV switches to the concatenated data set, however, even though the actual data size of all the records can be compatible.

  If incompatible record formats are detected in the concatenation and BSAM is being used, the system issues a warning message, see “BSAM Block Size with Like Concatenation” on page 382.
- LRECL is same as the LRECL of the preceding data set. With format-V or -VB records, the new data set can have a smaller LRECL than is in the DCB.
- All the data set block sizes should be compatible. For format-F or -FB records, each block size should be an integral multiple of the same LRECL value.
- If you code the BLKSIZE parameter in the DCB or DCBE macro, or on the first DD statement, the block size of each data set must be less than or equal to that block size.

  Note: If you specify DCB parameters such as BLKSIZE, LRECL, or BUFNO when allocating a data set after the first one, they have no effect when your program uses *like* concatenation, except as described in “BSAM Block Size with Like Concatenation” on page 382.

  You can specify a large BLKSIZE for the first data set to accommodate a later data set with blocks of that size.
- DASD data sets that are accessed by QSAM or BSAM can be concatenated in any order of block size. If you are using QSAM, you must use system-created buffers for the data set. The size of each system-created buffer equals the block sizes rounded up to a multiple of 8. For QSAM the system-created buffers are

used to process all data sets in the concatenation unless the next data set's BLKSIZE is larger than the buffers. In that case, the buffers are freed by end-of-volume processing and new system-created buffers are obtained. This also means the buffer address returned by GET is only guaranteed valid until the next GET or FEOV macro is issued, because the buffer pool can have been freed and a new system-created buffer pool obtained during end-of-volume concatenation processing. For system-managed data set processing, see "SMS-Managed Data Sets with Like Concatenation" on page 382. For BSAM processing see "BSAM Block Size with Like Concatenation" on page 382.

- For QSAM, if a data set after the first one is on magnetic tape and has a block size larger than all prior specifications, the volume must have IBM or ISO/ANSI standard tape labels or the BLKSIZE must be specified on the DD statement.
- For BSAM, if a data set after the first one is on magnetic tape and has a block size larger than all prior specifications, the BLKSIZE must be specified on the DD statement.
- The device is a direct access storage device, tape, or SYSIN device, as is the device of the preceding data set. Either one, or both DDs can be for a USS file. For example, you can concatenate a tape data set to a DASD data set, or a DASD data set to a tape data set. However, you cannot concatenate a tape data set to card reader. For USS file concatenation information, see "Sequential Concatenation" on page 426.

  Note: An extended format data set can be regarded as having the same characteristics as a physical sequential data set. See "Concatenating Extended-Format Data Sets with Other Data Sets" on page 399.
- If mixed tape and DASD, the POINT or CNTRL macros are not used—neither P nor C was coded in the DCB MACRF parameter.

OPEN/EOV Exit Processing

If the program has a DCB OPEN exit, it is called only at the beginning of the first data set.

With *like* concatenation, if the program has an end-of-volume exit, it is called at the beginning of each volume of each data set except the first volume of the first data set. If the type of data set does not have volumes, the system treats it as having one volume.

Persistence of DCB and DCBE Fields

Between the completion of OPEN and the completion of CLOSE with *like* concatenation, the system can change certain DCB or DCBE fields that represent data attributes. Your program and the system do not change the following attribute fields:

- RECFM
- LRECL for format-F for BSAM and for QSAM XLRI
- BLKSIZE for BSAM (your program can change this)
- KEYLEN
- NCP or BUFNO

With *like* concatenation the system can change the following when switching to another data set:

- BLKSIZE and BUFL for QSAM
- Field DCBDEVT in the DCB (device type)
- TRTCH (tape recording technique)
- DEN (tape density)

With or without concatenation the system sets LRECL in the DCB for each QSAM GET macro when reading format-V, format-D, or format-U records, except with XLRI. GET issues an ABEND if it encounters a record that is longer than LRECL was at the completion of OPEN.

If your program indicates *like* concatenation (by taking no special action about DCBOFPPC) and one of the *like* concatenation rules is broken, the results are unpredictable. A typical result is an I/O error, resulting in an ABEND, or entry to the SYNAD routine. The program might even appear to run correctly.

SMS-Managed Data Sets with Like Concatenation

If SMS-managed data sets are being concatenated, then the system does additional processing for the transition between data sets. This includes additional checking of data set attributes. This might result in OPEN issuing an ABEND after successful completion of the OPEN that the user program issued. A violation of the *like* concatenation requirements could result in an ABEND during the open of the next concatenated data set.

If the open routine for QSAM obtains the buffer pool automatically, the data set transition process might free the buffer pool and obtain a new one for the next concatenated data set. The buffer address that GET returns is valid only until the next GET or FEOV macro runs. The transition process frees the buffer pool and obtains a new, system-created buffer pool during end-of-volume concatenation processing. The procedure does not free the buffer pool for the last concatenated data set unless you coded RMODE31=BUFF. You should also free the system-created buffer pool before you attempt to reopen the DCB, unless you coded RMODE31=BUFF.

BSAM Block Size with Like Concatenation

After BSAM OPEN has merged the data set characteristics from the label to the JFCB and the DCB or DCBE, and before it calls your DCB OPEN exit routine, OPEN tries to search later DD statements to see if BSAM should use a larger maximum block size. OPEN searches only if you have enabled a larger block size. A larger block size is enabled if all three of the following conditions are true:

- BLKSIZE is not coded in the DCB or DCBE before OPEN or in the first JFCB. Each data set is represented by a JFCB.
- RECFM (record format) in the DCB specifies format-U or blocked records. Any data set can be fixed-standard blocked.
- You did not set on DCBOFPPC (X'08' in DCBOFLGS, which is at +48 in the DCB). This is the *unlike* attributes bit.

If you have enabled a larger block size, OPEN searches later concatenated data sets for the largest acceptable block size and stores it in the DCB or DCBE. A block size is acceptable if it comes from a source that does not also have a RECFM or LRECL inconsistent with the RECFM or LRECL already in the DCB.

Compatible Characteristics: For format-F records, if a data set has an LRECL value that differs from the value in the DCB, the block size for that data set is not considered during OPEN.

For format-V records, if a data set has an LRECL value that is larger than the value in the DCB, the block size for that data set is not considered during OPEN.

A RECFM value of U in the DCB is consistent with any other RECFM value.

BSAM considers the following RECFM values compatible with the specified record format for the first data set:

- F or FB—Compatible record formats are F, FB, FS, and FBS.
- V or VB—Compatible record formats are V and VB.
- U—All other record formats are compatible.

If OPEN finds an inconsistent RECFM, it will issue a warning message. OPEN does not examine DSORG when testing consistency. It does not issue ABEND since you might not read as far as that data set or you might later turn on the DCB *unlike* attributes bit.

Even though RECFMs of concatenated data sets can be considered compatible by BSAM (and you do not receive the expected warning message) that does not guarantee they can be successfully processed. It still can be necessary to treat them as *unlike*.

BSAM OPEN Processing Before First Data Set: OPEN tests the JFCB for each data set after the one being opened. The JFCB contains information coded when the data set was allocated and information that OPEN can have stored there before it was dynamically reconcatenated.

All of the above processing previously described occurs for any data set that is acceptable to BSAM. The OPEN that you issue does not read tape labels for data sets after the first. Therefore, if there is a tape data set after the first that has a block size larger than all of the prior specifications, the BLKSIZE value must be specified on the DD statement. The system later reads those tape labels but it is too late for the system to discover a larger block size at that time.

For each data set whose JFCB contains a block size of 0 and is on permanently resident DASD, OPEN obtains the data set characteristics from the data set label (DSCB). If they are acceptable and the block size is larger, OPEN copies the block size to the DCB or DCBE.

For each JFCB or DSCB that this function of OPEN examines, OPEN turns off the DCB's standard bit, if the block size differs from the DCB or DCBE block size and the DCB has fixed standard.

If DCBBUFL, either from the DCB macro or the first DD statement, is nonzero, then that value will be an upper limit for BLKSIZE from another data set. No block size from a later DD statement or DSCB is used during OPEN if it is larger than that DCBBUFL value. OPEN ignores that larger block size on the assumption that you will turn on the *unlike* attributes bit later, will not read to that data set, or the data set does not actually have blocks that large.

When OPEN finds an inconsistent record format, it issues the following message:

```
IEC034I INCONSISTENT RECORD FORMATS rrr AND iii,ddname+cccc,dsname
```

In the message, the variables represent the following values.

rrr	Specifies record format established at OPEN.
iii	Specifies record format found to be inconsistent. It is in a JFCB that has a nonzero BLKSIZE or in a DSCB.
cccc	Specifies the number of the DD statement after the first one, where +1 means the second data set in the concatenation.

Example of BSAM Like Concatenation: Figure 70 shows an example of JCL for a *like* concatenation that is read using BSAM. The application could use QSAM instead of BSAM. QSAM does not require BLKSIZE to be coded because this tape data set on 3590 has tape labels.

```
//INPUT DD  *
        ... (instream data set)
//      DD  DSN=D42.MAIN.DATA,DISP=SHR
//      DD  DSN=D42.SUPPL.DATA,UNIT=(3590,2),DISP=OLD,BLKSIZE=150000
```

Figure 70. Like Concatenation Read through BSAM

This example requires the application to use the large block interface because the BLKSIZE value is so large.

OPEN finds that the block size value for the second DD is larger than for the first DD, which normally is 80. If the second DD is for a disk data set, its maximum block size is 32 760. BSAM OPEN for the first DD uses the BLKSIZE from the third DD because it is the largest.

Concatenating Unlike Data Sets

To concatenate *unlike* sequential data sets, you must modify the DCBOFLGS field of the DCB before the end of the current data set is reached. This informs the system that you are concatenating *unlike* data sets.

DCBOFPPC is bit 4 of the DCBOFLGS field. Set bit 4, DCBOFPPC, to 1 by using the instruction OI DCBOFLGS,X'08'. If DCBOFPPC is 1, end-of-volume processing for each data set issues a close for the data set just read, and an open for the next concatenated data set. This closing and opening procedure updates the fields in the DCB and, performs the other functions of CLOSE and OPEN. If the buffer pool was obtained automatically by the open routine, the procedure also frees the buffer pool and obtains a new one for the next concatenated data set. The procedure does not free the buffer pool for the last concatenated data set unless your program supplied a DCBE with RMODE31=BUFF.

Unless you have some way of determining the characteristics of the next data set before it is opened, you should not reset the DCBOFLGS field to indicate *like* attributes during processing. When you concatenate data sets with *unlike* attributes (that is, turn on the DCBOFPPC bit of the DCBOFLGS field), the EOV exit is not taken for the first volume of any data set. If the program has a DCB OPEN exit it is called at the beginning of every data set in the concatenation.

If your program turns DCBOFPPC on before issuing OPEN, each time the system calls your DCB OPEN exit routine or JFCBE exit, DCBESLBI in your DCBE is on only if the current data set being started supports large block interface (LBI). If you want to know in advance if all the data sets support LBI, your program can take one of the following actions:

- Leave DCBOFPPC off until after OPEN. You do not need it on until your program attempts to read a record.
- Issue the DEVTYPE macro with INFO=AMCAP. See *z/OS DFSMSdfp Advanced Services*.

When a new data set is reached and DCBOFPPC is on, you must reissue the GET or READ macro that detected the end of the data set because with QSAM, the new data set can have a longer record length, or with BSAM the new data set can have a larger block size. You might need to allocate larger buffers. Figure 71 shows a possible routine for determining when a GET or READ must be reissued.

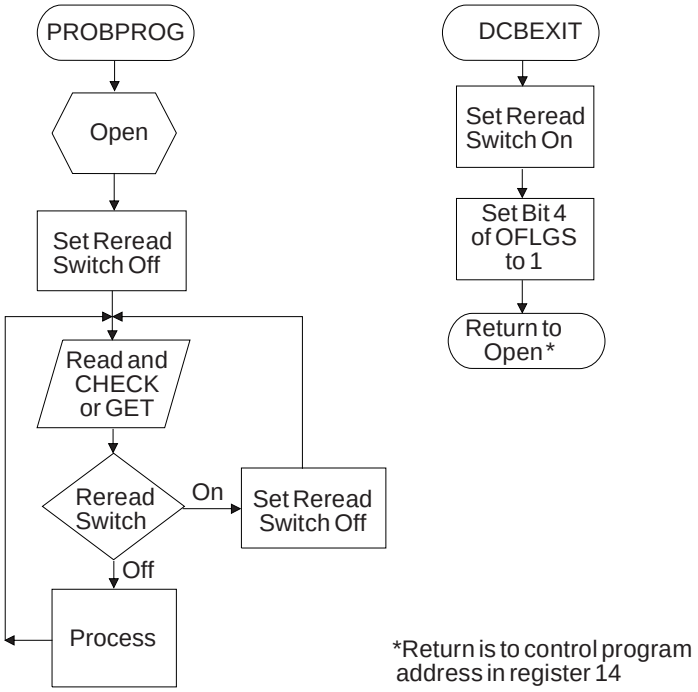


Figure 71. Reissuing a READ or GET for Unlike Concatenated Data Sets

You might need to take special precautions if the program issues multiple READ macros without intervening CHECK or WAIT macros for those READS. Do not issue WAIT or CHECK macros to READ requests that were issued after the READ that detected end-of-data. These restrictions do not apply to data set to data set transition of *like* data sets, because no OPEN or CLOSE operation is necessary between data sets.

You can code OPTCD=B in the DD statement, or you can code it for dynamic allocation. You cannot code OPTCD=B in the DCB macro. This parameter has an effect only during the reading of IBM, ISO, or ANSI standard labelled tapes. In those cases, it causes the system to treat the portion of the data set on each tape volume as a complete data set.

In this way, you can read tapes in which the trailer labels incorrectly are end-of-data instead of end-of-volume.

If you specify OPTCD=B in the DD statement for a multivolume tape data set, the system generates the equivalent of individual concatenated DD statements for each volume serial number and allocates one tape drive for each volume.

Restriction: If you have a variable-blocked spanned (VBS) data set that spans volumes in such a way that one segment (for example, the first segment) is at the end of the first volume and the next segment (for example, the middle segment) is at the beginning of the next volume, and you attempt to treat these volumes as separate data sets, the integrity of the data cannot be guaranteed. QSAM will

abend. QSAM's job is to ensure that it can put all of the segments together. This restriction will also be based on the data and whether or not the segments are split up between volumes.

Modifying Sequential Data Sets

You can modify a sequential data set in three ways:

- By changing the data in existing records (update-in-place).
- By adding new records to the end of a data set (extends the data set).
- Or, by opening for OUTPUT or OUTIN without DISP=MOD (replaces the data set's contents). The effect is the same as when creating the data set. See "Creating a Sequential Data Set" on page 377.

Updating in Place

When you update a data set in place, you read, process, and write records back to their original positions without destroying the remaining records on the track. The following rules apply:

- You must specify the UPDAT option in the OPEN macro to update the data set. To perform the update, you can use only the READ, WRITE, CHECK, NOTE, and POINT macros or you use only GET and PUTX macros. To use PUTX, code MACRF=(GL,PL) on the DCB macro.
- You cannot delete any record or change its length.
- You cannot add new records.
- The data set must be on a direct access storage device.
- You must rewrite blocks in the same order in which you read them.

A record must be retrieved by a READ or GET macro before it can be updated by a WRITE or PUTX macro. A WRITE or PUTX macro does not need to be issued after each READ or GET macro.

The READ and WRITE macros must be execute forms that refer to the same data event control block (DECB). The DECB must be provided by the list forms of the READ or WRITE macros. See *z/OS DFSMS Macro Instructions for Data Sets* for information about the execute and list forms of the READ and WRITE macros.

Restriction: You cannot use the UPDAT option to open a compressed-format data set, so an update-in-place is not allowed on it.

Using Overlapped Operations

To overlap I/O and processor activity, you can start several BSAM read or write operations before checking the first operation for completion. You cannot overlap read with write operations, however, because operations of one type must be checked for completion before operations of the other type are started or resumed. Note that each pending read or write operation requires a separate DECB. If a single DECB were used for successive read operations, only the last record read could be updated.

See Figure 85 on page 425 for an example of an overlap achieved by having a read or write request outstanding while each record is being processed.

Extending a Data Set

If you want to add records at the end of your data set, you must open the data set for output with DISP=MOD specified in the DD statement, or specify the EXTEND or OUTINX option of the OPEN macro. You can then issue PUT or WRITE macros to the data set.

Multivolume DASD Data Set

If all of the following are true, CLOSE sets on a last-volume indicator in the data set label for the volume containing the last user data block:

- DCB opened for OUTPUT, EXTEND, OUTIN, OUTINX, or INOUT
- Most recent operation was PUT, a CHECK for a WRITE, or (except for OPEN INOUT) an OPEN.
- DCB closed successfully

The system ensures that the data set labels on prior volumes do not have the last-volume indicator on. The volume with the last-volume bit on is not necessarily the last volume that contains space for the data set or is indicated in the catalog. A later volume might also have the last volume bit on.

When you later extend the data set with DISP=MOD or OPEN with EXTEND or OUTINX, OPEN must determine the volume containing the last user data.

With a system-managed data set, OPEN tests each volume from the first to the last until it finds the last-used volume.

With a non-system-managed data set, the system follows a different procedure. First OPEN tests the data set label on the last volume identified in the JFCB or JFCB extension. If the last-volume indicator is on, OPEN assumes it to be the last-used volume. If the indicator is not on, OPEN searches the volumes from the first to the second to last. It stops when it finds a data set label with the last-volume indicator on. This algorithm is for compatibility with older MVS levels that supported mountable DASD volumes. If this algorithm is unacceptable, you can either delete the data set from all volumes or delete the data set from the volume that has the last-volume indicator on.

Extended-Format Data Sets

Extended-format data sets allocated with more than one stripe cannot be extended to another set of volumes. These data sets cannot become multivolume data sets. The term multivolume should not be used for a multistriped data set because volumes that comprise multiple stripes of a data set appear to the user as one logical volume.

An extended format data set allocated with a single stripe may be extended to additional volumes. They can become multivolume data sets. All the rules that apply to physical sequential multivolume data sets also apply to single-striped multivolume extended format data sets.

Achieving Device Independence

Device independence is the characteristic of programs that work on any type of device, DASD or tape, for example. Achieving device independence is possible only for a sequential data set because input or output can be on DASD, a magnetic tape drive, a card reader or card punch, a printer, a spooled data set, a TSO/E terminal,

or a dummy data set. Other data set organizations (partitioned, indexed sequential, direct, and VSAM) are device-dependent because they require the use of direct access storage devices.

A dummy data set is a DD statement on which the first parameter is DUMMY or you coded the DDNAME= parameter but there is no DD statement by that name. You can use BSAM or QSAM with a dummy data set. A WRITE or PUT macro has no effect. A GET macro or the CHECK macro for the first READ macro causes your EODAD routine to be called.

Device independence can be useful for the following tasks:

- Accepting data from several recording devices, such as a disk volume, magnetic tape, or unit-record equipment. This situation could arise when several types of data acquisition devices are feeding a centralized installation.
- Bypassing restrictions imposed by the unavailability of I/O devices (for example, when devices on order have not been installed).
- Assembling, testing, and debugging on one system or device type and processing on a different one. For example, an IBM 3380 Direct Access Storage drive can be used as a substitute for a magnetic tape unit.
- Testing TSO commands such as REXX execs in the TSO/E background.

To make your program device independent, take the following actions:

- Omit all device-dependent macros and parameters from your program. For maximum efficiency it is best to omit the BLKSIZE parameter with a BSAM, BPAM or QSAM DCB. See “System-Determined Block Size” on page 324.
- Supply the parameters on your data definition (DD) statement, data class, or during the OPEN exit routine. That is, do not specify any required device-dependent parameters until the program is ready for execution. Your program can learn many of the device characteristics by issuing the DEVTYPE macro.

Device-Dependent Macros

The following is a list of device-dependent macros and macro parameters. Consider only the logical layout of your data record without regard for the type of device used. Even if your data is on a direct access volume, treat it as if it were on a magnetic tape. For example, when updating, you must create a new data set rather than attempt to update the existing data set.

OPEN—Specify INPUT, OUTPUT, INOUT, OUTIN, OUTINX, or EXTEND. The parameters RDBACK and UPDAT are device-dependent and can cause an abnormal end if directed to the wrong device type or to a compressed format data set on DASD.

READ—Specify forward reading (SF) only.

WRITE—Specify forward writing (SF) only; use only to create new records or modify existing records.

NOTE/POINT—These macros are valid for both magnetic tape and direct access volumes. To maintain independence of the device type and of the type of data set (physical sequential, extended format, PDSE, and so forth), do not test or modify the word returned by NOTE or calculate a word to pass to POINT.

BSP—This macro is valid for magnetic tape or direct access volumes. However, its use would be an attempt to perform device-dependent action.

SETPRT—Valid only for directly allocated printers and for SYSOUT data sets.

CNTRL/PRTOV—These are device-dependent macros.

DCB and DCBE Subparameters

Coding MODE, CODE, TRTCH, KEYLEN, or PRTSP in the DCB macro makes the program device-dependent. However, they can be specified in the DD statement.

DEVD —Specify DA if any direct access storage devices might be used. Magnetic tape and unit-record equipment DCBs will fit in the area provided during assembly. Specify unit-record devices only if you expect never to change to tape or direct access storage devices.

KEYLEN—Can be specified on the DD statement or in the data class if necessary.

RECFM, LRECL, BLKSIZE—These parameters can be specified in the DD statement, data class, or data set label. However, you must consider maximum record size for specific devices. Also, you must consider if you want to process XLRI records.

DSORG—Specify sequential organization (PS or PSU) to get the full DCB expansion.

OPTCD—This parameter is device dependent; specify it in the DD statement.

SYNAD—Any device-dependent error checking is automatic. Generalize your routine so that no device-dependent information is required.

Improving Performance for Sequential Data Sets

To make the I/O operations required for a data set faster, the operating system provides a technique called *chained scheduling*. Chained scheduling is not a DASD option; it is built into the access method for DASD. When chained scheduling is used, the system dynamically chains several I/O operations together. A series of separate read or write operations, functioning with chained scheduling, is issued to the computing system as one continuous operation.

The I/O performance is improved by reducing both the processor time and the channel start/stop time required to transfer data to or from virtual storage. Some factors that affect performance follow:

- Address space type (real or virtual)
- Block size. Larger blocks are more efficient. You can get significant performance improvement by using LBI, large block interface. It allows tape blocks longer than 32 760 bytes.
- BUFNO for QSAM
- The number of overlapped requests for BSAM (NCP=number of channel programs) and whether the DCB points to a DCBE that has MULTACC coded
- Other activity on the processor and channel
- Device class (for example, DASD, tape) and type (for example, IBM 3390, 3490)
- Data set type (for example, PDSE, HFS, extended format)
- Number of stripes if extended format.

The system defaults to chained scheduling for nondirect access storage devices, except for printers and format-U records, and for those cases in which it is not permitted.

Chained scheduling is most valuable for programs that require extensive input and output operations. Because a data set using chained scheduling can monopolize available time on a channel in a V=R region, separate channels should be assigned, if possible, when more than one data set is to be processed.

Limitations on Using Chained Scheduling with Non-DASD Data Sets

The following are limitations on using chained scheduling:

- Each data set for which chained scheduling is used must be assigned at least two (and preferably more) buffers with QSAM, or must have a value of at least two (and preferably more) for the NCP parameter with BSAM.
- A request for exchange buffering is not honored, but defaults to move mode and, therefore, has no effect on either a request for chained scheduling or a default to chained scheduling. Exchange buffering is an obsolete DCB option.
- A request for chained scheduling is ignored and normal scheduling used if any of the following are met when the data set is opened:
  - CNTRL macro is to be used.
  - Embedded VSE checkpoint records on tape input are bypassed (OPTCD=H).
  - Data set is not magnetic tape or unit record.
  - NCP=1 with BSAM or BUFNO=1 with QSAM.
  - It is a print data set, or any associated data set for the 3525 Card Punch.

- The number of channel program segments that the system can chain together is limited to the value specified in the NCP parameter of BSAM DCBs, and to the value specified in the BUFNO parameter of QSAM DCBs.
- When the data set is a printer, chained scheduling is not supported when channel 9 or channel 12 is in the carriage control tape or FCB.
- When chained scheduling is used, the automatic skip feature of the PRTOV macro for the printer will not function. Format control must be achieved by ANSI or machine control characters. See “Using Optional Control Characters” on page 308 and *z/OS DFSMS Macro Instructions for Data Sets* for more information about control characters.
- When you are using QSAM under chained scheduling to read variable-length, blocked, ASCII tape records (format-DB), you must code BUFOFF=L in the DCB for that data set.
- If you are using BSAM with the chained scheduling option to read format-DB records, and have coded a value for the BUFOFF parameter other than BUFOFF=L, the input buffers are converted from ASCII to EBCDIC for Version 3 (or to the specified character set (CCSID) for Version 4) as usual, but the record length returned to the DCBLRECL field equals the maximum block size for the data set, not the actual length of the block read in. Each record descriptor word (RDW), if present, is not converted from ASCII to binary.

DASD and Tape Performance

For direct access storage devices (DASD), the DCB OPTCD=C option has no effect. It requests chained scheduling but the access method automatically uses equivalent techniques.

In QSAM, the value of BUFNO determines how many buffers will be chained together before I/O is initiated. The default value of BUFNO is described in “Constructing a Buffer Pool Automatically” on page 344. When enough buffers are available for reading ahead or writing behind, QSAM attempts to read or write those buffers in successive revolutions of the disk.

In BSAM and BPAM, the first READ or WRITE instruction initiates I/O unless the system is honoring your MULTACC specification in the DCBE macro. The system puts subsequent I/O requests (without an associated CHECK or WAIT instruction) in a queue. When the first I/O request completes normally, the system checks the queue for pending I/O requests and builds a channel program for as many of these requests as possible. The number of I/O requests that the system can chain together is the maximum number of requests that the system can process in one I/O event. This limit is less than or equal to the NCP value.

For better performance with BSAM and BPAM, use the technique described in “Using Overlapped I/O with BSAM” on page 353 and Figure 84 on page 422.

Recommendation: Use the MULTACC and MULTSDN parameters in the DCBE macro for maximum performance with BSAM and BPAM.

Processing a Sequential Data Set

For BSAM sequential and PDSs, specifying a nonzero MULTACC value on a DCBE macro can result in more efficient channel programs. You can also code a nonzero MULTSDN value. If MULTSDN is nonzero and DCBNCP is zero, OPEN determines a value for NCP and stores that value in DCBNCP before giving control to the DCB open exit. If MULTACC is nonzero and your program uses the WAIT or EVENTS macro on a DECB or depends on a POST exit for a DECB, then you must precede that macro or dependence by a CHECK or TRUNC macro.

Notes:

1. For compressed format data sets, MULTACC is ignored since all buffering is handled internally by the system.
2. For tape data sets using large block interface (LBI) that have a block size greater than 32 768, the system-determined NCP value is between 2 and 16. If the calculated value is <2, it is set to 2, and if it is >16, it is set to 16.

Determining the Length of a Block when Reading with BSAM, BPAM, or BDAM

When you read a sequential data set, you can determine the length of the block in one of the following ways, depending on the access method and record format of the data set.

For unblocked and undefined record formats, each block contains one logical record.

1. **Fixed-length, unblocked records:** The length of all records is the value in the DCBBLKSI field of the DCB without LBI or the DCBEBLKSI field of the DCBE with LBI. You can use this method with BSAM or BPAM.
2. **Variable-length records and Format-D records with BUFOFF=L:** The block descriptor word in the block contains the length of the block. You can use this method with BSAM or BPAM. “Block Descriptor Word (BDW)” on page 293 describes the BDW format.
3. **Format-D records without BUFOFF=L:** The block length is in DCBLRECL after you issue the CHECK macro. It remains valid until you again issue a CHECK macro.
4. **Undefined-length records when using LBI or for fixed-length blocked:** The method described in the following paragraphs can be used to calculate the block length. You can use this method with BSAM, BPAM, or BDAM. (It should not be used when using chained scheduling with format-U records. In that case, the length of a record cannot be determined.
   a. After issuing the CHECK macro for the DECB for the READ request, but before issuing any subsequent data management macros that specify the DCB for the READ request, obtain the status area address in the word that is 16 bytes from the start of the DECB.
   b. If you are not using LBI, take the following steps:
      1) Obtain the residual count that has been stored in the status area. The residual count is in the halfword, 14 bytes from the start of the status area.
      2) Subtract this residual count from the number of data bytes requested to be read by the READ macro. If 'S' was coded as the length parameter of the READ macro, the number of bytes requested is the value of DCBBLKSI at the time the READ was issued. If the length was coded in the READ macro, this value is the number of data bytes and it is contained in the halfword 6 bytes from the beginning of the DECB. The result of the subtraction is the length of the block read.

      If you are using LBI for BSAM or BPAM, subtract 12 from the address of the status area. This gives the address of the 4 bytes that contain the length of the block read.
5. **Undefined-length records when not using LBI:** The actual length of the record that was read is returned in the DCBLRECL field of the DCB. Because of this use of DCBLRECL, you should omit LRECL. Use this method only with BSAM, or BPAM or after issuing a QSAM GET macro.

Figure 72 on page 394 shows an example of determining the length of a record when using BSAM to read undefined-length records.

```
         ...
         OPEN      (DCB,(INPUT))
         LA        DCBR,DCB
         USING     IHADCB,DCBR
         ...
         READ      DECB1,SF,DCB,AREA1,'S'
         READ      DECB2,SF,DCB,AREA2,50
         ...
         CHECK     DECB1
         LH        WORK1,DCBBLKSI              Block size at time of READ
         L         WORK2,DECB1+16              Status area address
         SH        WORK1,14(WORK2)             WORK1 has block length
         ...
         CHECK     DECB2
         LH        WORK1,DECB2+6               Length requested
         L         WORK2,DECB2+16              Status area address
         SH        WORK1,14(WORK2)             WORK1 has block length
         ...
         MVC       DCBBLKSI,LENGTH3            Length to be read
         READ      DECB3,SF,DCB,AREA3
         ...
         CHECK     DECB3
         LH        WORK1,LENGTH3               Block size at time of READ
         L         WORK2,DECB+16               Status area address
         SH        WORK1,14(WORK2)             WORK1 has block length
         ...
DCB      DCB       ...RECFM=U,NCP=2,...
         DCBD
         ...
```

Figure 72. One Method of Determining the Length of a Record when Using BSAM to Read Undefined-Length or Blocked Records

When you write a short block to an extended format data set, the system pads it to full length but retains the value of what your program said is the length. When you read such a block, be aware that the system reads as many bytes as the block can have and is not limited by the length specified for the write. If you know that a particular block is short and you plan to read it to a short data area, then you must decrease DCBBLKSI or DCBEBLKSI with LBI to the length of the short area before the READ.

Writing a Short Format-FB Block with BSAM or BPAM

If you have fixed-blocked record format, you can change the length of a block when you are writing blocks for a sequential data set. You can change the DCB block size field (DCBBLKSI, without LBI) or DCBE block size field (DCBEBLKSI, with LBI) to specify a block size that is shorter than what was originally specified for the data set. You should not, however, change that field to specify a block size that is greater than what was originally specified.

You change block size in the DCB or DCBE before issuing the WRITE macro. It must be a multiple of the LRECL parameter in the DCB. After this is done, any subsequent WRITE macros issued write records with the new block length until you change the block size again.

This technique works for all data sets supported by BSAM or BPAM. With extended format data sets, the system actually writes all blocks in the data set as the same size, but on a READ returns the length specified on the WRITE for the block.

Recommendation: You can create short blocks for PDSEs but their block boundaries are not saved when the data set is written to DASD. Therefore, if your program is dependent on short blocks, do not use a PDSE. See “Processing PDSE Records” on page 434 for information about using short blocks with PDSEs.

Using Hiperbatch

Hiperbatch is an extension of QSAM designed to improve performance in specific situations. Hiperbatch uses the data lookaside facility (DLF) services to provide an alternate fast path method of making data available to many batch jobs. Through Hiperbatch, applications can take advantage of the performance benefits of the operating system without changing existing application programs or the JCL used to run them. See *MVS Hiperbatch Guide* for information about using Hiperbatch. See *z/OS MVS System Commands* for information about the DLF commands.

Either Hiperbatch or extended format data sets can improve performance, but they cannot be used for the same data set.

Hiperbatch	Striping
Uses Hiperspace	Requires certain hardware
Improved performance requires multiple reading programs at the same time	Performance is best with only one program at a time
Relatively few data sets in the system can use it at once	Larger number of data sets can be used at once
QSAM only	QSAM and BSAM

Processing Non-VSAM Extended-Format Data Sets

Extended-format data sets, for most purposes, have the same characteristics as physical sequential data sets. Records are not necessarily stored in the same format or order as they appear. An extended format data set can be referred to as a striped data set if its data are interleaved across multiple volumes. This is called sequential data striping.

Large data sets with high I/O activity are the best candidates for striped data sets. Data sets defined as extended format sequential must be accessed by means of BSAM or QSAM and not EXCP or BDAM.

Processing a Sequential Data Set

The following are characteristics of extended format data sets:

- In most cases, sequential data striping does not require any changes to existing JCL. Your installation's ACS routines must request a data class which specifies data set name type of EXTENDED.
- In most cases, no changes to applications are needed to access extended format data sets.
- The following types of data sets cannot be allocated as extended format data sets:
  - Nonsequential data sets except VSAM
  - Non-system-managed data sets
  - VIO data sets.
- The following types of data sets should not be allocated as extended format data sets:
  - System data sets
  - GTF trace
  - DFSORT™ work data sets
  - Data sets used with Hiperbatch
  - Data sets accessed with EXCP
  - Data sets used with checkpoint/restart

  Extended format non-VSAM data sets are only supported for BSAM and QSAM. When extended format data sets were invented, dump processing used EXCP to produce dumps, and IPCS used BDAM to read dumps.
- Sequential data striping is exclusive to system-managed storage. Extended format data sets can be allocated only on system-managed volumes. An extended format data set is not allocated in a storage group containing mixed device types.
- Extended format data sets can be allocated only on certain devices.
- Extended format data sets have a maximum of 123 extents on each volume. (Physical sequential data sets have a maximum of 16 extents on each volume.)
- On a volume that has more than 65,535 tracks, a physical sequential data set cannot occupy more than 65,535 tracks. An extended format data set can occupy any number of tracks.
- Extended format data sets provide improved data integrity by detecting control unit padding. This type of data padding can occur when the processor loses electrical power while writing a block, it can be due to an operator CANCEL command, a time-out, or it can occur as a result of an ABEND when PURGE=QUIESCE was not specified on the active ESTAE macro. On input, SAM provides an I/O error instead of returning bad data when it detects an error due to control unit padding.
- The system can detect an empty extended format data set. If an extended format data set is opened for input and that data set has never been written to, the first read detects end-of-data and the EODAD routine is entered. You can override this with the PASTEOD parameter on the DCBE macro.
- If you specify SUL in the LABEL value when creating an extended format data set, the system treats it as SL. No space for user labels is allocated.

- All physical blocks in an extended format data set are the same size but when a program reads a block, the access method returns the length written by the writing program. The maximum block size for the data set is in the BLKSIZE field in the DCB or DCBE, depending on whether you are using LBI. The block size of the data set is determined to be BLKSIZE in the DCB or DCBE at OPEN for QSAM, or the maximum of BLKSIZE at OPEN and BLKSIZE at first WRITE for BSAM. For RECFM=U, length can be taken from the DECB instead of the DCB or DCBE at first WRITE. The system pads short blocks passed by the user so that full blocks are written. See “Determining the Length of a Block when Reading with BSAM, BPAM, or BDAM” on page 393 for more information. However, an attempt to write a block with a larger value than the data set’s maximum fails with ABEND 002-68.
- Extended format data sets allocated with more than one stripe cannot be extended to another set of volumes. These data sets cannot become multivolume data sets. The term multivolume should not be used for a multistriped data set because the volumes that comprise multiple stripes of a data set appear to the user as one logical volume.

  An extended format data set allocated with a single stripe can be extended to additional volumes. Such a data set can become a multivolume data sets. All the rules that apply to physical, sequential multivolume data sets also apply to single-striped, multivolume extended-format data sets.

Allocating Compressed-Format Data Sets

An extended-format data set can be allocated in the compressed format by specifying COMPACTION = YES in the data class. Extended format data sets allocated in the compressed format can be referred to as compressed format data sets. A compressed format data set is created with a data format which permits block level compression.

Types of Compression

Two compression techniques are available for sequential extended format data sets allocated in the compressed format. They are DBB-based compression and tailored compression. These techniques determine the method used to derive a compression dictionary for the data sets:

- **DBB-based compression (also referred to as GENERIC).** With DBB-based compression (the original form of compression used with both sequential and VSAM KSDS compressed format data sets), the system selects a set of dictionary building blocks (DBBs), found in SYS1.DBBLIB, which best reflects the initial data written to the data set. The system can later reconstruct the dictionary by using the information in the dictionary token stored in the catalog.
- **Tailored compression.** With tailored compression, the system attempts to derive a compression dictionary tailored specifically to the initial data written to the data set. Once derived, the compression dictionary is stored in system blocks which are imbedded within the data set itself. An OPEN for input reconstructs the dictionary by reading in the system blocks containing the dictionary.

  This form of compression is not supported for VSAM KSDSs.

Processing a Sequential Data Set

The form of compression the system is to use for newly created sequential compressed-format data sets can be specified at either or both the data set level and at the installation level. At the data set level, the storage administrator can specify TAILORED or GENERIC on the COMPACTION option in the data class. When the data class does not specify at the data set level, it is based on the COMPRESS(TAILORED|GENERIC) parameter found in the `IGDSMSxx` member of `SYS1.PARMLIB`. If the data class specifies the compression form, this takes precedence over that which is specified in SYS1.PARMLIB.
COMPRESS(GENERIC) refers to generic DBB-based compression. This is the default. COMPRESS(TAILORED) refers to tailored compression. When this member is activated using `SET SMS=xx` or IPL, new compressed format data sets are created in the form specified. The COMPRESS parameter in PARMLIB is ignored for VSAM KSDSs. For a complete description of this parameter see *z/OS DFSMSdfp Storage Administration Reference*.

Unless otherwise stated, all characteristics which apply to extended format data sets continue to apply to compressed format data sets. However, due to the differences in data format, there are some differences in characteristics.

The following characteristics are specific to compressed format data sets:

- A compressed format data set might or might not contain compressed records.
- The data format for a compressed format data set consists of physical blocks whose length has no correlation to the logical block size of the data set in the DCB, DCBE, and the data set label. The actual physical block size is calculated by the system and is never returned to the user. However, the system maintains the user's block boundaries when the data set is created so that the system can return the original user blocks to the user when the data set is read.
- A compressed format data set cannot be opened for update.
- When issued for a compressed format data set, the BSAM CHECK macro does not ensure that data is written to DASD. However, it does ensure that the data in the buffer has been moved to an internal system buffer, and that the user buffer is available to be reused.
- The block locator token returned by NOTE and used as input to POINT continues to be the relative block number (RBN) within each logical volume of the data set. A multistriped data set is seen by the user as a single logical volume. Therefore, for a multistriped data set the RBN is relative to the beginning of the data set and incorporates all stripes. To provide compatibility, this RBN refers to the logical user blocks within the data set as opposed to the physical blocks of the data set.
- However, due to the NOTE/POINT limitation of the 3 byte token, issuing a READ or WRITE macro for a logical block whose RBN value exceeds 3 bytes results in an ABEND if the DCB specifies NOTE/POINT (MACRF=P).

- When the data set is created, the system attempts to derive a compression token when enough data is written to the data set (between 8K and 64K for DBB compression and much more for tailored compression). If the system is successful in deriving a compression token, the access method attempts to compress any additional records written to the data set. However, if an efficient compression token could not be derived, the data set is marked as noncompressible and there is no attempt to compress any records written to the data set. However, if created with tailored compression, it is still possible to have system blocks imbedded within the data set although a tailored dictionary could not be derived.

  If the compressed format data set is closed before the system is able to derive a compression token, the data set is marked as noncompressible. Additional OPENs for output do not attempt to generate a compression token once the data set has been marked as noncompressible.
- A compressed format data set can be created using the LIKE keyword and not just through a DATACLAS.

Opening and Closing Extended Format Data Sets

If a DCBE exists and the data set is an extended format sequential data set, OPEN stores the number of stripes of the data set in the DCBE (DCBENSTR) before the OPEN exit is called. If a DCBE exists and the data set is not an extended format data set, OPEN stores 0 in DCBENSTR.

For a partial release request on an extended format sequential data set, CLOSE performs a partial release on each stripe. After the partial release, the size of some stripes can differ slightly from others. This difference is, at most, only one track or cylinder.

Reading, Writing, and Updating Extended Format Data Sets Using BSAM and QSAM

Extended format data sets are processed like other sequential data sets, except the data class and storage class must indicate sequential data striping.

Concatenating Extended-Format Data Sets with Other Data Sets

You can concatenate extended format data sets with non-extended-format data sets. There are no incompatibilities or added restrictions associated with concatenating extended-format data sets with non-extended-format data sets. For a QSAM concatenation containing extended-format sequential data sets, the system can recalculate the default BUFNO when switching between data sets.

Migrating to Extended-Format Data Sets

The following sections discuss changes you might need to make to take advantage of sequential data striping:

- Changing existing BSAM and QSAM applications
- Calculating DASD space used
- Changing to extended format data sets on devices with more than 64K tracks

Changing Existing BSAM and QSAM Applications

For existing BSAM, and QSAM applications, in most cases, programs do not have to be changed or recompiled to take advantage of sequential data striping. However, you can choose to update programs in order to more fully exploit new functions. In order to improve performance, you will want to have more buffers. You also can have the buffers be above the 16 MB line.

If you use BSAM, you can set a larger NCP value or have the system calculate an NCP value by means of the DCBE macro MULTSDN parameter. You can also request accumulation by means of the DCBE macro MULTACC parameter. DCBENSTR in the DCBE macro tells the number of stripes for the current data set. The DCBE and IHADCBE macros are described in *z/OS DFSMS Macro Instructions for Data Sets*. See “DASD and Tape Performance” on page 391.

If you use QSAM, you can request more buffers using the BUFNO parameter. Your program can calculate BUFNO according to the number of stripes. Your program can test DCBENSTR in the DCBE during the DCB open exit routine.

Existing programs need to be changed and reassembled if you want any of the following:

- To switch from 24-bit addressing mode to 31-bit mode SAM.
- To ask the system to determine an appropriate NCP value. Use the MULTSDN parameter of the DCBE macro.
- To get maximum benefit from BSAM performance chaining. You must change the program by adding the DCBE parameter to the DCB macro and including the DCBE macro with the MULTACC parameter. If the program uses WAIT or EVENTS or a POST exit (instead of, or in addition to, the CHECK macro), your program must issue the TRUNC macro whenever the WAIT or EVENTS macro is about to be issued or the POST exit is depended upon to get control. See “DASD and Tape Performance” on page 391.

Calculating DASD Space Used

Some programs read a data set’s data set control block (DSCB) to calculate the number of tracks used or the amount of unused space. For extended format data sets, the fields DS1LSTAR and DS1TRBAL have different meanings than for physical sequential data sets. You can change your program to test DS1STRIP or you can change it to test DCBESIZE in the DCBE. DSCB fields are described in *z/OS DFSMSdfp Advanced Services*. The DCBE fields are described in *z/OS DFSMS Macro Instructions for Data Sets*.

Extended format data sets can use more than 65 535 tracks on each volume. They use DS1TRBAL in conjunction with DS1LSTAR to represent one less than the number of tracks containing data. Thus, for extended format data sets, DS1TRBAL does not reflect the amount of space remaining on the last track written. Programs that rely on DS1TRBAL to determine the amount of free space must first check if the data set is an extended format data set.

Accessing USS Files Using BSAM, QSAM, or VSAM

You can access a USS file with BSAM or QSAM (DCB DSORG=PS) or through VSAM by specifying PATH=pathname in the JCL DD statement, SVC 99, or TSO ALLOCATE command. For information about using MVS access methods, see "Access to Files in HFS Data Sets" on page 7 and "Processing USS Files with an Access Method" on page 18.

USS files are POSIX-conforming files that reside in USS ″file systems.″ USS supports a variety of file systems, including HFS (hierarchical file system), NFS (network file system), and TFS (temporary file system). See *z/OS UNIX System Services Planning*. USS files are byte oriented, not record oriented like data sets. To identify and access them, you specify the path leading to them. They are the only type of data sets that you can access with both VSAM and non-VSAM interfaces. See "Simulated VSAM Access to USS files" on page 86.

When you use BSAM or QSAM, the system attempts to support a USS file so that the application program sees the file as a single-volume, physical, sequential data set that resides on DASD.

Since USS files are not actually stored as physical sequential data sets, the system cannot simulate all the characteristics of a sequential data set. Because of this, there are certain macros and services which have incompatibilities or restrictions when dealing with USS files.

Record Processing

Consider the following when you process records in USS files:

- Block boundaries are not maintained within the file. If you write a short block other than at the end of the file, a later read at that point will return a full block (except for RECFM=VB, which always returns a single record).
- Record boundaries are not maintained within binary files except with fixed-length records.
- Text files are presumed to be EBCDIC.
- Repositioning functions (such as POINT, BSP, CLOSE TYPE=T) is not permitted for FIFO or character special files.
- The default record format (DCBRECFM) is U for input and output.
- The default block size (DCBBLKSI) on input is 80. There is no default for output.
- The default LRECL (DCBLRECL) on input is 80. There is no default for output.
- When RECFM=F(B(S))
  - And the file is accessed as binary, if the last record in the file is smaller than LRECL bytes, it will be padded with zeros when read.
  - And the file is accessed as text, if any record in the file is smaller than LRECL bytes, it will be padded with blanks when read. If any record is longer than LRECL bytes, it will result in an I/O error due to incorrect length when read.

- When RECFM=V(B)
  - And the file is accessed as binary, each record will be returned as length LRECL, except, possibly, for the last one.
  - And the file is accessed as text, if any record in the file consists of zero bytes (that is, a text delimiter is followed by another text delimiter), the record returned will consist of an RDW and no data bytes. If any record is longer than LRECL bytes, it will result in an I/O error due to incorrect length when read.
- When RECFM=U
  - And the file is accessed as binary, each record will be returned with a length equal to block size, except, possibly, for the last one.
  - And the file is accessed as text, if any record in the file consists of zero bytes (that is, a text delimiter is followed by another text delimiter), the record returned will consist of one blank. If any record is longer than the block size, it will result in an I/O error due to incorrect length when read.

Restrictions

The following restrictions are associated with using BSAM and QSAM with USS files:

- OPEN for UPDAT is not supported.
- EXCP is not supported.
- DCB RECFM=V(B)S (spanned record format) is not supported.
- DCB MACRF=P (NOTE/POINT) is not supported for FIFO, for character special files, or if PATHOPTS=OAPPEND is specified.
- NOTE/POINT does not support a file that contains more than 16 megarecords minus two. A NOTE after 16 megarecords minus two (16 777 214) returns an invalid value of x'FFFFFF'. A POINT to an invalid value causes the next READ or WRITE to fail with an I/O error, unless preceded by another POINT.
- In a binary file with RECFM=V(B) or RECFM=U, a POINT to a block other than the first block in the file results in an ABEND.
- You can issue BSP only after a successful CHECK (for READ or WRITE), NOTE, or CLOSE TYPE=T LEAVE request.

Services and Utilities

The following services and utilities support USS files:

- DEVTYPE macro

  In prior levels of the operating system in which BSAM and QSAM did not support USS files, DEVTYPE gave return code 8 for an HFS file.
- TTR convert routines

  When called for a USS file, the TTR convert routines return the input value without performing conversion.

The following services and utilities do not support USS files. Unless stated otherwise, they will return an error or unpredictable value when issued for a USS file.

- OBTAIN
- SCRATCH
- RENAME
- TRKCALC
- Sector Convert Routine
- PARTREL
- PURGE by DSID is ignored

The preceding services and utilities require a DSCB or UCB. USS files do not have DSCBs or valid UCBs.

ISPF Browse/Edit does not support USS files, but you can use the OBROWSE command.

Sequential Concatenation

As with physical sequential data sets, when a USS file is found within a *sequential concatenation*, the system forces the use of the LRECL, RECFM, and BUFNO from the previous data set. (The *unlike* attributes bit is not set in a sequential concatenation.) Also, the system uses the same NCP and BLKSIZE values as for any BSAM sequential *like* concatenation. For QSAM, the system uses the value of BLKSIZE for each data set.

If you read a USS file in a concatenation, the transition out of the file must be in the same task as the task that made the transition into the file.

See *z/OS DFSMS Macro Instructions for Data Sets* for more information about BSAM, QSAM and VSAM interfaces and HFS files.

Chapter 27. Processing a Partitioned Data Set (PDS)

This chapter covers the following topics.

Topic	Page
Structure of a PDS	405
PDS Directory	406
Allocating Space for a PDS	409
Creating a PDS	410
Processing a Member of a PDS	413
Retrieving a Member of a PDS	419
Modifying a PDS	423
Concatenating PDSs	426
Reading a PDS Directory Sequentially	427

Processing PDSEs is described in “Chapter 28. Processing a Partitioned Data Set Extended (PDSE)” on page 429.

Structure of a PDS

A PDS is stored only on a direct access storage device. It is divided into sequentially organized members, each described by one or more directory entries.

Each member has a unique name, 1 to 8 characters long, stored in a directory that is part of the data set. The records of a given member are written or retrieved sequentially. See *z/OS DFSMS Macro Instructions for Data Sets* for the macros used with PDSs.

The main advantage of using a PDS is that, without searching the entire data set, you can retrieve any individual member after the data set is opened. For example, in a program library that is always a PDS, each member is a separate program or subroutine. The individual members can be added or deleted as required. When a member is deleted, the member name is removed from the directory, but the space used by the member cannot be reused until the data set is reorganized; that is, compressed using the IEBCOPY utility.

The directory, a series of 256-byte records at the beginning of the data set, contains an entry for each member. Each directory entry contains the member name and the starting location of the member within the data set (see Figure 73 on page 406). You can also specify as many as 62 bytes of information in the entry. The directory entries are arranged by name in alphanumeric collating sequence.

© Copyright IBM Corp. 1987, 2002

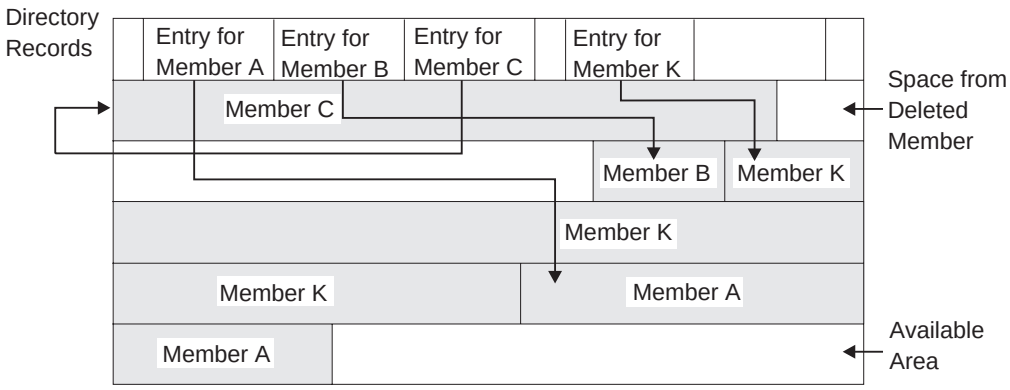


Figure 73. A Partitioned Data Set (PDS)

The starting location of each member is recorded by the system as a relative track address (from the beginning of the data set) rather than as an absolute track address. Thus, an entire data set that has been compressed can be moved without changing the relative track addresses in the directory. The data set can be considered as one continuous set of tracks regardless of where the space was actually allocated.

If there is not sufficient space available in the directory for an additional entry, or not enough space available within the data set for an additional member, or no room on the volume for additional extents, no new members can be stored. A directory cannot be extended and a PDS cannot cross a volume boundary.

PDS Directory

The directory of a PDS occupies the beginning of the area allocated to the data set on a direct access volume. It is searched and maintained by the BLDL, FIND, and STOW macros. The directory consists of member entries arranged in ascending order according to the binary value of the member name or alias.

PDS member entries vary in length and are blocked into 256-byte blocks. Each block contains as many complete entries as will fit in a maximum of 254 bytes. Any remaining bytes are left unused and are ignored. Each directory block contains a 2-byte count field that specifies the number of active bytes in a block (including the count field). In Figure 74, each block is preceded by a hardware-defined key field containing the name of the last member entry in the block, that is, the member name with the highest binary value. Figure 74 shows the format of the block returned when using BSAM to read the directory.

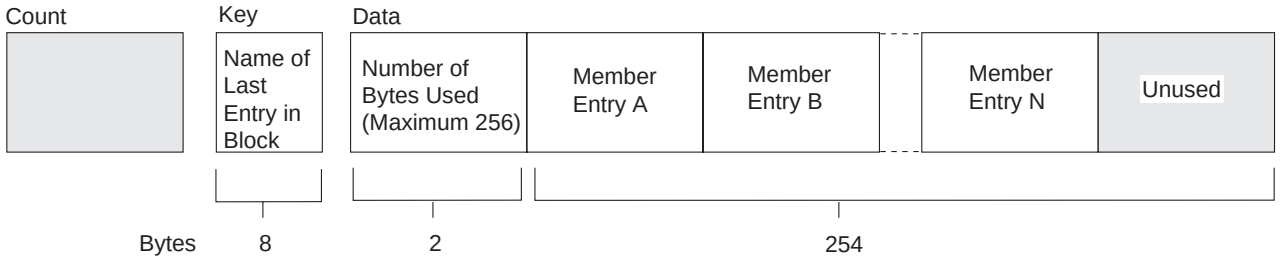


Figure 74. A PDS Directory Block

Each member entry contains a member name or an alias. As shown in Figure 75 , each entry also contains the relative track address of the member and a count field. It can also contain a user data field. The last entry in the last used directory block has a name field of maximum binary value (all 1s, a TTR field of zeros, and a zero-length user data field).

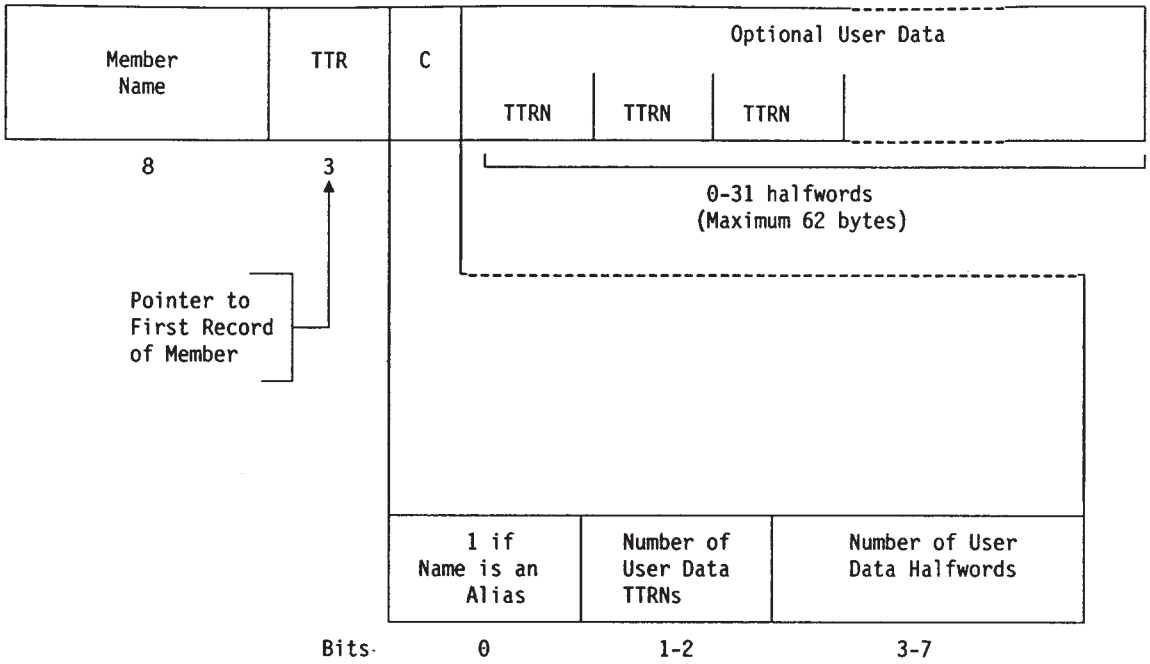


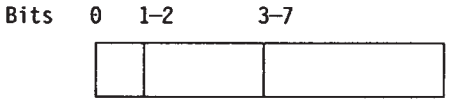
Figure 75. A PDS Directory Entry

Figure 75 shows the following fields:

Member Name—Specifies the member name or alias. It contains as many as 8 alphanumeric characters, left justified, and padded with blanks if necessary.

TTR—Is a pointer to the first block of the member. TT is the number of the track, starting from 0 for the beginning of the data set, and R is the number of the block, starting from 1 for the beginning of that track.

C—Specifies the number of halfwords contained in the user data field. It can also contain additional information about the user data field, as shown below:



0—When set to 1, indicates that the NAME field contains an alias.

1-2—Specifies the number of pointers to locations within the member.

The operating system supports a maximum of three pointers in the user data field. Additional pointers can be contained in a record called a note list discussed in the following note. The pointers can be updated automatically if the data set is moved

or copied by a utility program such as IEHMOVE. The data set must be marked unmovable under any of the following conditions:

- More than three pointers are used in the user data field.
- The pointers in the user data field or note list do not conform to the standard format.

  A note list for a PDS containing variable length records does not conform to standard format. Variable-length records contain BDWs and RDWs that are treated as TTRXs by IEHMOVE.
- The pointers are not placed first in the user data field.
- Any direct access address (absolute or relative) is embedded in any data blocks or in another data set that refers to the data set being processed.

3-7—Contains a binary value indicating the number of halfwords of user data. This number must include the space used by pointers in the user data field.

You can use the user data field to provide variable data as input to the STOW macro. If pointers to locations within the member are provided, they must be 4 bytes long and placed first in the user data field. The user data field format is as follows:

User Data

TTRN	TTRN	TTRN	Optional

TT—Is the relative track address of the note list or the area to which you are pointing.

R—Is the relative block number on that track.

N—Is a binary value that shows the number of additional pointers contained in a note list pointed to by the TTR. If the pointer is not to a note list, N=0.

A note list consists of additional pointers to blocks within the same member of a PDS. You can divide a member into subgroups and store a pointer to the beginning of each subgroup in the note list. The member can be a load module containing many control sections (CSECTs), each CSECT being a subgroup pointed to by an entry in the note list. Use the NOTE macro to point to the beginning of the subgroup after writing the first record of the subgroup. Remember that the pointer to the first record of the member is stored in the directory entry by the system.

If a note list exists, as shown above, the list can be updated automatically when the data set is moved or copied by a utility program such as IEHMOVE. Each 4-byte entry in the note list has the following format:

TTRX

TT—Is the relative track address of the area to which you are pointing.

R—Is the relative block number on that track.

X—Is available for any use.

To place the note list in the PDS, you must use the WRITE macro. After checking the write operation, use the NOTE macro to determine the address of the list and place that address in the user data field of the directory entry.

The linkage editor builds a note list for the load modules in overlay format. The addresses in the note list point to the overlay segments that are read into the system separately.

Restriction: Note lists are not supported for PDSEs. If a PDS is to be converted to a PDSE, the PDS should not use note lists.

Allocating Space for a PDS

If you do not specify a block size and the record format is fixed or variable, OPEN determines an optimum block size for you. Therefore, you do not need to perform calculations based on track length. When you allocate space for your data set, specify the average record length in kilobytes or megabytes by using the SPACE and AVGREC parameters, and have the system use the block size it calculated for your data set.

If your data set is large, or if you expect to update it extensively, it might be best to allocate a large data set. A PDS cannot occupy more than 65 535 tracks and cannot extend beyond one volume. If your data set is small or is seldom changed, let the system calculate the space requirements to avoid wasted space or wasted time used for recreating the data set.

VSAM, extended format, HFS, and PDSE data sets can occupy more than 65 535 tracks.

Calculating Space

If you want to estimate the space requirements yourself, you need to answer the following questions to estimate your space requirements accurately and use the space efficiently.

- What is the average size of the members to be stored on your direct access volume?
- How many members will fit on the volume?
- Will you need directory entries for the member names only, or will aliases be used? If so, how many?
- Will members be added or replaced frequently?

You can calculate the block size yourself and specify it in the BLKSIZE parameter of the DCB or DCBE. For example, if the average record length is close to or less than the track length, or if the track length exceeds 32 760 bytes the most efficient use of the direct access storage space can be made with a block size of one-third or one-half the track length.

For a 3380 DASD, you might then ask for either 75 tracks, or 5 cylinders, thus permitting for 3 480 ,000 bytes of data. Assuming the allocation size of 3 480 000 bytes and an average length of 70 000 bytes for each member, you need space for at least 50 directory entries. If each member also has an average of three aliases, space for an additional 150 directory entries is required.

Each member in a data set and each alias need one directory entry apiece. If you expect to have 10 members (10 directory entries) and an average of 3 aliases for each member (30 directory entries), allocate space for at least 40 directory entries.

Space for the directory is expressed in 256-byte blocks. Each block contains from 3 to 21 entries, depending on the length of the user data field. If you expect 200 directory entries, request at least 10 blocks. Any unused space on the last track of the directory is wasted unless there is enough space left to contain a block of the first member.

Any of the following space specifications would allocate approximately the same amount of space for a 3380 DASD. Ten blocks have been allocated for the directory. The first two examples would not allocate a separate track for the directory. The third example would result in allocation of 75 tracks for data, plus 1 track for directory space.

```
SPACE=(CYL,(5,,10))

SPACE=(TRK,(75,,10))

SPACE=(23200,(150,,10))
```

Allocating Space with SPACE and AVGREC

You can also allocate space by using both the SPACE and AVGREC JCL keywords together. In the following examples, the average length is 70000 bytes for each member, each record in the member is 80 bytes long, and the block size is 23200. Using the AVGREC keyword changes the first value specified in SPACE from the average block length to average record length. These examples are device independent because they request space in bytes, rather than tracks or cylinders. They would allocate approximately the same amount of space as the previous examples (about 75 tracks if the device were a 3380 disk).

```
SPACE=(80,(44,,10)),AVGREC=K

SPACE=(80,(43500,,10)),AVGREC=U
```

For more information on using the SPACE and AVGREC parameters, see “Chapter 3. Allocating Space on Direct Access Volumes” on page 37 in this manual, and also see *z/OS MVS JCL Reference* and *z/OS MVS JCL User’s Guide*.

Although a secondary allocation increment has been omitted in these examples, it could have been supplied to provide for extension of the member area. The directory size, however, cannot be extended. The directory must be in the first extent.

Recommendation: The SPACE parameter can be derived from either the data class, the LIKE keyword, or the DD statement. Specify the SPACE parameter in the DD statement if you do not want to use the space allocation amount defined in the data class.

Creating a PDS

You can create a PDS or members of a PDS with BSAM, QSAM, or BPAM.

Creating a PDS Member with BSAM or QSAM

If you have no need for your program to add entries to the directory (the STOW macro is not used), you can write a member of a PDS, such as the one in Figure 76 on page 411.

The following steps create the data set and its directory, write the records of the member, and make a 12-byte entry in the directory:

1. Code DSORG=PS or DSORG=PSU in the DCB macro.
2. In the DD statement specify that the data is to be stored as a member of a new PDS, that is, DSNAME=name(membername) and DISP=NEW.
3. Optionally specify a data class in the DD statement or let the ACS routines assign a data class.
4. Request space for the member and the directory in the DD statement, or obtain from data class.
5. Process the member with an OPEN macro, a series of PUT or WRITE macros, and then a CLOSE macro. A STOW macro is issued automatically when the data set is closed.

```
//PDSDD  DD    ---,DSNAME=MASTFILE(MEMBERK),SPACE=(TRK,(100,5,7)),
//             DISP=(NEW,CATLG),DCB=(RECFM=FB,LRECL=80,BLKSIZE=80)---
         ...
         OPEN  (OUTDCB,(OUTPUT))
         ...
         PUT   OUTDCB,OUTAREA     Write record to member
         ...
         CLOSE (OUTDCB)           Automatic STOW
         ...
OUTAREA  DS    CL80               Area to write from
OUTDCB   DCB   ---,DSORG=PS,DDNAME=PDSDD,MACRF=PM
```

Figure 76. Creating One Member of a PDS

If the preceding conditions are true but you code DSORG=PO (to use BPAM) and your last operation on the DCB before CLOSE is a STOW macro, CLOSE does not issue the STOW macro.

Converting PDSs

You can use IEBCOPY or DFSMSdss COPY to convert a PDS to a PDSE, or a PDSE to a PDS. See *z/OS DFSMS Migration* for how to plan conversion of PDSs and “Converting PDSs to PDSEs and Back” on page 467 for examples of using IEBCOPY and DFSMSdss to convert PDSs to PDSEs.

Adding Members

To add additional members to the PDS, follow the procedure described in Figure 76. However, a separate DD statement (with the space request omitted) is required for each member. The disposition should be specified as modify (DISP=MOD). The data set must be closed and reopened each time a new member is specified on the DD statement.

You can use the basic partitioned access method (BPAM) to process more than one member without closing and reopening the data set. Use the STOW, BLDL, and FIND macros to provide more information with each directory entry, as follows:

- Request space in the DD statement for the entire data set and the directory.
- Define DSORG=PO or DSORG=POU in the DCB macro.
- Use WRITE and CHECK to write and check the member records.
- Use NOTE to note the location of any note list written within the member, if there is a note list, or to note the location of any subgroups. A note list is used to point to the beginning of each subgroup in a member.

- When all the member records have been written, issue a STOW macro to enter the member name, its location pointer, and any additional data in the directory. The STOW macro writes an end-of-file mark after the member.
- Continue to use the WRITE, CHECK, NOTE, and STOW macros until all the members of the data set and the directory entries have been written.

Figure 77 shows an example of using STOW to create members of a PDS.

```
//PDSDD  DD    ---,DSN=MASTFILE,DISP=MOD,SPACE=(TRK,(100,5,7))
         ...
         OPEN  (OUTDCB,(OUTPUT))
         LA    STOWREG,STOWLIST   Load address of STOW list
         ...

**   WRITE MEMBER RECORDS AND NOTE LIST

MEMBER   WRITE DECBX,SF,OUTDCB,OUTAREA   WRITE  first record of member
         CHECK DECBX
         LA    NOTEREG,NOTELIST          Load address of NOTE list
*
         WRITE DECBY,SF,OUTDCB,OUTAREA   WRITE and CHECK next record
         CHECK DECBY
*
         NOTE  OUTDCB                  To divide the member into subgroups,
         ST    R1,0(NOTEREG)           NOTE the TTRN of the first record in
*                                      the subgroup, storing it in the NOTE list.
         LA    NOTEREG,4(NOTEREG)      Increment to next NOTE list entry
         ...
         WRITE DECBZ,SF,OUTDCB,NOTELIST   WRITE NOTE list record at the
*                                         end of the member
         CHECK DECBZ
         NOTE  OUTDCB                 NOTE TTRN of NOTE list record
         ST    R1,12(STOWREG)         Store TTRN in STOW list
         STOW  OUTDCB,(STOWREG),A     Enter the information in directory
*                                     for this member after all records
*                                     and NOTE lists are written.
         LA    STOWREG,16(STOWREG)    Increment to the next STOW list entry
         ...
Repeat from label "MEMBER" for each additional member, changing the
member name in the "STOWLIST" for each member
         ...
         CLOSE (OUTDCB)      (NO automatic STOW)
         ...

OUTAREA  DS    CL80          Area to write from
OUTDCB   DCB   ---,DSORG=PO,DDNAME=PDSDD,MACRF=W
R1       EQU   1             Register one, return register from NOTE
NOTEREG  EQU   4             Register to address NOTE list
NOTELIST DS    0F            NOTE list
         DS    F             NOTE list entry  (4 byte TTRN)
         DS    19F           one entry per subgroup
STOWREG  EQU   5             Register to address STOW list
STOWLIST DS    0F            List of member names for STOW
         DC    CL8'MEMBERA'  Name of  member
         DS    CL3           TTR of first record (created by STOW)
         DC    X'23'         C byte, 1 user TTRN, 4 bytes of user data
         DS    CL4           TTRN of NOTE list
         ...                 one list entry per member (16 bytes each)
```

Figure 77. Creating Members of a PDS Using STOW

Recommendation: Do not use the example in Figure 77 on page 412 for PDSEs. If your installation plans to convert PDSs to PDSEs, follow the procedure described in Figure 92 on page 443.

Processing a Member of a PDS

Because a member of a PDS is sequentially organized, it is processed in the same manner as a sequential data set. To locate a member or to process the directory, several macros are provided by the operating system. The BLDL macro can be used to read one or more directory entries into virtual storage. The FIND macro locates a member of the data set and positions the DCB for subsequent processing. The STOW macro adds, deletes, replaces, or changes a member name in the directory. To use these macros, you must specify DSORG=PO or POU in the DCB macro. Before issuing the FIND, BLDL, or STOW macro, you must check all preceding I/O operations for completion.

BLDL—Construct a Directory Entry List

The BLDL macro reads one or more directory entries into virtual storage. Place member names in a BLDL list before issuing the BLDL macro. For each member name in the list, the system supplies the relative track address (TTR) and any additional information contained in the directory entry. If there is more than one member name in the list, the member names must be in collating sequence, regardless of whether the members are from the same or different PDSs or PDSEs in the concatenation.

BLDL also searches a concatenated series of directories when (1) a DCB is supplied that is opened for a concatenated PDS or (2) a DCB is not supplied, in which case the search order begins with the TASKLIB, then proceeds to the JOBLIB or STEPLIB (themselves perhaps concatenated) followed by LINKLIB.

You can improve retrieval time by directing a subsequent FIND macro to the BLDL list rather than to the directory to locate the member to be processed.

By specifying the BYPASSLLA option, you can direct BLDL to search PDS and PDSE directories on DASD only. If BYPASSLLA is coded, the BLDL code will not call LLA to search for member names.

The BLDL list must begin with a 4-byte list descriptor that specifies the number of entries in the list and the length of each entry (12 to 76 bytes). (See Figure 78 on page 414.) If you specify the BYPASSLLA option, an 8-byte BLDL prefix must precede the 4-byte list descriptor.

(Each entry starts on halfword boundary)

List Description: FFLL

Filled in by BLDL

Member Name (C)	TTR (3)	K (1)	Z (1)	C (1)	User Data (C Halfwords)

```
Programmer Supplies:
             FF   Number of member entries in list.
             LL   Even number giving byte length of each entry (minimum of 12).
    Member Name   Eight bytes, left-justified.

BLDL Supplies:
        TTR   Member starting location.
          K   If single data set = 0.  If concatenation = number.
              Not required if no user data.
          Z   Source of directory entry.  Private library = 0.
              Link library = 1.  Job or step library = 2.
              Not required if no user data.
          C   Same C field from directory.  Gives number of user data halfwords.
  User Data   As much as will fit in entry.
```

Figure 78. BLDL List Format

The first 8 bytes of each entry contain the member name or alias. The next 6 bytes contain the TTR, K, Z, and C fields. If there is no user data entry, only the TTR and C fields are required. If additional information is to be supplied from the directory, as many as 62 bytes can be reserved.

DESERV

The DESERV macro returns SMDEs for specific members or all members of opened PDS or PDSEs. You can specify either DESERV GET or DESERV GET_ALL.

FUNC=GET

DESERV GET returns SMDEs for specific members of opened PDS or PDSEs, or a concatenation of PDSs and PDSEs. The data set can be opened for either input, output, or update. The system managed directory entry (SMDE) contain the PDS or PDSE directory. The SMDE is mapped by the macro IGWSMDE and contains a superset of the information that is mapped by IHAPDS. The SMDE returned can be selected by name or by BLDL directory entry.

Input by Name List: If you want to select SMDEs by name, you supply a list of names must be sorted in ascending order, without duplicates. Each name is comprised of a two-byte length field followed by the characters of the name. When searching for names with less than eight characters, the names are padded on the right with blanks to make up eight characters. Names greater than eight characters will have trailing blanks and nulls stripped (to a minimum length of eight) before the search.

In addition to retrieving the SMDE, member level connections can be established for each member name found. The members are connected with the HOLD type

connection. A connection type of HOLD ensures that the member cannot be removed from the system until the connection is released. The connection intent is specified by the CONN_INTENT parameter, CONN_INTENT=HOLD must be specified.

All connections made through a single call to GET are associated with a single unique connect identifier. The connect identifier may be used to release all the connections in a single invocation of the RELEASE function. Figure 79 shows an example of DESERV GET:

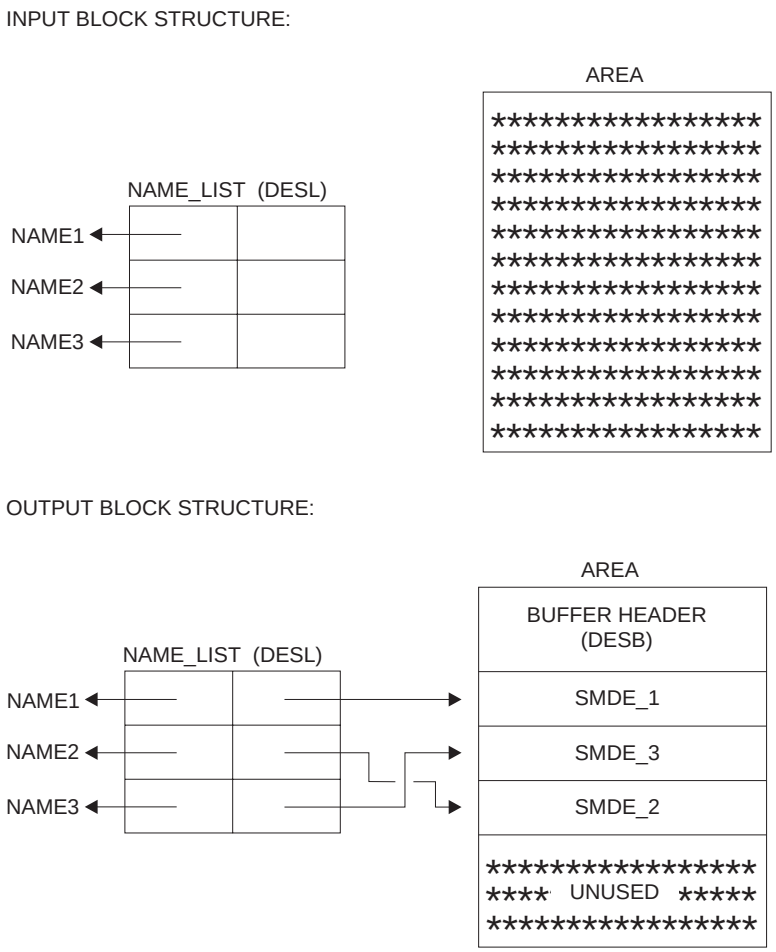


Figure 79. DESERV GET by NAME_LIST Control Block Structure

Input by BLDL Directory Entry (PDSDE): If the search argument specified is a PDSDE, the GET function is significantly restricted. The PDSDE (as mapped by the IHAPDS macro) identifies only one name to be searched for. Since the PDSDE also identifies the concatenation number of the library in which this member is to reside (PDS2CNCT), only that library can be searched. Since the PDSDE identifies a specific version of the member name (this identification is made through MLT (PDS2TTRP)), the name can only be considered found if the same version can be found in the target library. However, a library search can only be performed if the target library is a PDSE. If the target library is a PDS, the input PDSDE will simply be converted to an equivalent directory entry in SMDE format and returned. No directory search can be performed. If the caller has specified BYPASS_LLA=NO, the library search will search LLA for LLA managed libraries. If the caller has specified BYPASS_LLA=YES, only the DASD directories of the library will be

searched. Figure 80 shows an example of DESERV GET by PDSDE control block:

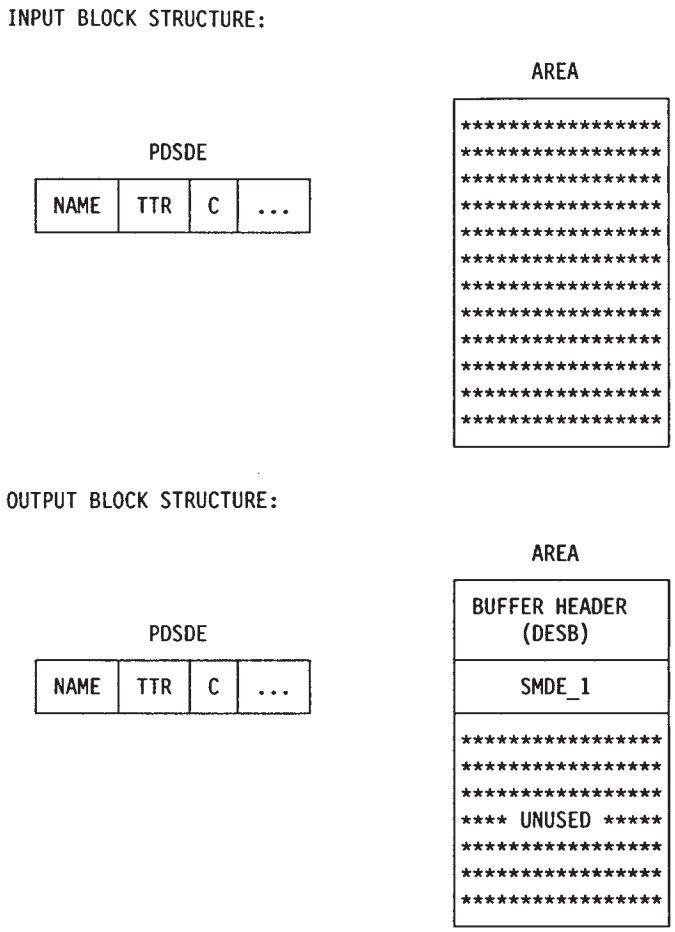


Figure 80. DESERV GET by PDSDE Control Block Structure

FUNC=GET_ALL

The GET_ALL function returns SMDEs for all the member names in a PDS, a PDSE, or a concatenation of PDSs and PDSEs. Member-level connections can be established for each member found in a PDSE. A caller uses the CONCAT parameter to indicate which data set in the concatenation is to be processed, or if all of the data sets in the concatenation are to be processed.

If the caller requests that DESERV GET_ALL return all the SMDE directory entries for an entire concatenation, the SMDEs are returned in sequence as sorted by the SMDE_NAME field without returning duplicate names. As with the GET function, all connections can be associated with a single connect identifier established at the time of the call. This connect identifier can then be used to release all the connections in a single invocation of the RELEASE function. Figure 81 on page 417 shows an overview of control blocks related to the GET_ALL function.

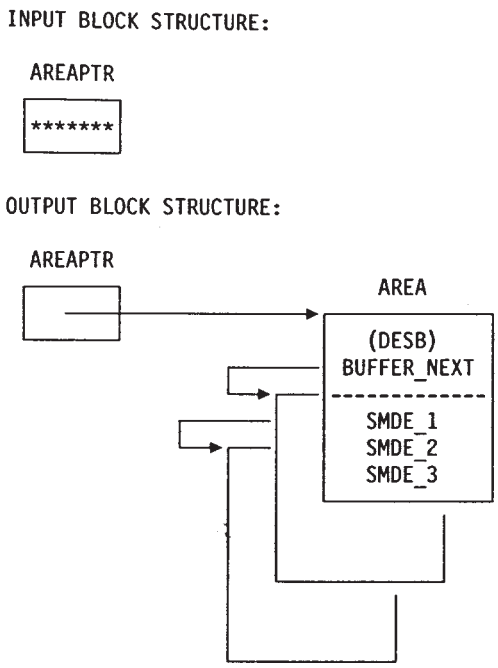


Figure 81. DESERV GET_ALL Control Block Structure

FIND—Position to the Starting Address of a Member

To position to the beginning of a specific member, you must issue a FIND macro. The next input or output operation begins processing at the point set by the FIND. The FIND macro lets you search a concatenated series of PDSE and PDS directories when you supply a DCB opened for the concatenated data sets.

There are two ways you can direct the system to the right member when you use the FIND macro. Specify the address of an area containing the name of the member, or specify the address of the TTR field of the entry in a BLDL list you have created, by using the BLDL macro. In the first case, the system searches the directory of the data set for the relative track address. In the second case, no search is required, because the relative track address is in the BLDL list entry.

The system searches a concatenated series of directories when a DCB is supplied that is opened for a concatenated PDS.

If you want to process only one member, you can process it as a sequential data set (DSORG=PS) using either BSAM or QSAM. You specify the name of the member you want to process and the name of the PDS in the DSNAME parameter of the DD statement. When you open the data set, the system places the starting address in the DCB so that a subsequent GET or READ macro begins processing at that point. You cannot use the FIND, BLDL, or STOW macro when you are processing one member as a sequential data set.

Because the DCBRELAD address in the DCB is updated when the FIND macro is used, you should not issue the FIND macro after WRITE and STOW processing without first closing the data set and reopening it for INPUT processing.

STOW—Update the Directory

When you add more than one member to a PDS, you must issue a STOW macro after writing each member so that an entry for each one will be added to the directory. To use the STOW macro, DSORG=PO or POU must be specified in the DCB macro.

You can also use the STOW macro to delete, replace, or change a member name in the directory and store additional information with the directory entry. Because an alias can also be stored in the directory the same way, you should be consistent in altering all names associated with a given member. For example, if you replace a member, you must delete related alias entries or change them so that they point to the new member. An alias cannot be stored in the directory unless the member is present.

Although you can use any type of DCB with STOW, it is intended to be used with a BPAM DCB. If you use a BPAM DCB, you can issue several writes to create a member followed by a STOW to write the directory entry for the member. Following this STOW, your application can write and stow another member.

If you add only one member to a PDS, and specify the member name in the DSNAME parameter of the DD statement, it is not necessary for you to use BPAM and a STOW macro in your program. If you want to do so, you can use BPAM and STOW, or BSAM or QSAM. If you use a sequential access method, or if you use BPAM and issue a CLOSE macro without issuing a STOW macro, the system will issue a STOW macro using the member name you have specified on the DD statement.

Note that no checks are made in STOW to ensure that a stow with a BSAM or QSAM DCB came from CLOSE. When the system issues the STOW, the directory entry that is added is the minimum length (12 bytes). This automatic STOW macro will not be issued if the CLOSE macro is a TYPE=T or if the TCB indicates the task is being abnormally ended when the DCB is being closed. The DISP parameter on the DD statement determines what directory action parameter will be chosen by the system for the STOW macro.

If DISP=NEW or MOD was specified, a STOW macro with the add option will be issued. If the member name on the DD statement is not present in the data set directory, it will be added. If the member name is already present in the directory, the task will be abnormally ended.

If DISP=OLD was specified, a STOW macro with the replace option will be issued. The member name will be inserted into the directory, either as an addition, if the name is not already present, or as a replacement, if the name is present.

Thus, with an existing data set, you should use DISP=OLD to force a member into the data set; and DISP=MOD to add members with protection against the accidental destruction of an existing member.

Retrieving a Member of a PDS

To retrieve a specific member from a PDS, you can use either the basic access or queued access method, as follows (see Figure 82):

1. Code DSORG=PS or DSORG=PSU in the DCB macro.
2. In the DD statement specify that the data is a member of an existing PDS by coding DSNAME=name(membername) and DISP=OLD, DISP=SHR or DISP=MOD.
3. Process the member with an OPEN macro, a series of GET or READ macros, and then a CLOSE macro.

```
//PDSDD  DD    ---,DSN=MASTFILE(MEMBERK),DISP=SHR
         ...
         OPEN  (INDCB)            Open for input, automatic FIND
         ...
         GET   INDCB,INAREA       Read member record
         ...
         CLOSE (INDCB)
         ...

INAREA   DS    CL80               Area to read into
INDCB    DCB   ---,DSORG=PS,DDNAME=PDSDD,MACRF=GM
```

Figure 82. Retrieving One Member of a PDS

When your program is executed, OPEN searches the directory automatically and positions the DCB to the member.

To process several members without closing and reopening, or to take advantage of additional data in the directory, use the procedure described in Figure 83 on page 421 or Figure 84 on page 422.

The system supplies a value for NCP during OPEN. For performance reasons, the example shown in Figure 84 automatically takes advantage of the NCP value calculated in OPEN or set by the user on the DD statement. If the FIND macro is omitted and DSORG on the DCB changed to PS, the example shown in Figure 84 works to read a sequential data set with BSAM. The logic to do that is summarized in “Using Overlapped I/O with BSAM” on page 353.

Processing a Partitioned Data Set (PDS)

To retrieve a member of a PDS, take the following steps. Figure 83 on page 421 is an example that uses note lists, which should not be used with PDSEs.

1. Code DSORG=PO or POU in the DCB macro.
2. In the DD statement specify the data set name of the PDS by coding DSNAME=name.
3. Issue the BLDL macro to get the list of member entries you need from the directory.
4. Repeat the following steps for each member to be retrieved:
   a. Use the FIND macro to prepare for reading the member records. If you use the POINT macro it will not work in a partitioned concatenation.
   b. The records can be read from the beginning of the member, or a note list can be read first, to obtain additional locations that point to subcategories within the member. If you want to read out of sequential order, use the POINT macro to point to blocks within the member.
   c. Read (and check) the records until all those required have been processed.
   d. Your end-of-data-set (EODAD) routine receives control at the end of each member. At that time, you can process the next member or close the data set.

Figure 83 on page 421 shows the technique for processing several members without closing and reopening. This demonstrates synchronous reading.

```
   //PDSDD  DD    ---,DSN=D42.MASTFILE,DISP=SHR

            ...
            OPEN  (INDCB)               Open for input, no automatic FIND
            ...
            BLDL  INDCB,BLDLLIST        Retrieve the relative disk locations
   *                                      of several names in virtual storage
            LA    BLDLREG,BLDLLIST+4     Point to the first entry

Begin a member possibly in another concatenated data set
   MEMBER   FIND  INDCB,8(,BLDLREG),C   Position to member
            ...
```

Read the NOTE list

```
            LA    NOTEREG,NOTELIST     Load address of NOTE list
            MVC   TTRN(4),14(BLDLREG)  Move NOTE list TTRN
   *                                   to fullword boundary
            POINT INDCB,TTRN           Point to the NOTE list record
            READ  DECBX,SF,INDCB,(NOTEREG)     Read the NOTE list
            CHECK DECBX
            ...
```

Read data from a subgroup

```
   SUBGROUP POINT INDCB,(NOTEREG)    Point to subgroup
            READ  DECBY,SF,INDCB,INAREA  Read record in subgroup
            CHECK DECBY
            LA    NOTEREG,4(NOTEREG)  Increment to next subgroup TTRN
            ...
Repeat from label "SUBGROUP" for each additional subgroup
            AH    BLDLREG,BLDLLIST+2

Repeat from label "MEMBER" for each additional member
           ...
           CLOSE (INDCB)
           ...

   INAREA   DS    CL80
   INDCB    DCB   ---,DSORG=PO,DDNAME=PDSDD,MACRF=R
   TTRN     DS    F             TTRN of the NOTE list to point at
   NOTEREG  EQU   4             Register to address NOTE list entries
   NOTELIST DS    0F            NOTE list
            DS    F             NOTE list entry  (4 byte TTRN)
            DS    19F           one entry per subgroup
   BLDLREG  EQU   5             Register to address BLDL list entries
   BLDLLIST DS    0F            List of member names for BLDL
            DC    H'10'         Number of entries (10 for example)
            DC    H'18'         Number of bytes per entry
            DC    CL8'MEMBERA'  Name of  member
            DS    CL3           TTR of first record (created by BLDL)
            DS    X             K byte, concatenation number
            DS    X             Z byte, location code
            DS    X             C byte, flag and user data length
            DS    CL4           TTRN of NOTE list
            ...                 one list entry per member (18 bytes each)
```

Figure 83. Retrieving Several Members and Subgroups of a PDS without Overlapping I/O Time and CPU Time

The example in Figure 84 on page 422 does not use large block interface (LBI). With BPAM there is no advantage in the current release to using LBI because the block size cannot exceed 32 760 bytes. You can convert the example to BSAM by omitting the FIND macro and changing DSORG in the DCB to PS. With BSAM LBI you can read tape blocks that are longer than 32 760 bytes.

Processing a Partitioned Data Set (PDS)

The technique shown in Figure 84 is more efficient than the technique shown in Figure 83 on page 421 because the access method is transferring data while the program is processing data that was previously read.

```
         OPEN  LIBDCB              Open DCB, setting RECFM, LRECL, and BLKSIZE
         USING IHADCB,LIBDCB
         USING DCBE,MYDCBE         DCB addressability (needs HLASM)
         TM    DCBOFLGS,DCBOFPPC   Branch if open
         BZ    ---                   failed
         FIND  LIBDCB,MEMNAME,D    Position to member to read
         SR    R3,R3               GET NCP calculated by OPEN or
         IC    R3,DCBNCP             coded on DD statement
 (1)     LH    R1,DCBBLKSI         Get maximum size of a block
ROUND    LA    R1,DATAOFF+7(,R1)   Add length of a DECB (READ MF=L) and pointer
         SRL   R1,3                  and round up to a
         SLL   R1,3                  doubleword length
         LR    R4,R1               Save length of DECB + pointer + block size
         MR    R0,R3               Calculate length of area to get
* Get area for DECB's, a pointer to the next DECB for each DECB and a data area
* for each DECB.  Each DECB is followed by a pointer to the next one and then the
* associated data area.  The DECB's are chained in a circle.  DECB's must be
* below line; data areas can be above line if running in 31-bit mode, however,
* they will be retrieved below the line in this example.
         ST    R1,AREALEN          Save length of area
         GETMAIN R,LV=(R1),LOC=(BELOW,64) DECB virtual addr below the line, but real above
         ST    R1,AREAAD
         LR    R5,R1
* Loop to build DECB's and issue first READ's.
BLDLOOP  MVC   0(NEXTDECBOFF,R5),MODELDECB    Create basic DECB
         LA    R1,0(R4,R5)                    Point to next DECB
         ST    R1,NEXTDECBOFF(,R5)            Set chain pointer to next DECB
         READ  (R5),SF,,DATAOFF(R5)           Store data address in DECB and issue READ
         AR    R5,R4                          Point to next DECB
         BCT   R3,BLDLOOP                     Branch if another READ to issue
         SR    R5,R4                          Point back to last DECB
         L     R1,AREAAD                      Point to first DECB
         ST    R1,NEXTDECBOFF(,R5)            Point last DECB to first DECB
         LR    R5,R1                          Point to first (oldest) DECB
* Loop to read until end-of-file is reached.
MAINLOOP CHECK (R5)                           Wait for READ, branch to EODATA if no more
         L     R1,16(,R5)                     Point to status area
         LH    R0,DCBBLKSI                    Get length of read attempted
 (2)     SH    R0,14(,R1)                     Subtract residual count to get length read
RECORD1  LA    R1,DATAOFF(,R5)                 Point to first record in block
          .
          .    (Process records in block)
          .
         READ  (R5),SF,MF=E                   Issue new read
         L     R5,NEXTDECBOFF(,R5)            Point to next DECB
         B     MAINLOOP                       Branch to next block
* End-of-data.  CHECK branched here because DECB was for a READ after the last block.
```

Figure 84. Reading a Member of a PDS or PDSE using Asynchronous BPAM (Part 1 of 2)

```
EODATA   CLOSE LIBDCB
         L     R0,AREALEN
         L     R1,AREAAD
         FREEMAIN R,LV=(0),A=(1)
          .
          .
          .
AREAAD   DC    A(0)                                Address of gotten storage
AREALEN  DC    F'0'                                Length of gotten storage
LIBDCB   DCB   DSORG=PO,DCBE=MYDCBE,MACRF=R,DDNAME=DATA
MYDCBE   DCBE  MULTSDN=2,EODAD=EODATA,MULTACC=1   Request OPEN to supply NCP
         READ  MODELDECB,SF,LIBDCB,MF=L
NEXTDECBOFF EQU  *-MODELDECB                       Offset to addr of next DECB
DATAOFF  EQU   NEXTDECBOFF+4                       Offset to data
MEMNAME  DC    CL8'MASTER'                         Name of member to read
         DCBD  DSORG=PS,DEVD=DA
         IHADCBE ,
```

Figure 84. Reading a Member of a PDS or PDSE using Asynchronous BPAM (Part 2 of 2)

Tip: You can convert Figure 84 on page 422 to use LBI by making the following changes:

- Add BLKSIZE=0 in the DCBE macro. Coding a nonzero value also requests LBI, but it overrides the block size.
- After line (1), test whether the access method supports LBI. This is in case the type of data set or the level of operating system does not support LBI. Insert these lines to get the maximum block size:

```
TM    DCBEFLG1,DCBESLBI                  Branch if access method does
BZ    ROUND                                not support LBI
L     R1,DCBEBLKSI                       Get maximum size of a block
```

- After line (2) get the size of the block:

```
TM    DCBEFLG1,DCBESLBI                 Branch if
BZ    RECORD1                            not using LBI
SH    R1,=X'12'                        Point to size
L     R0,0(,R1)                         Get size of block
```

Modifying a PDS

A member of a PDS can be updated in place, or it can be deleted and rewritten as a new member.

Updating in Place

A member of a PDS can be updated in place. Only one user can update at a time. When you update-in-place, you read records, process them, and write them back to their original positions without destroying the remaining records. The following rules apply:

- You must specify the UPDAT option in the OPEN macro to update the data set. To perform the update, you can use only the READ, WRITE, GET, PUTX, CHECK, NOTE, POINT, FIND, BLDL, and STOW macros.
- You cannot update concatenated data sets.
- You cannot delete any record or change its length; you cannot add new records.
- You do not need to issue a STOW macro unless you want to change the user data in the directory entry.
- You cannot use LBI.

With BSAM and BPAM

A record must be retrieved by a READ macro before it can be updated by a WRITE macro. Both macros must be execute forms that refer to the same DECB; the DECB must be provided by a list form. (The execute and list forms of the READ and WRITE macros are described in *z/OS DFSMS Macro Instructions for Data Sets*.)

With Overlapped Operations

To overlap I/O and processor activity, you can start several read or write operations before checking the first for completion. You cannot overlap read and write operations. However, as operations of one type must be checked for completion before operations of the other type are started or resumed. Note that each outstanding read or write operation requires a separate DECB. If a single DECB were used for successive read operations, only the last record read could be updated.

In Figure 85 on page 425, overlap is achieved by having a read or write request outstanding while each record is being processed.

```
//PDSDD  DD      DSNAME=MASTFILE(MEMBERK),DISP=OLD,---
         ...
UPDATDCB DCB     DSORG=PS,DDNAME=PDSDD,MACRF=(R,W),NCP=2,EODAD=FINISH
         READ    DECBA,SF,UPDATDCB,AREAA,MF=L           Define DECBA
         READ    DECBB,SF,UPDATDCB,AREAB,MF=L           Define DECBB
AREAA    DS      ---                                    Define buffers
AREAB    DS      ---
         ...
         OPEN    (UPDATDCB,UPDAT)                 Open for update
         LA      2,DECBA                          Load DECB addresses
         LA      3,DECBB
READRECD READ    (2),SF,MF=E                      Read a record
NEXTRECD READ    (3),SF,MF=E                      Read the next record
         CHECK   (2)                              Check previous read operation
```

(If update is required, branch to R2UPDATE)

```
        LR      4,3                           If no update is required,
        LR      3,2                           switch DECB addresses in
        LR      2,4                           registers 2 and 3
        B       NEXTRECD                      and loop
```

In the following statements, 'R2' and 'R3' refer to the records that were read using the DECBs whose addresses are in registers 2 and 3, respectively. Either register can point to either DECBA or DECBB.

```
R2UPDATE CALL    UPDATE,((2))                     Call routine to update R2

* Must issue CHECK for the other outstanding READ before switching to WRITE.
* Unfortunately this CHECK can send us to EODAD.

         CHECK   (3)                              Check read for next record
         WRITE   (2),SF,MF=E                      (R3) Write updated R2
```

(If R3 requires an update, branch to R3UPDATE)

```
         CHECK   (2)                              If R3 requires no update,
         B       READRECD                         check write for R2 and loop
R3UPDATE CALL    UPDATE,((3))                     Call routine to update R3
         WRITE   (3),SF,MF=E                      Write updated R3
         CHECK   (2)                              Check write for R2
         CHECK   (3)                              Check write for R3
         B       READRECD                         Loop
FINISH   EQU     *                                End-of-Data exit routine
```

(If R2 was not updated, branch to CLOSEIT)

```
         WRITE   (2),SF,MF=E                      Write last record read
         CHECK   (2)
CLOSEIT  CLOSE   (UPDATDCB)
```

Figure 85. Updating a Member of a PDS

Note the use of the execute and list forms of the READ and WRITE macros, identified by the parameters MF=E and MF=L.

With QSAM

Update a member of a PDS using the locate mode of QSAM (DCB specifies MACRF=(GL,PL)) and using the GET and PUTX macros. The DD statement must specify the data set and member name in the DSNAME parameter. This method permits only the updating of the member specified in the DD statement.

Rewriting a Member

There is no actual update option that can be used to add or extend records in a PDS. If you want to extend or add a record within a member, you must rewrite the complete member in another area of the data set. Because space is allocated when the data set is created, there is no need to request additional space. Note, however, that a PDS must be contained on one volume. If sufficient space has not been allocated, the data set must be reorganized by the IEBCOPY utility program or ISPF.

When you rewrite the member, you must provide two DCBs, one for input and one for output. Both DCB macros can refer to the same data set, that is, only one DD statement is required.

Concatenating PDSs

Two or more PDSs can be automatically retrieved by the system and processed successively as a single data set. This technique is known as concatenation. Two types of concatenation are: sequential and partitioned.

Sequential Concatenation

Sequentially concatenated data sets are processed with a DCB that has DSORG=PS, and can include sequential data sets, and members of PDSs and PDSEs. See “Concatenating Data Sets Sequentially” on page 379 for the rules for concatenating *like* and *unlike* data sets.

You can use sequential concatenation (DSORG=PS in DCB) to sequentially read directories of PDSs and PDSEs. See “Reading a PDS Directory Sequentially” on page 427 .

Partitioned Concatenation

Concatenated PDSs are processed with a DSORG=PO in the DCB. When PDSs are concatenated, the system treats the group as a single data set. A partitioned concatenation can contain a mixture of PDSs and PDSEs. Each PDS can hold up to 16 extents. Each PDSE is treated as if it had one extent, although it might have multiple extents on DASD.

The maximum number of PDSs that you can concatenate depends on the number of extents in the data sets. The total number of extents cannot exceed 120. For example, you can concatenate a maximum of 114 single-extent data sets, but you cannot concatenate 8 data sets with 16 extents each because 8 times 16 is 128. Use the following algorithm to determine if you can open a partitioned concatenation successfully:

```
                (100) + (16n) + (k-1)
where:
   n = the number of DASD extents (count a PDSE as one extent)
   k = the number of data sets in the concatenation
```

The result of this calculation cannot exceed 2040, or the system will be unable to open the partitioned concatenation.

Partitioned concatenation is supported only when the DCB is open for input.

Concatenated PDSs are always treated as having like attributes, except for block size. They use the attributes of the first data set only, except for the block size. BPAM OPEN uses the largest block size among the concatenated data sets. All

attributes of the first data set are used, even if they conflict with the block size parameter specified. For concatenated format-F data sets (blocked or unblocked), the LRECL for each data set must be equal.

You process a concatenation of PDSs the same way you process a single PDS, with one exception: you must use the FIND macro to begin processing a member. You cannot use the POINT (or NOTE) macro until after the FIND macro has been issued for the appropriate member. If two members of different data sets in the concatenation have the same name, the FIND macro determines the address of the first one in the concatenation. You would not be able to process the second one in the concatenation. The BLDL macro provides the concatenation number of the data set to which the member belongs in the K field of the BLDL list. (See "BLDL—Construct a Directory Entry List" on page 413.)

Reading a PDS Directory Sequentially

You can read a PDS directory sequentially just by opening the data set to its beginning (without using positioning macros) and reading it.

- The DD statement must identify the DSNAME without a member name.
- You can use either BSAM or QSAM with MACRF=R or G.
- Specify BLKSIZE=256 and RECFM=F or RECFM=U.
- QSAM always requires LRECL=256
- You should test for the last directory entry (8 bytes of X'FF'). Records and blocks after that point are unpredictable. After reading the last allocated directory block, control passes to your EODAD routine or reading continues with a concatenated data set. You can issue an FEOV macro to cease reading the current data set and continue with the next one. If you issue FEOV while reading the last or only data set, control passes to your EODAD routine.
- If you also want to read the keys (the name of the last member in that block), use BSAM and specify KEYLEN=8.

This technique works when PDSs and PDSEs are concatenated. The system considers this to be a *like* sequential concatenation. See "Reading a PDSE Directory" on page 466.

Chapter 28. Processing a Partitioned Data Set Extended (PDSE)

This chapter covers the following topics.

Topic	Page
Advantages of PDSEs	429
Structure of a PDSE	431
Processing PDSE Records	434
Allocating Space for a PDSE	437
Defining a PDSE	440
Creating a PDSE Member	441
Processing a Member of a PDSE	444
Retrieving a Member of a PDSE	458
Sharing PDSEs	460
Modifying a Member of a PDSE	464
Reading a PDSE Directory	466
Concatenating PDSEs	466
Converting PDSs to PDSEs and Back	467

Advantages of PDSEs

This section compares PDSEs to PDSs.

A PDSE is a data set divided into sequentially organized members, each described by one or more directory entries. PDSEs are stored only on direct access storage devices. In appearance, a PDSE is similar to a PDS. For accessing a PDS directory or member, most PDSE interfaces are indistinguishable from PDS interfaces. However, PDSEs have a different internal format, which gives them increased usability. Each member name can be eight bytes long. The primary name for a program object can be eight bytes long. Alias names for program objects can be up to 1024 bytes long. The records of a given member of a PDSE are written or retrieved sequentially.

You can use a PDSE in place of a PDS to store data, or to store programs in the form of program objects. A program object is similar to a load module in a PDS. A load module cannot reside in a PDSE and be used as a load module. One PDSE cannot contain a mixture of program objects and data members. For more information about load modules and program objects see *z/OS DFSMS Program Management*.

PDSEs and PDSs are processed using the same access methods (BSAM, QSAM, BPAM) and macros but you cannot use EXCP because of the data set's internal structures. See "Processing a Member of a PDSE" on page 444 for information about macros used with PDSEs, and *z/OS DFSMS Macro Instructions for Data Sets* for information about macros. See "Chapter 5. Protecting Data Sets" on page 55 for information about using RACF to protect PDSEs.

© Copyright IBM Corp. 1987, 2002

Processing a Partitioned Data Set Extended (PDSE)

PDSEs have several features that improve both your productivity and system performance. The main advantage of using a PDSE over a PDS is that PDSEs automatically reuse space within the data set without anyone having to periodically run a utility to reorganize it. See “Rewriting a Member” on page 426. The size of a PDS directory is fixed regardless of the number of members in it, while the size of a PDSE directory is flexible and expands to fit the members stored in it. Also, the system reclaims space automatically whenever a member is deleted or replaced, and returns it to the pool of space available for allocation to other members of the same PDSE. The space can be reused without having to do an IEBCOPY compress. Figure 86 shows these advantages.

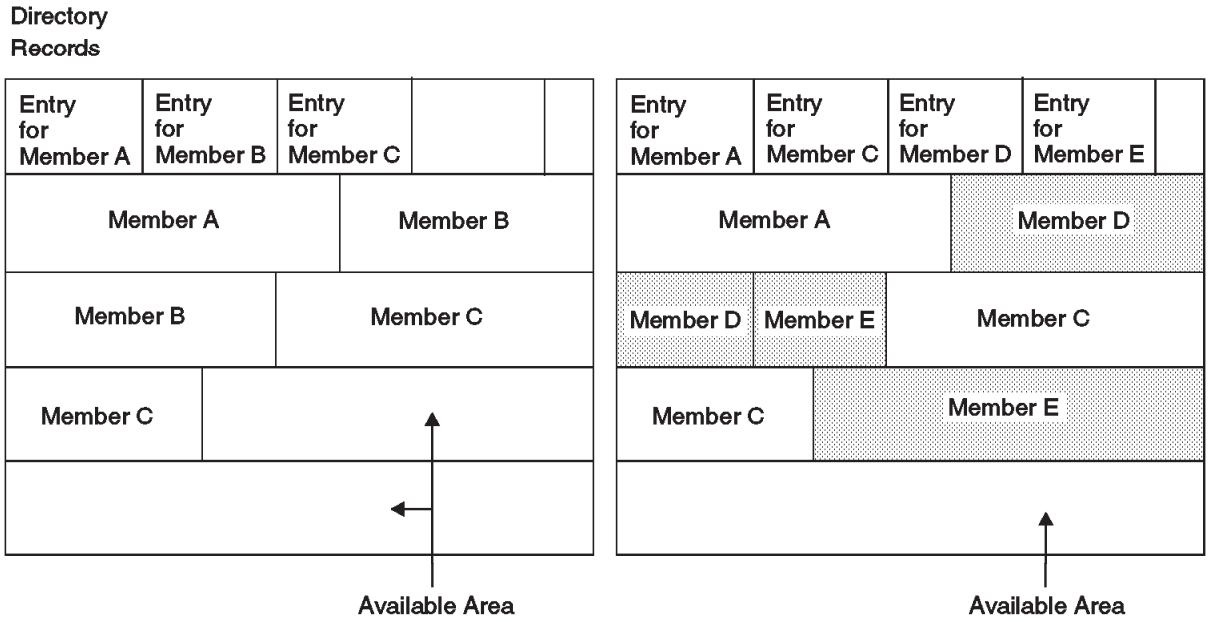


Figure 86. A Partitioned Data Set Extended (PDSE)

In Figure 86, when member B is deleted, the space it occupied becomes available for reuse by new members D and E.

Other advantages of PDSEs follow:

- PDSE members can be shared. This makes it easier to maintain the integrity of the PDSE when modifying separate members of the PDSE at the same time.
- Reduced directory search time. The PDSE directory, which is indexed, is searched using that index. The PDS directory, which is organized alphabetically, is searched sequentially. The system might cache in storage directories of frequently used PDSEs.
- Creation of multiple members at the same time. For example, you can open two DCBs to the same PDSE and write two members at the same time.
- PDSEs contain up to 123 extents. An extent is a continuous area of space on a DASD storage volume, occupied by or reserved for a specific data set.
- When written to DASD, logical records are extracted from the user’s blocks and reblocked. When read, records in a PDSE are reblocked into the block size specified in the DCB. The block size used for the reblocking can differ from the original block size.

PDSE and PDS Similarities

The significant similarities between PDSEs and PDSs are as follows:

- The same access methods and macros are used, with minor incompatibilities in some cases.
- Records are stored in members; members are described in the directory.

PDSE and PDS Differences

Table 30 shows the significant differences between PDSEs and PDSs:

Table 30. PDSE and PDS Differences

PDSE Characteristics	PDS Characteristics
Data set has a 123-extent limit.	Data set has a 16-extent limit.
Directory is expandable and indexed by member name; faster to search directory.	Fixed size directory is searched sequentially.
PDSEs are device independent: records are reblockable and the TTR is simulated as a system key.	For PDSs, TTR addressing and block sizes are device dependent.
Uses dynamic space allocation and automatically reclaims space. Using IEBCOPY or DFSMSdss COPY to copy all members of a PDSE creates a larger index than copying members one at a time. **Recommendation:** Allocate a PDSE with secondary space to permit the dynamic variation in the size of the PDSE index.	Must use IEBCOPY COMPRESS to reclaim space.
You can create multiple members at the same time.	You can create one member at a time.
PDSEs contain either program objects or data members but not both.	Whereas PDSs contain data members or load modules but there is no attempt to keep data members and load modules from being created in the same PDS.
Replacing a member without replacing all of its aliases causes the alias entries to be deleted.	Replacing a member without replacing all of its aliases causes the alias entries to be abandoned. They are ″orphans″ and eventually might point to the wrong member.

Alias names for program objects, in a PDSE, can be up to 1024 bytes long. To access these names, you must use the DESERV interfaces. Primary names for program objects are restricted to 8 bytes. All names for PDS members must be 8 bytes long.

Structure of a PDSE

When accessed sequentially, through BSAM or QSAM, the PDSE directory appears to be constructed of 256-byte blocks containing sequentially ordered entries. The PDSE directory looks like a PDS directory even though its internal structure and block size are different. PDSE directory entries vary in length. Each directory entry contains the member name or an alias, the starting location of the member within the data set and optionally user data. The directory entries are arranged by name in alphanumeric collating sequence.

Processing a Partitioned Data Set Extended (PDSE)

You can use BSAM or QSAM to read the directory sequentially. The directory is searched and maintained by the BLDL, DESERV, FIND, and STOW macros. If you use BSAM or QSAM to read the directory of a PDSE which contains program objects with names longer than 8 bytes, directory entries for these names will not be returned. If you need to be able to view these names, you must use the DESERV FUNC=GET_ALL interface instead of BSAM or QSAM. Similarly, the BLDL, FIND, and STOW macro interfaces allow specification of only 8-byte member names. These are analogous DESERV functions for each of these interfaces to allow for processing names greater than 8 bytes. See “PDS Directory” on page 406 for a description of the fields in a PDSE directory entry.

The PDSE directory is indexed, permitting more direct searches for members. Hardware-defined keys are not used to search for members. Instead, the name and the relative track address of a member are used as keys to search for members. The TTRs in the directory can change if you move the PDSE, since for PDSE members the TTRs are not relative track and record numbers but rather pseudo randomly generated aliases for the PDSE member. These TTRs may sometimes be referred to as Member Locator Tokens (MLTs).

The limit for the number of members in a PDSE directory is 522,236. The PDSE directory is expandable; you can keep adding entries up to the directory’s size limit or until the data set runs out of space. The system uses the space it needs for the directory entries from storage available to the data set.

For a PDS, the size of the directory is determined when the data set is initially allocated. There can be fewer members in the data set than the directory can contain, but when the preallocated directory space is full, the PDS must be copied to a new data set before new members can be added.

PDSE Logical Block Size

The significance of the block size keyword (BLKSIZE) is slightly different from a PDS. All PDSEs are stored on DASD as fixed 4 KB blocks. These 4 KB physical blocks are also known as pages. The PDSE is logically reblocked to the block size you specify when you open the data set. The block size does not affect how efficiently a PDSE is stored on DASD, but it can influence how efficiently the system reads from it and writes to it. The block size also affects the size of the storage buffer allocated for a PDSE. You should let the system-determined block size function calculate the best block size for your data set.

Reuse of Space

When a PDSE member is updated or replaced, it is written in the first available space. This is either at the end of the data set or in a space in the middle of the data set marked for reuse. This space need not be contiguous. The objective of the space reuse algorithm is not to extend the data set unnecessarily.

With the exception of UPDATE, a member is never immediately written back to its original space. The old data in this space is available to programs that had a connection to the member before it was rewritten. The space is marked for reuse only when all connections to the old data are dropped. However, once they are dropped, there are no pointers to the old data, so no program can access it. A connection may be established at the time a PDSE is opened, or by BLDL, FIND, POINT, or DESERV. These connections remain in effect until the program closes the PDSE or the connections are explicitly released by issuing DESERV FUNC=RELEASE, STOW disconnect, or (under certain cases) another POINT or FIND. Pointing to the directory can also release connections. Connections are

dropped when the data set is closed. For more information about connections see *z/OS DFSMS Macro Instructions for Data Sets*.

Directory Structure

Logically, a PDSE directory looks the same as a PDS directory. It consists of a series of directory records in a block. Physically, it is a set of pages at the front of the data set, plus additional pages interleaved with member pages. Five directory pages are initially created at the same time as the data set. New directory pages are added, interleaved with the member pages, as new directory entries are required. A PDSE always occupies at least five pages of storage.

Relative Track Addresses (TTR)

The starting location of each member is recorded by the system as a relative track address (TTR). The TTRs do not represent the actual track or record location. Instead, the TTRs are tokens that simulate the track and record location also known as Member Locator Tokens (MLT) and Record Locator Tokens (RLT).

TTRs used with PDSEs have the following format:

- TTRs represent individual logical records, not blocks or spanned record segments.
- The TTR for the PDSE directory is X'000001'.
- TTRs for PDSE members are randomly generated to be unique within a PDSE. TTRs for members that are deleted can be reused by the system for newly created members.
- TTRs for PDSE members range from X'000002' to X'07FFFF'.
- The TTR of a block is the record number for the first logical record of that block. TTRs for a block are unique within a member, but not unique within the PDSE.
- Record numbers start at X'100001' within a PDSE member.
- Record numbers range from X'100001' to X'FFFFFF'.
- Record numbers are contiguous within a PDSE member.

While the preceding notes can be used to define an algorithm for calculating PDSE TTRs, it is strongly recommended that you not do TTR calculations because this algorithm might change with new releases of the system.

Figure 87 shows examples of TTRs for unblocked records.

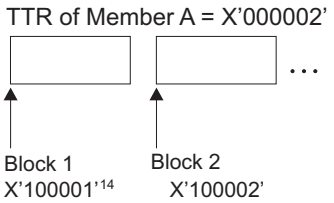


Figure 87. TTRs for a PDSE Member (Unblocked Records)

Figure 88 on page 434 shows examples of TTRs for blocked records.

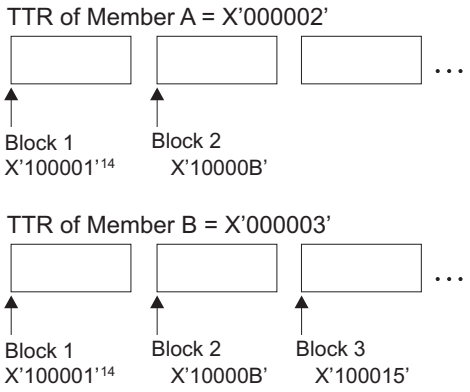


Figure 88. TTRs for Two PDSE Members (LRECL=80, BLKSIZE=800)

In both examples, PDSE member `A` has a TTR of X'000002'. In Figure 87, the records are unblocked, the record number[1] for logical record 1 is X'100001' and logical record 2 is X'100002'.

In Figure 88, the records are fixed length, blocked with LRECL=80 and BLKSIZE=800. The first block is identified by the member TTR, the second block by a TTR of X'10000B', and the third block by a TTR of X'100015'. Note that the TTRs of the blocks differ by an amount of 10, which is the blocking factor.

To position to a member, use the TTR obtained from the BLDL or NOTE macro, or a BSAM read of the directory, or DESERV FUNC=GET or FUNC=GET_ALL. To locate the TTR of a record within a member, use the NOTE macro (see “Using the NOTE Macro to Provide Relative Position” on page 455).

Processing PDSE Records

PDSE members are accessed sequentially, as are PDS members. Each member can contain a maximum number of 15 728 639 logical records. A logical end-of-file mark is at the end of each PDSE member.

Some restrictions on processing PDSEs follow:

- The JCL keyword DSNTYPE cannot be specified with the JCL keyword RECORG.
- You should not use PDSEs if your application is dependent on processing short blocks other than the last one in a member of processing or SAM record null segments. See “Processing Short Blocks” on page 436 and “Processing SAM Null Segments” on page 437 for more information.
- See “Using the BSP Macro to Backspace a Physical Record” on page 446 for the restrictions in using BSP with variable spanned or variable blocked spanned records.
- You cannot write or update the PDSE directory using the WRITE or PUT macros. To write or update the directory, you must use the STOW or DESERV FUNC=UPDATE macros.
- You cannot add or replace members of a PDSE program library using the STOW macro.
- Aliases for members must point to the beginning of the member.

1. The first record in a member can be pointed to using the TTR for the member (in the examples above, X'000002').

- The deletion of the primary member name causes all aliases to be deleted.
- EXCP, EXCPVR, and XDAP are not supported macros for PDSEs.
- If you allocate a PDSE, it cannot be read on an earlier version or release of DFSMSdfp that does not support PDSEs.
- Note lists are not supported for PDSEs. When using STOW with a PDSE, do not supply a list of TTRs in the user data field of the directory.
- The CHECK, STOW, and CLOSE macros do not guarantee that the data has been synchronized to DASD. Use the SYNCDEV macro or the storage class parameter SYNCDEV=YES to guarantee synchronizing data when open for update. See “Using the SYNCDEV Macro to Synchronize Data” on page 483 for the definition of synchronization.
- DS1LSTAR field of the format 1 DSCB is unpredictable for PDSEs.
- The OPTCD=W DCB parameter (write-check) is ignored for PDSEs.
- A checkpoint data set cannot be a PDSE. Checkpoint fails if the checkpoint data set is a PDSE. Also, a failure occurs if a checkpoint is requested when a DCB is opened to a PDSE, or if a PDS was opened at checkpoint time but was changed to a PDSE by restart time.
- Do not use the TRKCALC macro because results could be inaccurate. (However, no error indication is returned.)
- A library that contains cataloged procedures must be either a PDS or PDSE. The system procedure library, SYS1.PROCLIB, must be a PDS. The installation can have other procedure libraries, which can be PDSEs.
- When a cataloged procedure library is used, the library is opened for input and stays open for extended periods. If another user or system attempts to open the procedure library for output in a non-XCF (sysplex) environment, the second user receives an ABEND. This could prevent procedure library updates for extended periods.

See *z/OS DFSMS Migration* for more information about PDSE processing restrictions.

Using BLKSIZE with PDSEs

When reading a PDSE directory that uses fixed-length blocked records, you can specify a BLKSIZE value of 256 or greater (the system ignores the LRECL value). You can calculate the block size and specify it in the BLKSIZE parameter of the DCB, or you can let the system determine the block size for you. Although the block size makes no difference in space usage for PDSEs, block size affects the buffer size of a job using the PDSE.

See “Block Size (BLKSIZE)” on page 322 for information about using BLKSIZE. Also see “Reading a PDSE Directory” on page 466.

Using KEYLEN with PDSEs

For reading a PDSE directory, a key length of 0 or 8 are the only values allowed. You can use keys for reading PDSE members, but not for writing PDSE members. See “Key Length (KEYLEN)” on page 328 for information about using the KEYLEN parameter.

Reblocking PDSE Records

PDSE members are reblockable. When reading the PDSE members, the system constructs the block size specified in the DCB. If the block size is not specified in the DCB, the default block size is used. The system packs as many records as can

fit into the specified block size. Logical records are written to DASD in blocks whose lengths are not determined by the user, and can span physical records. The user-defined or system-defined block size is saved in the data set label when the records are written, and becomes the default block size for input. These constructed blocks are called simulated blocks.

Figure 89 shows an example of how the records are reblocked when the PDSE member is read:

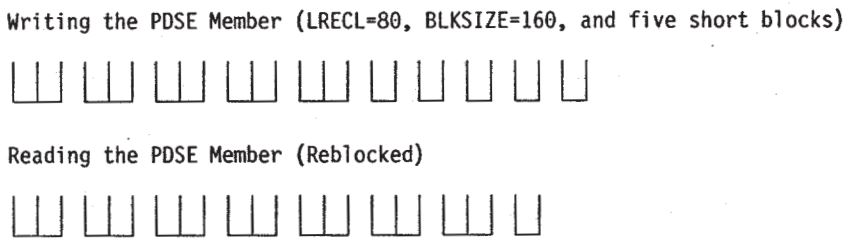


Figure 89. Example of How PDSE Records Are Reblocked

Suppose you create a PDSE member that has a logical record length of 80 bytes, such that you write five blocks with a block size of 160 (blocking factor of 2) and five short blocks with a block size of 80. Five blocks with a block size of 160 (blocking factor of 2) and five short blocks with a block size of 80. When you read back the PDSE member, the logical records are reblocked into seven 160-byte simulated blocks and one short block. Note that short block boundaries are not saved on output.

You also can change the block size of records when reading the data set. Figure 90 shows how the records are reblocked when read:

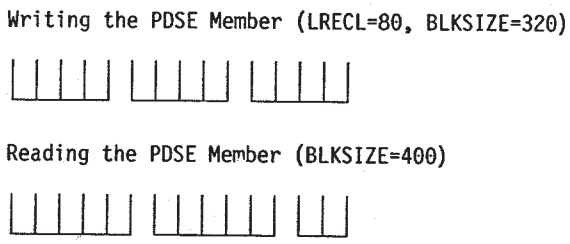


Figure 90. Example of Reblocking When the Block Size Has Been Changed

Suppose you write three blocks with block size of 320, and the logical record length is 80 bytes. Then if you read this member with a block size of 400 (blocking factor of 5), the logical records are reblocked into two 400-byte simulated blocks and one 160-byte simulated block.

If the data set was a PDS, you would not be able to change the block size when reading the records.

Processing Short Blocks

You should not use PDSEs if your application expects short blocks before the end of a member. You can create short blocks for PDSE members, but their block boundaries are not saved when the data set is written. For example, if you use the TRUNC macro with QSAM to create short blocks, the short blocks are not shown

when the data set is read. If the QSAM TRUNC macro is used, a message is written to the job log and an indicator is set in the SMF record type 15. A BSAM or BPAM TRUNC macro does not have this effect.

Processing SAM Null Segments

You should not use PDSEs if your application processes SAM null record segments. Null record segments, which are created only with variable blocked spanned (VBS) records, are not saved when the data set is written. For example, if you create a null record segment when writing the data set, it is not returned when the data set is read. On the first write of a null record segment, a message is written to the job log and an indicator is set in the SMF record type 15. See *z/OS MVS System Management Facilities (SMF)* for more information about SMF.

Allocating Space for a PDSE

Space allocation for a PDSE is slightly different from a PDS.

This section shows how to use the SPACE JCL keyword to allocate primary and secondary storage space amounts for a PDSE. The PDSE directory can extend into secondary space. A PDSE can have a maximum of 123 extents. A PDSE cannot extend beyond one volume. Note that a fragmented volume might use up extents more quickly because you get less space with each extent. With a SPACE=(CYL,(1,1,1)) specification, the data set can extend to 123 cylinders (if space is available).

Because PDSE directory space is allocated dynamically, you do not need to estimate the number of PDSE members to be created. Therefore, the directory block quantity you specify on the SPACE keyword is not used for PDSEs, but is saved and available for use by conversion utilities. Alternately, you can omit the directory quantity by specifying DSORG=PO in your JCL DD statement or the data class when allocating the data set.

Finding More Information about Space Allocation:

- See “Allocating Space for a PDS” on page 409 for examples of the SPACE keyword.
- See “Chapter 3. Allocating Space on Direct Access Volumes” on page 37, *z/OS MVS JCL User’s Guide*, and *z/OS MVS JCL Reference* for information about allocating space.

PDSE Space Considerations

Several factors affect space utilization in a PDSE. Compared to a PDS, there are both advantages and disadvantages in how a PDSE uses space. However, when considering the space differences, remember that a PDSE has the following functional advantages. A PDSE does not need to be compressed, it has an expandable directory so that space planning is less critical, and it can be shared more efficiently.

The following are some areas to consider when determining the space requirements for a PDSE.

Use of Noncontiguous Space

When members are created, the first available space is allocated. However, noncontiguous space can be used if contiguous space is not available. This can be

reclaimed space from deleted members. This is a clear advantage over PDSs, which require all member space to be contiguous and do not automatically reclaim space from deleted members.

Integrated Directory

All PDSE space is available for either directory or member use. Within a data set, there is no difference between pages used for the directory and pages used for members. As the data set grows, the members and directory have the same space available for use. The directory, or parts of it, can be in secondary extents.

Directory pages are no longer a factor in determining space requirements. A PDSE does not have preallocated, unused, directory pages. Directory space is automatically expanded as needed.

The format of a PDSE lets the directory contain more information. This information can take more space than a PDS directory block.

The PDSE directory contains keys (member names) in a compressed format. The insertion or deletion of new keys may cause the compression of other directory keys to change. Therefore the change in the directory size may be different than the size of the inserted or deleted record.

Full Block Allocation

A PDSE member is allocated in full page increments. A member is maintained on full page boundaries, and any remaining space in the page is unused. This unused space is inaccessible for use as other member space or directory storage.

PDSE Unused Space

PDS gas is the unreclaimed space in a PDS that was vacated when members were deleted or rewritten. Users often overallocate their PDSs to allow for the inevitable amount of PDS gas that would develop over time. With PDSEs, you do not need to add additional space to the allocation to allow for growth of the data set due to gas.

Studies show that a typical installation has 18% to 30% of its PDS space in the form of gas. This space is unusable to the data set until it has been compressed. A PDSE dynamically reuses all the allocated space according to a first-fit algorithm. You do not need to make any allowance for gas when you allocate space for a PDSE.

Space is only reclaimed for an OPEN for output when it is the only open for output on that system. PDSE space cannot be reclaimed immediately after a member is deleted or dated. If a deleted or updated member still has an existing connection from another task (or the input DCB from an ISPF edit session), the member space is not reclaimed until the connection is released and the data set is opened for output and that OPEN for OUTPUT is the only one on that system.

ABEND D37 can occur on a PDSE indicating it is FULL, but another member can still be saved in the data set. Recovery processing from an ABEND D37 in ISPF closes and reopens the data set. This new open of the data set allows PDSE code to reclaim space so a member can now be saved.

Frequency of Data Set Compression

Data set compression is the process by which unused space is removed from a PDS.

Data set compression is not necessary with a PDSE. Since there is no gas accumulation in a PDSE, there is no need for compression.

Extent Growth

A PDSE can have up to 123 extents. Because a PDSE can have more secondary extents, you can get the same total space allocation with a smaller secondary allocation. A PDS requires a secondary extent about eight times larger than a PDSE to have the same maximum allocation. Conversely, for a given secondary extent value, PDSEs can grow about eight times larger before needing to be condensed. Defragmenting is the process by which multiple small extents are consolidated into fewer large extents. This operation can be performed directly from the interactive storage management facility (ISMF) data set list.

Although the use of smaller extents can be more efficient from a space management standpoint, to achieve the best performance you should avoid fragmenting your data sets whenever possible.

Logical Block Size

The logical block size can affect the performance of a PDSE. In general a large block size improves performance. IBM recommends that you use the system-determined block size function for allocating data sets. It chooses the optimal block size for the data set. A PDSE is given the maximum block size of 32 760 bytes for variable records. For fixed-length records, the block size is the largest multiple of record size that is less than or equal to 32 760.

Applications can use different logical block sizes to access the same PDSE. The block size in the DCB is logical and has no effect on the physical block (page) size being used.

Physical Block Size (Page Size)

PDSEs use a physical block size of 4 KB, the same size as an MVS page. These 4 KB physical blocks are also often referred to as pages.

The DCB block size does not affect the physical block size and all members of a PDSE are assumed to be reblockable. If you code your DCB with BLKSIZE=6160, the data set is physically reblocked into 4 KB pages, but your program still sees 6160-byte logical blocks.

Free Space

The space for any library can be overallocated. This excess space can be released manually with the FREE command in ISPF. Or you could code the release (RLSE) parameter on your JCL or select a management class that includes the release option partial.

RLSE is complemented by the partial release parameter in the management class, which can cause DFSMShsm to release free space during the daily space management cycle or depend on the equivalent in the management class. This function works the same for PDSEs as it does for PDSs or sequential data sets. The partial release parameter frees the unused space but it does not consolidate extents.

Fragmentation

Most allocation units are approximately the same size. This is because of the way members are buffered and written in groups of multiple pages. There is very little, if any, fragmentation in a PDSE.

If there is fragmentation, copy the data set with IEBCOPY or DSS. The fragmented members are recombined in the new copy.

Summary of PDSE Storage Requirements

When planning for PDSE processing, consider the following storage requirements:

- The storage necessary for directory information is obtained from storage that is generally available to the data set. Because the directory expands dynamically, storage for the directory is obtained whenever needed. The directory storage need not be limited to the primary extent, but can be obtained from any available storage.
- For each PDSE member, a variety of information is retained in the directory (such as attributes, statistics, and status). The directory storage required to support any single member is variable, as is the storage required to support alias names. For a medium-sized PDSE containing approximately 150 members, approximately 12 pages (4096 bytes per page) of directory storage is required.
- Deleting of a PDSE member can, in some rare cases, actually increase the amount of directory space used by the remaining members. This can cause the total amount of directory space required to be larger after the delete.

  The only time this problem could occur is if nearly all the storage allocated for the PDSE is used. If a STOW delete or rename of a member requires that the PDSE be extended and the extend fails, STOW will return an error return code. This return code will indicate why additional DASD storage could not be allocated.

Defining a PDSE

This section shows how to define a PDSE. The DSNTYPE keyword defines either a PDSE or PDS. The DSNTYPE values follow:

- LIBRARY (defines a PDSE)
- PDS (defines a partitioned data set)

To define PDSE data set types, specify DSNTYPE=LIBRARY in a data class definition, a JCL DD statement, the LIKE keyword, the TSO ALLOCATE command, or the DYNALLOC macro.

Your storage administrator can assign DSNTYPE=LIBRARY in the SYS1.PARMLIB member IGDSMSxx as an installation default value. If the installation default is DSNTYPE=LIBRARY, specify DSORG=PO or DIR *space* in the JCL or data class definition to allocate the data set as a PDSE (you do not need to specify DSNTYPE in this case).

The following parameters are both required to allocate a PDSE:

- Specify DIR *space* (greater than zero) or DSORG=PO (partitioned organization) in the JCL, in the DYNALLOC macro, in the data class, or in the TSO ALLOCATE command.
- Specify DSNTYPE=LIBRARY in the JCL, in the data class, in the TSO ALLOCATE command, using the LIKE keyword, or as the installation default specified in SYS1.PARMLIB.

Recommendation: If you do not want to allocate the data set as a PDSE, but the data class definition set up in the ACS routine specifies DSNTYPE, override it in one of two ways:

- By specifying a data class without the DSNTYPE keyword (in the JCL DD statement or ISMF panel).
- By specifying DSNTYPE=PDS in the JCL DD statement, data class, LIKE keyword, or ALLOCATE command.

When you create a data set and specify the number of directory entries or DSORG=PO or the data class has DSORG=PO without being overridden, SMS chooses whether it will be a PDS or PDSE. SMS uses the first source of information in the following list:

- DSNTYPE=PDS or DSNTYPE=LIBRARY (for a PDSE) in JCL or dynamic allocation.
- DSNTYPE of PDS or LIBRARY in data class.
- Installation default (in IGDSMSxx member if SYS1.PARMLIB).
- PDS.

An error condition exists and the job is ended with appropriate messages if the DSNTYPE keyword was specified in the JCL, but the job runs on a processor with a release of MVS that does not support the JCL keyword DSNTYPE. Message IEF630I is issued.

Creating a PDSE Member

You can create PDSE members with BSAM, QSAM, or BPAM.

Creating a PDSE Member with BSAM or QSAM

If you do not need your program to add user data entries to the directory, you can write a member without using the STOW macro, as shown in Figure 91.

```
//PDSEDD DD    DSNAME=MASTFILE(MEMBERK),SPACE=(TRK,(100,5,7)),
//            DISP=(NEW,CATLG),DCB=(RECFM=FB,LRECL=80,BLKSIZE=80),
//            DSNTYPE=LIBRARY,STORCLAS=S1P01S01,---
          ...
          OPEN  (OUTDCB,(OUTPUT))
          ...
          PUT   OUTDCB,OUTAREA     Write record to member
          ...
          CLOSE (OUTDCB)           Automatic STOW
          ...
OUTAREA   DS    CL80               Area to write from
OUTDCB    DCB   ---,DSORG=PS,DDNAME=PDSEDD,MACRF=PM
```

Figure 91. Creating One Member of a PDSE

Processing a Partitioned Data Set Extended (PDSE)

You can use the same program to allocate either a sequential data set or a member of a PDS or PDSE with only a change to the JCL, as follows:

1. The PDSE might be system managed. Specify a STORCLAS in the DD statement for the PDSE, or let the ACS routines direct the data set to system-managed storage.
2. Code DSORG=PS in the DCB macro.
3. Specify in the DD statement that the system is to store the data as a member of a PDSE; that is, DSNAME=name(membername).
4. Either specify a data class in the DD statement or allow the ACS routines to assign a data class.
5. Use an OPEN macro, a series of PUT or WRITE macros, and then a CLOSE macro to process the member. When the data set is closed, the system issues a STOW macro.

As a result of these steps, the data set and its directory are created, the records of the member are written, and an entry is automatically made in the directory with no user data.

A PDSE becomes a PDSE program library when the binder stores the PDSE's first member.

Adding or Replacing PDSE Members Serially

To add additional members to the data set or replace members, follow the procedure described in Figure 91 on page 441. However a separate DD statement omitting the space request is required for each member. Specify the disposition as old or shared (DISP=OLD or SHR). You can process more than one member without closing and reopening the data set, as follows:

1. Code DSORG=PO in the DCB macro.
2. Use WRITE and CHECK to write and check the member records.
3. When all the member records have been written, issue a STOW macro to enter the member name, its location pointer, and any additional data in the directory.
4. Continue to use WRITE, CHECK, and STOW until all the members of the data set and the directory entries have been written.

The example in Figure 92 on page 443 shows how to process more than one PDSE or PDS member without closing and reopening the data set.

```
//PDSEDD DD    ---,DSN=MASTFILE,DISP=MOD,SPACE=(TRK,(100,5,7))
         ...
         OPEN  (OUTDCB,(OUTPUT))
         ...
**   WRITE MEMBER RECORDS

MEMBER   WRITE DECBX,SF,OUTDCB,OUTAREA    WRITE first record of member
         CHECK DECBX
*
         WRITE DECBY,SF,OUTDCB,OUTAREA   WRITE and CHECK next record
         CHECK DECBY
         ...                              WRITE/CHECK remaining records of member
*
         STOW  OUTDCB,STOWLIST,A       Enter the information in directory
*        ...                           for this member after all records are written.
Repeat from label "MEMBER" for each additional member, changing the
member name in the "STOWLIST" for each member.
         ...
         CLOSE (OUTDCB)                (NO automatic STOW)
         ...
OUTAREA  DS    CL80                    Area to write from
OUTDCB   DCB   ---,DSORG=PO,DDNAME=PDSEDD,MACRF=W
STOWLIST DS    0F                      List of member names for STOW
         DC    CL8'MEMBERA'            Name of member
         DS    CL3                     TTR of first record (created by STOW)
         DC    X'00'                   C byte, no user TTRNs, no user data
```

Figure 92. Adding PDSE Members Serially

The A option on STOW in Figure 92 means the members did not exist before. You can code R to replace or all members.

Adding or Replacing Multiple PDSE Members Concurrently

You can create PDSE members at the same time from multiple DCBs or jobs, as follows:

- Multiple DCBs (open for output) in the same job step
- Multiple jobs on the same central processing complex
- A combination of “1” and “2”.

Figure 93 shows you how to use BPAM to create multiple PDSE members at the same time.

```
         ...
         OPEN  (DCB1,(OUTPUT),DCB2,(OUTPUT))
         WRITE DECB1,SF,DCB1,BUFFER           Write record to 1st member
         CHECK DECB1
         ...
         WRITE DECB2,SF,DCB2,BUFFER           Write record to 2nd member
         CHECK DECB2
         ...
         STOW  DECB1,PARML1,R           Enter 1st member in the directory
         STOW  DECB2,PARML2,R           Enter 2nd member in the directory
         ...
DCB1     DCB   DSORG=PO,DDNAME=X, ...         Both DCBs open to the
DCB2     DCB   DSORG=PO,DDNAME=X, ...         same PDSE
```

Figure 93. Replacing Multiple PDSE Members Concurrently

The R option of STOW in Figure 93 on page 443 means you are adding new members or replacing members. You could code A to mean you are only adding new members.

Open two DCBs to the same PDSE, write the member records, and issue STOW for them. Code different names for the parameter list in the STOW macro for each member written in the PDSE directory.

Processing a Member of a PDSE

Your programs process PDSEs in the same manner as PDSs. To locate a member or to process the directory, several macros are provided by the operating system, and are discussed in this section.

PDSEs are designed to automatically reuse data set storage when a member is replaced. PDSs do not reuse space automatically. If a member is deleted or replaced, the old copy of the PDS or PDSE member remains available to applications that were accessing that member's data before it was deleted or replaced.

Establishing Connections to Members

A connection to a PDSE member provides a temporary version of that member. The connection lets the member remain available to applications that were accessing that member's data before it was deleted or replaced. Connections to PDSE members are established by:

- JCL using DSNAME=*libname*(*memname*). This connection occurs at OPEN.
- BLDL
- DESERV FUNC=GET
- DESERV FUNC=GET_ALL
- FIND by name
- FIND by TTR
- POINT

All connections established to members while a data set was opened are released when the data set is closed. If the connection was established by FIND by name, the connection is released when another member is connected through FIND or POINT. The system reclaims the space used when all connections for a specific member have been released.

If deleting or replacing a member, the old version of the member is still accessible by those applications connected to it. Any application connecting to the member by name (through BLDL, FIND, or OPEN) following the replace operation accesses the new version. (The replaced version cannot be accessed using a FIND by TTR or POINT unless a connection already exists for it.)

Connections established by OPEN, BLDL, FIND, and POINT are used by BSAM, QSAM, and BPAM for reading and writing member data. Connections established by DESERV are primarily used by program management. When your program or the system closes the DCB, the system drops all connections between the DCB and the data set. If you use BLDL, FIND by TTR, or POINT to connect to members, you can disconnect those members before closing the DCB by issuing STOW DISC. If you use DESERV to connect to members you can disconnect those members before closing the DCB by issuing DESERV FUNC=RELEASE.

Using the BLDL Macro to Construct a Directory Entry List

The BLDL macro reads one or more directory entries into virtual storage. Place member names in a BLDL list before issuing the BLDL macro. For each member name in the list, the system supplies the relative track address (TTR) and any additional information contained in the directory entry. Note that if there is more than one member name in the list, the member names must be in collating sequence, regardless of whether the members are from the same or different PDSs or PDSEs in the concatenation.

BLDL also searches a concatenated series of directories when (1) a DCB is supplied that is opened for a concatenated PDS or (2) a DCB is not supplied, in which case the search order begins with the TASKLIB, then proceeds to the JOBLIB or STEPLIB (themselves perhaps concatenated) followed by LINKLIB.

You can improve retrieval time by directing a subsequent FIND macro to the BLDL list rather than to the directory to locate the member to be processed.

Figure 78 on page 414 shows the BLDL list, which must begin with a 4-byte list description that specifies the number of entries in the list and the length of each entry (12 to 76 bytes). If you specify an option such as NOCONNECT or BYPASSLLA, an 8-byte BLDL prefix must precede the 4-byte list descriptor. The first 8 bytes of each entry contain the member name or alias. The next 6 bytes contain the TTR, K, Z, and C fields. The minimum directory length is 12 bytes.

The BLDL macro, unless the NOCONNECT option is specified, establishes a connection to each member of a PDSE when that member is found in the PDSE.

Like a BSAM or QSAM read of the directory, the BLDL NOCONNECT option does not connect the PDSE members. The BLDL NOCONNECT option causes the system to use less virtual storage. The NOCONNECT option is appropriate when BLDLs are issued for many members that might not be processed.

Do not use the NOCONNECT option if two applications will process the same member. For example, if an application deletes or replaces a version of a member and NOCONNECT was specified, that version is inaccessible to any application that is not connected.

For PDSE program libraries, you can direct BLDL to search the LINKLST, JOBLIB, and STEPLIB. Directory entries for load modules located in the link pack area (LPA) cannot be accessed by the BLDL macro.

Using the BSP Macro to Backspace a Physical Record

You can use the BSP macro to backspace the current member one simulated block. You can then reread or rewrite the simulated block. However, you cannot backspace beyond the start of a PDSE member nor backspace within the PDSE directory.

For variable spanned records (RECFM=VS), if positioned to the beginning of a record, the BSP macro backspaces to the start of the previous record. If positioned within a record, the BSP macro backspaces to the start of that record.

For variable blocked spanned (RECFM=VBS) records, the BSP macro backspaces to the start of the first record in the buffer just read. The system does not backspace within record segments. In other words, issuing the BSP macro followed by a read always begins the block with the first record segment or complete segment. (A block can contain more than one record segment.)

If you write in a PDSE member and then issue the BSP macro followed by a WRITE macro, you destroy all the data of the member beyond the record just written.

Using the Directory Entry Services

DESERV provides interfaces to access the directories of PDS and PDSE data sets. With DESERV you can get all the directory entries for a PDSE or selected directory entries for a PDS or a PDSE with the GET_ALL and GET functions, respectively. You can delete or rename a list of members in a PDSE with the DELETE and RENAME functions. You can alter the attributes of a program object (in a PDSE) with the UPDATE function.

All functions return results to the caller in areas provided by the invoker or areas returned by DE services. All functions provide status information in the form of return and reason codes. The IGWDES macro maps all the parameter areas.

DE services also introduces a new format directory entry called system-managed directory entry (SMDE), mapped by the IGWSMDE macro. The SMDE is an extended and extensible version of the directory entry produced by BLDL. Its chief features are that it provides users with long name support, longer application (user) data, load module indication and version control. Table 31 on page 447 describes all the functions provided by DESERV. This section discusses the GET, GET_ALL, RELEASE, GET_NAMES, and UPDATE functions. The DELETE and RENAME functions are described later.

Table 31. DE Services Function Summary

Function	Description
GET	Obtain directory entries based on a list of names, or a BLDL directory entry to establish connections to these members.
GET_ALL	Obtain all directory entries for each member name. Optionally, establish connections to all names in the library.
GET_NAMES	Obtain buffer filled with the names and directory entires of each member name (primary and aliases) defined to a specified program object member of a PDSE.
DELETE	DELETE one or more PDSE member names. For more information see “Deleting a Member” on page 465.
RELEASE	Remove selected connections established by one or more previous calls to the GET or the GET_ALL functions. Or, remove all connections established by a previous call to the GET or the GET_ALL function.
RENAME	Rename one or more PDSE members. For more information see “Renaming a Member” on page 465.
UPDATE	Let caller update select fields of the directory entries of PDSE program objects.

FUNC=GET

DESERV GET returns SMDEs for members of opened PDSs or PDSEs or a concatenation of PDSs or PDSEs. The data set can be opened for either input, output, or update. The SMDE contains the PDS or PDSE directory. The SMDE is mapped by the IGWSMDE macro and contains a superset of the information that is mapped by IHAPDS. The SMDE returned can be selected by name or by BLDL directory entry.

Input by Name List: If you want to select SMDEs by name, you supply a list of names that must be sorted in ascending order, without duplicates. Each name comprises a 2-byte length field followed by the characters of the name. When searching for names with less than eight characters, the names are padded on the right with blanks to make up eight characters. For each length field that contains a value greater than eight, DE services ignores trailing blanks and nulls beyond the eighth byte when doing the search.

In addition to retrieving the SMDE, member level connections can be established for each member name found. The members will be connected with the HOLD type connection. A connection type of HOLD insures that the member cannot be removed from the system until the connection is released. The connection intent is specified by the CONN_INTENT parameter, CONN_INTENT=HOLD must be specified.

All connections made through a single call to GET are associated with a single unique connect identifier. The connect identifier may be used to release all the connections in a single invocation of the RELEASE function. An example of DESERV GET is shown in Figure 94.

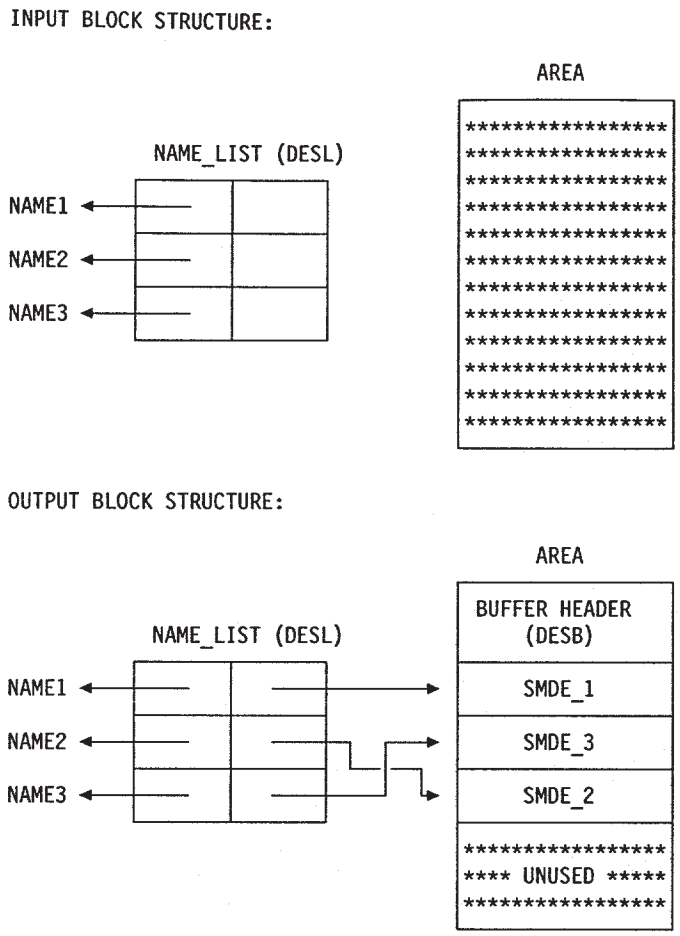


Figure 94. DESERV GET by NAME_LIST Control Block Structure

Input by BLDL Directory Entry (PDSDE): If the search argument specified is a PDSDE, the GET function is significantly restricted. The PDSDE (as mapped by the IHAPDS macro) identifies only one name to be searched for. Since the PDSDE also identifies the concatenation number of the library in which this member is to reside (PDS2CNCT), only that library can be searched. Since the PDSDE identifies a specific version of the member name (this identification is made through the MLT (PDS2TTRP)), the name can only be considered found if the same version can be found in the target library. However, a library search can only be performed if the target library is a PDSE. If the target library is a PDS, the input PDSDE will simply be converted to an equivalent directory entry in SMDE format and returned. No directory search can be performed. If the caller has specified BYPASS_LLA=NO, the library search will search LLA for LLA managed libraries. If the caller has specified BYPASS_LLA=YES, only the DASD directories of the library will be searched. An example of DESERV GET by PDSDE control block is shown in Figure 95.

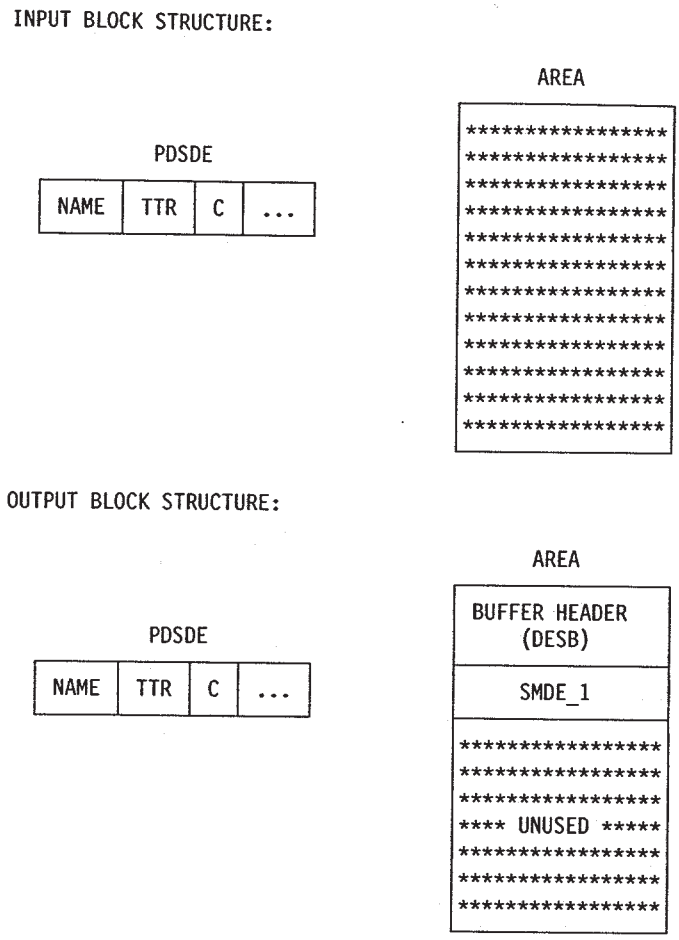


Figure 95. DESERV GET by PDSDE Control Block Structure

FUNC=GET_ALL

The GET_ALL function returns SMDEs for all the member names in a PDS, a PDSE, or a concatenation of PDSs and PDSEs. Member level connections can be established for each member found in a PDSE. A caller uses the CONCAT parameter to indicate which data set in the concatenation is to be processed, or whether all of the data sets in the concatenation are to be processed.

If the caller requests that DESERV GET_ALL return all the SMDE directory entries for an entire concatenation, the SMDEs are returned in sequence as sorted by the SMDE_NAME field without returning duplicate names. As with the GET function, all connections can be associated with a single connect identifier established at the time of the call. This connect identifier can then be used to release all the connections in a single invocation of the RELEASE function. See Figure 96 on page 450 for an overview of control blocks related to the GET_ALL function.

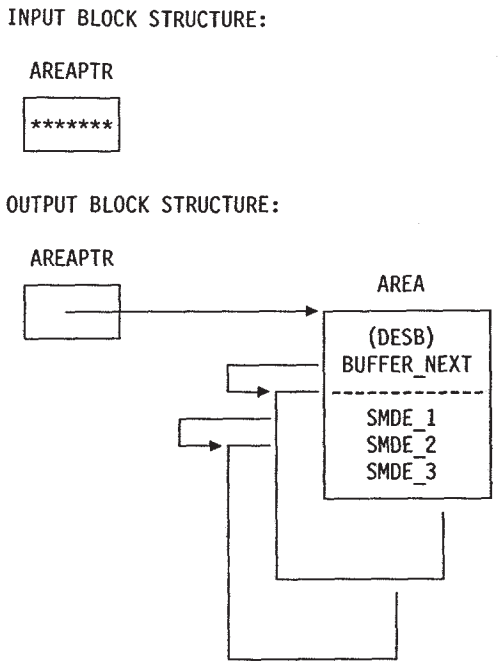


Figure 96. DESERV GET_ALL Control Block Structure

FUNC=GET_NAMES

The GET_NAMES function will obtain a list of all names and associated application data for a member of a new PDSE. This function does not support PDSs.

The caller provides a name or its alias name for the member as input to the function. The buffer is mapped by the DESB structure and is formatted by GET_NAMES. This function will return data in a buffer obtained by GET_NAMES. The data structure returned in the DESB is the member descriptor structure (DESD). The DESD_NAME_PTR field points to the member or alias name. The DESD_DATA_PTR points to the application data. For a data member, the application data is the user data from the directory entry. For a primary member name of a program object, the application data is mapped by the PMAR and PMARL structures of the IGWPMAR macro. For an alias name of a program object, the application data is mapped by the PMARA structure of the IGWPMAR macro. The DESB_COUNT field indicates the number of entries in the DESD, which is located at the DESB_DATA field. The buffer is obtained in a subpool as specified by the caller and must be released by the caller. If the caller is in key 0 and subpool 0 is specified, the DESB will be obtained in subpool 250.

See Figure 97 on page 451 for an overview of control blocks related to the GET_NAMES function.

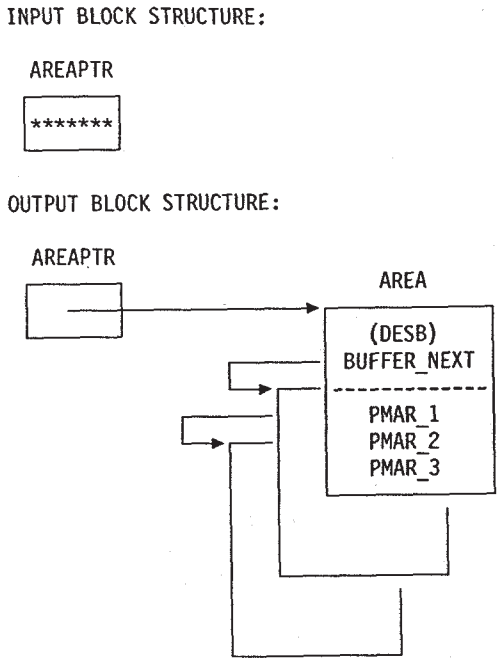


Figure 97. DESERV GET_NAMES Control Block Structure

FUNC=RELEASE

The RELEASE function can remove connections established through the GET and GET_ALL functions. The caller must specify the same DCB which was passed to DESERV to establish the connections. The connections established by the BLDL, FIND, or POINT macro are unaffected.

The caller can specify which connections are to be removed in one of two ways, either a connect id or a list of SMDEs by supplying. The function removes all connects from a single request of the GET or GETALL functions if the caller passes a connect identifier. Alternatively, if provided with a list of SMDEs, the function removes the connections associated with the versions of the member names in the SMDEs.

Recommendation: The SMDEs as returned by GET and GETALL contain control information used to identify the connection. Do not modify this information before issuing the RELEASE function.

If all connections of a connect identifier are released based on SMDEs, the connect identifier is not freed or reclaimed. Only release by connect identifier will cause DE services to reclaim the connect id for future use. It is not an error to include SMDEs for PDS data sets even though connections can't be established. It is an error to release an used connect identifier. It is also an error to release a PDSE SMDE for which there is no connection.

The DE services user does not need to issue the RELEASE function to release connections as all connections not explicitly released can be released by closing the DCB. See Figure 98 on page 452 for an overview of control blocks related to the RELEASE function.

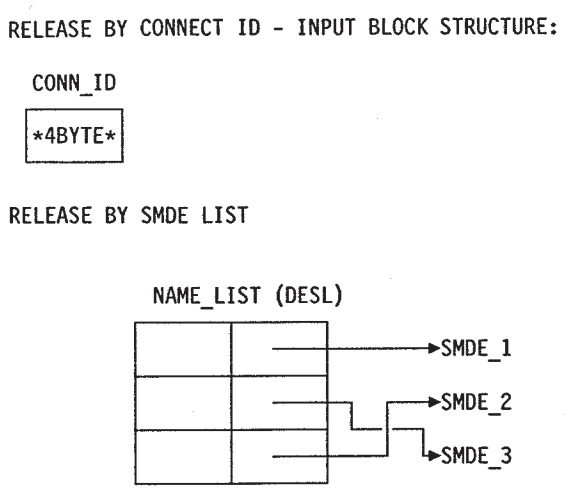


Figure 98. DESERV RELEASE Input Control Block Structure

FUNC=UPDATE

Update selected fields of the directory entry for a PDSE program object using the DESERV UPDATE function. This lets the caller update selected fields of the PMAR. The caller must supply a DCB that is open for output or update. The caller must also supply a DESL that points to the list of SMDEs to be updated. The DESL can be processed in sequence and a code can indicate successful and unsuccessful update. The SMDE (as produced by the GET function) contains the MLT and concatenation number of the member as well as an item number. These fields will be used to find the correct directory record to be updated. The DESL_NAME_PTR is ignored. The caller should issue a DESERV GET function call to obtain the SMDEs; modify the SMDEs as required; and issue a DESERV UPDATE function call to pass the updated DESL.

The UPDATE function does not affect the directory entry imbedded in the program object. This has implications for a binder inclusion of a PDSE program object as a sequential file. The binder can use the directory entry in the program object rather than the one in the directory.

You can update the fields in Figure 99 on page 453.

```
PMAR_RENT          1 BIT          Reenterable
PMAR_REUS          1 BIT          Reusable
PMAR_TEST          1 BIT          Module to be tested - TSO TEST
PMAR_LOAD          1 BIT          Only loadable
PMAR_EXEC          1 BIT          Executable
PMAR_PAGA          1 BIT          Page alignment is required
PMAR_XSSI          1 BIT          SSI information present
PMAR_XAPF          1 BIT          APF information present
PMAR_RMOD          1 BIT          RMODE is ANY.
PMAR_AAMD          2 BITS         Alias entry point addressing mode. If B'00',
                                  AMODE is 24.If B'10', AMODE is 31. Is B'11', AMODE is ANY.
PMAR_MAMD          2 BITS         Main entry point addressing mode. If
                                  B'00', AMODE is 24. If B'10', AMODE is 31. If B'11', AMODE is ANY.
PMAR_AC            BYTE           APF authorization code
PMAR_EPM           WORD           Main entry point offset
PMAR_EPA           WORD           This entry point offset
PMAR_SSI           32 BITS        SSI information
PMARL_PRIM         1 BIT          FETCHOPT PRIME option
PMARL_PACK         1 BIT          FETCHOPT PACK option
```

Figure 99. DESERV UPDATE

If any field other than PMAR_EPA or PMAR_AAMD is updated, this update becomes effective for all entry point names. If PMAR_EPA or PMAR_AAMD are updated, these updates only affect the entry point represented by the input SMDE.

The UPDATE function does not affect connections established by other DE services invocations.

Using the FIND Macro to Position to the Beginning of a Member

To position to the beginning of a specific member, you must issue a FIND macro. The next input or output operation begins processing at the point set by FIND. The FIND macro lets you search a concatenated series of PDSE and PDS directories when you supply a DCB opened for the concatenated data set.

There are two ways you can direct the system to the right member when you use the FIND macro. Specify the address of an area containing the name of the member, or specify the address of the TTR*k* field of the entry in a BLDL list you have created, by using the BLDL macro. *k* is the concatenation number of the data set containing the member. In the first case, the system searches the directory of the data set to connect to the member. In the second case, no search is required, because the relative track address is in the BLDL list entry.

FIND by NAME or TTR establishes a connection to the specific PDSE member. Connections established by *name* remain until positioning is established to another member. Connections established by TTR remain until CLOSE.

If the data set is open for output, close it and reopen it for input or update processing before issuing the FIND macro.

Processing a Partitioned Data Set Extended (PDSE)

If you have insufficient access authority (RACF execute authority), or if the share options are violated, the FIND macro fails. See “Sharing PDSEs” on page 460 for a description of the share options permitted for PDSEs.

Using ISITMGD to Determine Whether the Data Set Is System Managed

You can use the ISITMGD macro to determine if an open data set is system managed and if it is a PDSE. The ISITMGD macro sets some bits in the ISITMGD parameter list that you should test to see what type of data set is being processed. The IGWCISM macro maps the ISITMGD parameter list. In the assembler example in Figure 100, ISITMGD was issued for the second data set in the concatenation (CONCAT=1).

```
         OPEN  (PDSEDCB,(INPUT))          OPEN PDSE
         ...
*************************************************************
* ISSUE ISITMGD FOR SECOND DATA SET IN THE CONCATENATION
*************************************************************
         ISITMGD DCB=PDSEDCB,MF=(E,ISITPARM),CONCAT=1
         USING ISM,1
         LTR   15,15                     Did it complete successfully
         BNZ   ISITERR                   No, branch to error routine
         TM    ISMOFLG1,ISMMGD           Is data set system-managed?
         BZ    NOTMGD                    No, branch to non-SMS routine
         TM    ISMOFLG2,ISMPDSE          Is data set a PDSE
         BO    ANPDSE                    Yes, branch to PDSE routine
OTHER    WTO   'PDS: system-managed data set'
         B     EXIT
*************************************************************
* PROCESS A NON-SMS MANAGED PDS
*************************************************************
NOTMGD   EQU   *
         TM    ISMOFLG2,ISMPDSE          Is data set a PDSE
         BO    ANUMPDSE                  Yes, branch to PDSE routine
         WTO   'PDS: non-system-managed data set'
         B     EXIT
*************************************************************
* PROCESS AN UNMANAGED MANAGED PDSE
*************************************************************
ANUMPDSE EQU   *
         WTO   'PDSE: unmanaged data set'
         B     EXIT
*************************************************************
* PROCESS A MANAGED PDSE
*************************************************************
ANPDSE   EQU   *
         WTO   'PDSE: system-managed data set'
         ...
PDSEDCB  DCB     DSORG=PO,DDNAME=PDSEDDN,MACRF=(R)
ISITPARM ISITMGD MF=L                    Defines space for parameter list
         IGWCISM                         Maps parameter list
         ...
```

Figure 100. ISITMGD Example

If you are testing a single data set, use the CONCAT default, which is 0. The CONCAT parameter is used only for partitioned concatenation, not sequential concatenation. For sequential concatenation, the current data set is tested. The return code in register 15 shows whether the function failed or is not supported on the system.

ISITMGD can also be used to determine the type of library, data or program. Specifying the DATATYPE option on the ISITMGD macro will set the library type in the parameter list. See constants ISMDTREC, ISMDTPGM, and ISMDTUNK in macro IGWCISM for the possible data type settings.

Using the NOTE Macro to Provide Relative Position

You can use the NOTE macro to find the starting address (TTRz) of the most recent record read or written. NOTE returns a TTRz, which can be used by POINT to position to any record in a member. NOTE returns a value of X'7FFF' for the track balance or track capacity. There is no need to calculate the track capacity or balance.

If you issue the NOTE macro while pointing to within the PDSE directory, a TTRz is returned that represents the location of the first directory record.

The TTRz returned from a NOTE for the first directory record is the only valid TTRz that can be used for positioning by POINT while processing within the PDSE directory.

Here are some examples of results when using NOTE with PDSEs. A NOTE:

- immediately following an OPEN returns an invalid address (X'00000000'). Also, if a member is being pointed to using a FIND macro or by the member name in the JCL, but no READ, WRITE, or POINT has been issued, NOTE returns an invalid address of (X'00000000').
- immediately following a STOW ADD or STOW REPLACE returns the TTRz of the logical end-of-file mark for the member stowed. If the member is empty (no writes done), the value returned is the starting TTRz of the member stowed.
- following any READ after an OPEN returns the starting TTRz of the PDSE directory if no member name is in the JCL, or the TTRz of the member if the member name is in the JCL.
- following the first READ after a FIND or POINT (to the first record of a member) returns the TTRz of the member.
- following the first WRITE of a member returns the TTRz of the member.
- following a later READ or WRITE returns the TTRz of the first logical record in the block just read or written.
- issued while positioned in the middle of a spanned record returns the TTRz of the beginning of that record.
- issued immediately following a POINT operation (where the input to the POINT was in the form “TTR1”) will return a note value of “TTR0”.
- issued immediately following a POINT operation (where the input to the POINT was in the form “TTR0”) will return an invalid note value X'00000000').

For information about the NOTE macro, see “Using the NOTE Macro to Return the Relative Address of a Block” on page 481 and *z/OS DFSMS Macro Instructions for Data Sets*.

Using the POINT Macro to Position to a Block

The POINT macro causes the next READ or WRITE operation to position at the beginning of a PDSE member, or anywhere within a PDSE member. The POINT macro uses the track record address (TTRz), which you can obtain from NOTE, BLDL, or a BSAM read of the directory, to position to the correct location. If

positioning to the beginning of a member, the z byte in the TTR must be zero. The POINT macro establishes a connection to the PDSE member (unless the connection already exists).

The POINT macro positions to the first segment of a spanned record even if the NOTE was done on another segment. If the current record spans blocks, setting the z byte of the TTRz field to one lets you access the next record (not the next segment).

You can position from one PDSE member to the first block of another member. Then you can position to any record within that member. Attempting to position from one member into the middle of another member causes the wrong record to be accessed. Either data from the first member will be read, or an I/O error will occur. When the PDSE is open for output, using the POINT macro to position to a member other than the one being written results in a system ABEND.

If you have insufficient access authority (you have only RACF execute authority) or if the share options are violated, the POINT macro fails with an I/O error. See “Sharing PDSEs” on page 460.

For more information about the POINT macro, see *z/OS DFSMS Macro Instructions for Data Sets* and “Using the POINT Macro to Position to a Block” on page 482.

Switching between Members

You can use the NOTE, FIND, and POINT macros to switch between PDSE members (process member 1, then process member 2, then continue processing member 1). Information about how to locate the TTR within a member is also shown in Figure 101 on page 457.

```
         OPEN    (PODCB,(INPUT))          Open the DSORG=PO DCB
         ...
         BLDL    PODCB,BLDLLIST           Construct directory entry list
         ...
         FIND    PODCB,BLDLTTR1,C         Position to the 1st member
         ...
         READ    DECB1,SF,PODCB,BUFFER1   Read records from 1st member
         ...
         CHECK   DECB1                    Check the read
         ...
         NOTE    PODCB             Note a position within the 1st member
         ST      1,INMEM1          Store the TTR for the position in member 1
         ...
         FIND    PODCB,BLDLTTR2,C         Position to the 2nd member
         ...
         READ    DECBA,SF,PODCB,BUFFER1   Read records from 2nd member
         ...
         CHECK   DECBA                    Check the read
         ...
         FIND    PODCB,BLDLTTR1,C         Position back to 1st member
         ...
         POINT   PODCB,INMEM1,TYPE=REL    Position to within 1st member
         ...
         READ    DECB2,SF,PODCB,BUFFER1   Read records from 1st member
         ...
         CHECK   DECB2                    Check the read
         ...
         CLOSE   PODCB                    Close the DCB
         ...
PODCB    DCB     DSORG=PO,MACRF=(R),NCP=1,DDNAME=PDSEDD,---
*
INMEM1   DS      F
*
BLDLLIST DS      0F                BLDL parmlist
BLDLFF   DC      H'2'              Number of entries in BLDL list
BLDLLL   DC      H'12'             Length of storage for directory entry
BLDLN1   DC      CL8'MEMBER1'      Member name
BLDLTTR1 DS      CL3               TTR of 1st member
BLDLK1   DS      CL1               Concatenation # of 1st member
BLDLN2   DC      CL8'MEMBER2'      Member name
BLDLTTR2 DS      CL3               TTR of 2nd member
BLDLK2   DS      CL1               Concatenation # of 2st member
*
BUFFER1  DS      ...
```

Figure 101. Using NOTE and FIND to Switch Between Members of a Concatenated PDSE

This example uses FIND by TTR. Note that when your program resumes reading a member, that member might have been replaced by another program. See “Sharing PDSEs” on page 460.

Using the STOW Macro to Update the Directory

When you add more than one member to a PDSE, you must issue a STOW macro after writing each member so that an entry for each one will be added to the directory. To use the STOW macro, DSORG=PO must be specified in the DCB macro.

You can also use the STOW macro to add, delete, replace, or change a member name in the directory. The add and replace options also store additional information in the directory entry. When you use STOW REPLACE to replace a primary member name, any existing aliases are deleted. When you use STOW DELETE to

delete a primary member name, any existing aliases are deleted. STOW ADD and REPLACE are not permitted against PDSE program libraries.

The STOW INITIALIZE function allows you to clear, or reset to empty, a PDSE directory, as shown in Figure 102:

```
         OPEN    (PDSEDCB,(OUTPUT))       Open the PDSE
         ...
         STOW    PDSEDCB,,I               Initialize (clear) the PDSE directory
         ...
PDSEDCB  DCB     DSORG=PO,MACRF=(W), ...  PDSE DCB
```

Figure 102. STOW INITIALIZE Example

Issuing the STOW macro synchronizes the data to DASD. See “Using the SYNCDEV Macro to Synchronize Data” on page 483 for more information about synchronizing data, and “STOW—Update the Directory” on page 418 for more information about using the STOW macro.

Retrieving a Member of a PDSE

To retrieve a specific member from a PDSE, use either BSAM or QSAM as follows:

1. Code DSORG=PS in the DCB macro.
2. Specify in the DD statement that the data is a member of an existing PDSE by coding DSNAME=name(membername) and DISP=OLD.
3. Process the member with an OPEN macro, a series of GET or READ macros, and then a CLOSE macro.

Figure 103 gives an example of retrieving a member of a PDSE.

```
//PDSEDD DD    ---,DSN=MASTFILE(MEMBERK),DISP=OLD
         ...
         OPEN  (INDCB)            Open for input, automatic FIND
         ...
         GET   INDCB,INAREA       Read member record
         ...
         CLOSE (INDCB)
         ...
INAREA   DS    CL80               Area to read into
INDCB    DCB   ---,DSORG=PS,DDNAME=PDSEDD,MACRF=GM
```

Figure 103. Retrieving One Member of a PDSE

When your program is executed, OPEN searches the directory automatically and positions the DCB to the member.

Processing a Partitioned Data Set Extended (PDSE)

To retrieve several PDSE or PDS members without closing and reopening the data set, use this procedure or the procedure shown in Figure 84 on page 422:

1. Code DSORG=PO in the DCB macro.
2. Specify the name of the PDSE in the DD statement by coding DSNAME=name.
3. Issue the BLDL macro to get the list of member entries you need from the directory.
4. Repeat the following steps for each member to be retrieved.
   a. Use the FIND or POINT macro to prepare for reading the member records. If you use the POINT macro, it will not work in a partitioned concatenation.
   b. The records can be read from the beginning of the member. If you want to read out of sequential order, use the POINT macro to point to records within the member.
   c. Read and check the records until all those required have been processed.
   d. Your end-of-data-set (EODAD) routine receives control at the end of each member. At that time, you can process the next member or close the data set.

To read randomly within a member, use the POINT macro.

Figure 104 on page 460 shows the technique for processing several members without closing and reopening. Figure 84 on page 422 shows a variation of retrieving members. It gives better performance with a PDS or a concatenation of PDSs and PDSEs.

```
//PDSEDD DD    ---,DSN=D42.MASTFILE,DISP=SHR
         ...
         OPEN  (INDCB)               Open for input, no automatic FIND
         ...
         BLDL  INDCB,BLDLLIST        Retrieve the relative disk locations
*                                    of several user-supplied names in
*                                    virtual storage.
         LA    BLDLREG,BLDLLIST+4    Point to the first entry in the list
         ...
Begin a "MEMBER", possibly in another concatenated data set
         MVC   TTRN(4),8(BLDLREG)     Get relative disk address of member
         FIND  INDCB,TTRN,C           Point to the member
         ...
         READ  DECBX,SF,INDCB,INAREA  Read a block of the member
         CHECK DECBX                  Wait for completion of READ

         ...                          READ and CHECK additional blocks
EODRTN   EQU   *                      EOD routine label

         AH    BLDLREG,BLDLLIST+2     Move to next member entry
```

Repeat from label “MEMBER” for each additional member

```
         ...
         CLOSE (INDCB)
         ...

INAREA   DS    CL80
INDCB    DCB   ---,DSORG=PO,DDNAME=PDSEDD,MACRF=R,EODAD=EODRTN
TTRN     DS    F                 TTRN of the start of the member
BLDLREG  EQU   5                 Register to address BLDL list entries
BLDLLIST DS    0F                List of member names for BLDL
         DC    H'10'             Number of entries (10 for example)
         DC    H'14'             Number of bytes per entry
         DC    CL8'MEMBERA'      Name of member, supplied by user
         DS    CL3               TTR of first record (set by BLDL)
*                                The following 3 fields are set by BLDL
         DS    X                 K byte, concatenation number
         DS    X                 Z byte, location code
         DS    X                 C byte, flag and user data length
         ...                     one list entry per member (14 bytes each)
```

Figure 104. Retrieving Several Members of a PDSE or PDS

Sharing PDSEs

PDSE data sets and members can be shared. If allocated with DISP=SHR, the PDSE directory can be shared by multiple writers and readers, and each PDSE member can be shared by a single writer or multiple readers. Any number of systems can have the same PDSE open for input. If one system has a PDSE open for output, that PDSE can be opened on other systems only if the systems are using the PDSE extended sharing protocol. If one system has a PDSE open for update, that PDSE cannot be opened by any other system.

Sharing within a Computer System

Specifying DISP=OLD, NEW, or MOD restricts access to a PDSE by a single job, started task, or TSO/E user. DISP=SHR lets multiple jobs or users access the PDSE at the same time, and permits sharing between DCBs in the same job step and in different jobs.

Member-Level Sharing. As in a PDS, multiple copies (versions) of a member having the same name but different TTRs can exist in a PDSE at the same time. Sharing is done on a “per version” level. The sharing rules depend on whether the DCB is open for input, update, or output.

INPUT—A version of a member can be accessed by any number of DCBs open for input.

UPDATE—You cannot have any DCBs reading a version of a member while another DCB is updating the same version of the member.

OUTPUT—Any number of DCBs open for output can create members at the same time. The members are created in separate areas of the PDSE. If the members being created have the same name (specified in the STOW done after the data is written), the last version stowed is the version that is seen by users, and the storage occupied by the first version is added to the available space for the PDSE. You can have:

- Multiple DCBs reading and creating new versions of the same member at the same time. Readers continue to see the “old” version until they do a new BLDL or FIND by name.
- A single DCB updating a version of a member while multiple DCBs are creating new versions of the member. The user updating the data set continues to access the “old” version until the application does a new BLDL or FIND by name.

Sharing Violations

Violation of the sharing rules, either within a computer system or across several computer systems, can result in OPEN failing with a system ABEND.

Under some conditions, using the FIND or POINT macro might violate sharing rules:

- The share options let only one user update at a time. Suppose you are updating a PDSE and are using the FIND or POINT macros to access a specific member. If someone else is reading that member at the same time, the first WRITE or PUTX issued after the FIND or POINT fails. (The failure does not occur until the WRITE or PUTX because you could be open for update but only reading the member. However, your FIND or POINT would succeed if the other user is reading a different member of the same PDSE at the same time. A POINT error simulates an I/O error.
- If the calling program has insufficient RACF access authority, the FIND or POINT will fail. For example, if the calling program opens a PDSE for input but only has RACF execute authority, the FIND will fail. See *z/OS Security Server RACF Security Administrator’s Guide*.

Multiple System Sharing of PDSEs

Multiple systems in a sysplex can concurrently access PDSE members for input and output, but not for update-in-place.

A shared-access user of a PDSE can read existing members and create new members or new copies of existing members concurrently with other shared-access users on the same system and on other systems. Shared access to a PDSE during an update-in-place of a member is restricted to a single system. Programs on other systems cannot open the data set.

Figure 105 shows the results of OPEN for UPDAT.

System 2 - attempting to:		System 1 - Current PDSE OPEN Status		
		OPEN for UPDAT and positioned to a member	OPEN for UPDAT and NOT positioned to a member	OPEN for INPUT or OUTPUT
OPEN for Input or Output		ABEND 213-70	Successful	Successful
OPEN for UPDAT	BSAM BSAM/QSAM - OPEN to directory, no member name in JCL	ABEND 213-70	Successful	Successful
	BSAM/QSAM - member name in JCL	ABEND 213-70	ABEND 213-74	ABEND 213-74

Note: If no other system is OPEN to the PDSE, then any type of OPEN will be successful.

Figure 105. OPEN Success/Failure

Figure 106 on page 463 show the results of OPEN for UPDAT with positioning in a decision table.

System 2 - OPEN for UPDAT, but NOT YET positioned to a member, and attempting to:	System 1 - Currently OPEN (INPUT, OUTPUT, or UPDAT)
POINT to member READ CHECK	I/O error reported on CHECK: 1. ABEND 001 or 2. entry to SYNAD (If defined in DCB)
FIND to member	RC=4 RSN=8

Note: If no other system is OPEN to the PDSE, then a member can be positioned to without error.

Figure 106. OPEN for UPDAT and Positioning to a Member Decision Table

Buffered Data Invalidation—VARY OFFLINE

When the PDSE sharing protocol is in use, PDSE data is buffered after close of the data set. VARY OFFLINE causes the nonopen PDSE data in system buffers of the system on which the VARY occurs to be invalidated.

Before using program packages which change the VTOC and the data on the volume (for example, DSS full volume, and tracks RESTORE), it is recommended that the volume be VARIED OFFLINE to all other systems. Applications that perform these modifications to data residing on volumes without using the PDSE API should specify in their usage procedure that the volume being modified should be OFFLINE to all other systems, to help ensure there are no active connections to PDSEs residing on the volume while performing the operation.

DFP Share Attributes Callable Service (IGWLSHR)

IGWLSHR can be used by applications to determine the PDSE sharing protocol currently in use. When PDSE EXTENDED sharing protocol is in use, OPEN/CLOSE protocols can be modified to improve performance.

IGWLSHR is invoked by issuing a program CALL accompanied by a parameter (IGWLSHR) identifying the DFP share attributes call service, a list of arguments, and a storage area to return the result. You can use the information obtained by means of IGWLSHR to optimize PDSE access protocols. With concurrent sharing of a PDSE for output between multiple MVS systems, you can open a PDSE for OUTPUT for an extended period without locking out other INPUT or OUTPUT sharers of the PDSE. The exception is opening for update-in-place, which obtains exclusive control of the PDSE for one MVS instance.

See *z/OS DFSMSdfp Advanced Services* for information about the DFP share attributes callable service.

Modifying a Member of a PDSE

The following sections discuss updating, rewriting, and deleting members of a PDSE.

Members of a PDSE program library cannot be rewritten, extended, or updated in place. When updating program objects in a PDSE program library, the AMASPZAP service aid invokes the program management binder, which creates a new version of the program rather than updating the existing version in place.

Updating in Place

A member of a PDSE can be updated in place. Only one user can update at a time. When you update-in-place, you read records, process them, and write them back to their original positions without destroying the remaining records. The following rules apply:

- You must specify the UPDAT option in the OPEN macro to update the data set. To perform the update, you can use only the READ, WRITE, GET, PUTX, CHECK, NOTE, POINT, FIND, BLDL, and STOW macros.
- You cannot update concatenated PDSEs.
- You cannot delete any record or change its length; you cannot add new records.
- You cannot use LBI, large block interface.

With BSAM and BPAM

A record must be retrieved by a READ macro before it can be updated by a WRITE macro. Both macros must be execute forms that refer to the same DECB; the DECB must be provided by a list form. (The execute and list forms of the READ and WRITE macros are described in *z/OS DFSMS Macro Instructions for Data Sets*.)

With Overlapped Operations

See Figure 85 on page 425 for an example of overlap achieved by having a read or write request outstanding while each record is being processed.

With QSAM

You can update a member of a PDSE using the locate mode of QSAM (DCB specifies MACRF=(GL,PL)) and using the GET and PUTX macros. The DD statement must specify the data set and member name in the DSNAME parameter. Using this method, only the member specified in the DD statement can be updated.

Extending a Member

You cannot extend a PDSE member by opening the PDSE for output and positioning to that member. If you used POINT for positioning, the next write would result in an I/O error. If you used FIND for positioning, the FIND will fail with an error return code. To extend the member, rewrite it while open for output and issue a STOW REPLACE.

When you rewrite the member, you must provide two DCBs, one for input and one for output. Both DCB macros can refer to the same data set; that is, only one DD statement is required.

Because space is allocated when the data set is created, you do not need to request additional space. You do not need to compress the PDSE after rewriting a member because the system automatically reuses the member's space whenever a member is replaced or deleted.

Deleting a Member

This section describes the two interfaces used to delete members: STOW and DESERV DELETE. DESERV only supports PDSEs but it does support deleting names longer than 8 bytes.

When the primary name is deleted, the system also deletes all aliases. If an alias is deleted, the system deletes only the alias name and its directory entry.

A PDSE member is not actually deleted while in use. Any program connected to the member when the delete occurs can continue to access the member until the data set is closed. This is called a deferred delete. Any program not connected to the member at the time it is deleted cannot access the member. It appears as though the member does not exist in the PDSE.

Unlike a PDS, after a PDSE member is deleted, it cannot be accessed. (The pointer to the member is removed so that the application can no longer access it. The data can be overwritten by the creation of another member later.)

With DESERV DELETE, it is possible to define a list of PDSE member names (primary and alias) that are to be deleted. The DESL_NAME_PTR fields of the DESL array are used to point to the names to be deleted. The DELETE function requires the caller to pass a DCB open for output or update. The names are processed in DESL sequence. As with any PDSE member deletion, if a primary name is deleted, all the associated aliases are also deleted. Codes in the DESL indicate whether the DELETE was successful for each of the names in the list. The DESL_SMDE_PTR is ignored. The DELETE function terminates processing of entries in the list if it encounters an error where the return code value is greater than RC_WARN. Currently the only error for which processing can continue is when a name is not found, DESRS_NOTFOUND.

Renaming a Member

This section describes the two ways to rename a member: STOW and DESERV RENAME. DESERV RENAME only supports PDSEs but it supports names longer than 8 characters.

With DESERV RENAME, it is possible to define a list of PDSE member names (primary and alias) that are to be renamed. The DESL_OLD_NAME_PTR fields of the DESL array are used to point to the names which are to be renamed. The associated DESL_NAME_PTR fields are used to point to the new names. The RENAME function requires the caller to pass a DCB open for output or update. The renames are processed in DESL sequence. Codes in the DESL indicate whether the rename was successful for each entry in the list.

Reading a PDSE Directory

You can read a PDSE directory sequentially just by opening the data set (without using positioning macros) and reading it. The PDSE directory cannot be updated or written. The following rules and guidelines apply to reading a PDSE directory:

- The DD statement must identify the DSNAME without a member name.
- You can use either BSAM or QSAM with MACRF=R or G.
- Specify BLKSIZE=256 and RECFM=F or RECFM=U.
- If you also want to read the keys (the name of the last member in that block), use BSAM and specify KEYLEN=8.
- After reading the last PDSE directory entry, you read the next directory, or control passes to your EODAD routine. The last directory entry is indicated with a dummy name of eight bytes of X'FF'.
- Alias entries with names longer than eight bytes are omitted. To read them, use DESERV.

You can use the above technique to sequentially read the directories of a concatenation of PDSs and PDSEs. This is considered to be a *like* sequential concatenation. To proceed to each successive data set you can rely on the system's EOV function or you can issue the FEOV macro.

Concatenating PDSEs

Two or more PDSEs can be automatically retrieved by the system and processed successively as a single data set. This technique is known as concatenation. There are two types of concatenation: sequential and partitioned. You can concatenate PDSEs with sequential and PDSs.

Sequential Concatenation

Sequentially concatenated data sets are processed with a DCB that has DSORG=PS, and can include sequential data sets, and members of PDSs and PDSEs. See "Concatenating Data Sets Sequentially" on page 379 for the rules for concatenating *like* and *unlike* data sets.

You can use sequential concatenation (DSORG=PS in DCB) to sequentially read directories of PDSs and PDSEs. See "Reading a PDS Directory Sequentially" on page 427 and "Reading a PDSE Directory".

Partitioned Concatenation

Concatenated PDSs are processed with a DCB that has DSORG=PO, and can include complete PDSs and PDSEs. The system treats the group as a single-partitioned-organization data set. For concatenation, the system regards each PDSE as taking only one extent regardless of the actual number of DASD extents taken. The limit on the number of extents for a concatenation is 120. However, there is also a limit of 114 on the total number of data sets in a partitioned concatenation.

If all the data sets in the concatenation are PDSEs, you can concatenate up to 114 PDSEs. You can concatenate PDSEs and PDSs when the number of DASD extents in the PDSs plus the number of PDSEs does not exceed the concatenation limit of 120. For example, you can concatenate 7 PDSs of 16 extents each with 8 PDSEs (7 x 16 + 8 = 120 extents). See “Concatenating PDSs” on page 426 for information about partitioned concatenation.

Converting PDSs to PDSEs and Back

You can use IEBCOPY or DFSMSdss COPY to convert PDSs to PDSEs. You can convert the entire data set or individual members, and also back up and restore PDSEs. PDSEs can be converted back to PDSs. When copying members from a PDS load module library into a PDSE program library, or vice versa, the system invokes the program management binder.

To copy one or more specific members using IEBCOPY, use the SELECT control statement. In this example, IEBCOPY copies members A, B, and C from USER.PDS.LIBRARY to USER.PDSE.LIBRARY.

```
//INPDS   DD  DSN=USER.PDS.LIBRARY,DISP=SHR
//OUTPDSE DD  DSN=USER.PDSE.LIBRARY,DISP=OLD
//SYSIN   DD  DD *
          COPY OUTDD=OUTPDSE
          INDD=INPDS
          SELECT MEMBER=(A,B,C)
```

This DFSMSdss COPY example converts all PDSs with the high-level qualifier of “MYTEST” on volume SMS001 to PDSEs with the high-level qualifier of “MYTEST2” on volume SMS002. The original PDSs are then deleted. If you use dynamic allocation, specify INDY and OUTDY for the input and output volumes. However, if you define the ddnames for the volumes, use the INDD and OUTDD parameters.

```
COPY DATASET(INCLUDE(MYTEST.**)   -
             BY(DSORG = PDS))     -
             INDY(SMS001)         -
             OUTDY(SMS002)        -
             CONVERT(PDSE(**))    -
             RENAMEU(MYTEST2)     -
             DELETE
```

If you want the PDSEs to retain the original PDS names, use the TSO RENAME command to rename each PDSE individually. (You cannot use pattern-matching characters, such as asterisks, with TSO RENAME.)

```
RENAME (old-data-set-name) (new-data-set-name)
```

If you want to rename all the PDSEs at once, use the access method services ALTER command and run a job:

```
ALTER MYTEST2.* NEWNAME(MYTEST.*)
```

See *z/OS DFSMSdss Storage Administration Reference* for information about using DFSMSdss and *z/OS DFSMSdfp Utilities* for information about using IEBCOPY to convert PDSs to PDSEs.

Rule: Once you have converted to a PDSE, you cannot access the data from a system at an earlier level than MVS/DFP™ 3.2. That might affect disaster recovery plans.

PDSE to PDS Conversion

Situations in which you might want to convert a PDSE to a PDS follow:

- You are shipping the PDSE to a system that does not support PDSEs.
- An application does not run against a PDSE.
- A system is sharing the PDSE (using shared DASD) with a system that does not support PDSE access.
- You are performing disaster recovery at a site running earlier versions of MVS/DFP on its systems.

To convert a PDSE to a PDS, specify a DSNTYPE of PDS in the JCL or data class definition.

Restrictions on Converting PDSEs

If you attempt to copy members of PDSs containing user TTRs or note lists to a PDSE, you get an error message and the copy fails.

If the SYNCDEV macro is coded in an application you want to convert, the application can handle the return and reason codes for PDSs correctly. The correct return code is 4, which means “SYNCDEV does not support PDSs”.

When copying members from a PDSE program library into a PDS, certain restrictions must be considered. Program objects which exceed the limitations of load modules, such as total module size or number of external names, cannot be correctly converted to load module format.

Improving Performance

After many adds and deletes, the PDSE members might become fragmented. This can affect performance. To reorganize the PDSE, use IEBCOPY or DFSMSdss COPY to back up all the members. You can either delete all members and then restore them, or delete and reallocate the PDSE. It is preferable to delete and reallocate the PDSE because it usually uses less processor time and does less I/O than deleting every member.

Recovering Space in Fragmented PDSEs

PDSEs can become fragmented depending on the access pattern. This does not normally occur when the adding and deleting of members is balanced, but might occur when members are deleted and new members are not added to reclaim the space. To reclaim the space and reorganize the PDSE, copy it to a new PDSE using IEBCOPY or DFSMSdss COPY.

Chapter 29. Processing Generation Data Groups

This chapter covers the following topics.

Topic	Page
Absolute Generation and Version Numbers	470
Relative Generation Number	471
Programming Considerations for Multiple Step Job	471
Naming Generation Data Groups for ISO/ANSI Version 3 or Version 4 Labels	472
Creating a New Generation	473
Retrieving a Generation Data Set	477
Building a Generation Data Group Index	478

You can catalog successive updates or generations of related data. They are called generation data groups. Each data set within a generation data group is called a generation data set or generation. Within a generation data group, the generations can have like or unlike DCB attributes and data set organizations. If the attributes and organizations of all generations in a group are identical, the generations can be retrieved together as a single data set.

Generation data sets can be sequential, direct, or indexed sequential. They cannot be partitioned, HFS, or VSAM.

There are advantages to grouping related data sets. For example, the catalog management routines can refer to the information in a special index called a generation index in the catalog. Thus:

- All of the data sets in the group can be referred to by a common name.
- The operating system is able to keep the generations in chronological order.
- Outdated or obsolete generations can be automatically deleted by the operating system.

Generation data sets have sequentially ordered absolute and relative names that represent their age. The catalog management routines use the absolute generation name. Older data sets have smaller absolute numbers. The relative name is a signed integer used to refer to the latest (0), the next to the latest (−1), and so forth, generation. For example, a data set name LAB.PAYROLL(0) refers to the most recent data set of the group; LAB.PAYROLL(−1) refers to the second most recent data set; and so forth. The relative number can also be used to catalog a new generation (+1).

A generation data group base is allocated in a catalog before the generation data sets are cataloged. Each generation data group is represented by a generation data group base entry. Use the access method services DEFINE command to allocate the generation data group base.

Note: For new non-system-managed data sets, if you do not specify a volume and the data set is not opened, the system does not catalog the data set. New system-managed data sets are always cataloged when allocated, with the volume assigned from a storage group.

© Copyright IBM Corp. 1987, 2002

Processing Generation Data Groups

See *z/OS DFSMS Access Method Services* for information about defining and cataloging generation data sets in a catalog.

Notes:

1. A generation data group base that is to be system managed must be created in a catalog. Generation data sets that are to be system managed must also be cataloged in a catalog.
2. Both system-managed and non-system-managed generation data sets can be contained in the same generation data group. However, if the catalog of a generation data group is on a volume that is system managed, the model DSCB cannot be defined.
3. You can add new non-system-managed generation data sets to the generation data group by using cataloged data sets as models without needing a model DSCB on the catalog volume.

Absolute Generation and Version Numbers

An absolute generation and version number is used to identify a specific generation of a generation data group. The generation and version numbers are in the form `GxxxxVyy`, where `xxxx` is an unsigned 4-digit decimal generation number (0001 through 9999) and `yy` is an unsigned 2-digit decimal version number (00 through 99). For example:

- A.B.C.G0001V00 is generation data set 1, version 0, in generation data group A.B.C.
- A.B.C.G0009V01 is generation data set 9, version 1, in generation data group A.B.C.

The number of generations and versions is limited by the number of digits in the absolute generation name; that is, there can be 9,999 generations. Each generation can have 100 versions.

The system automatically maintains the generation number. The number of generations kept depends on the size of the generation index. For example, if the size of the generation index permits ten entries, the ten latest generations can be maintained in the generation data group.

The version number lets you perform normal data set operations without disrupting the management of the generation data group. For example, if you want to update the second generation in a 3-generation group, replace generation 2, version 0, with generation 2, version 1. Only one version is kept for each generation.

You can catalog a generation using either absolute or relative numbers. When a generation is cataloged, a generation and version number is placed as a low-level entry in the generation data group. To catalog a version number other than V00, you must use an absolute generation and version number.

You can catalog a new version of a specific generation automatically by specifying the old generation number along with a new version number. For example, if generation A.B.C.G0005V00 is cataloged and you now create and catalog A.B.C.G0005V01, the new entry is cataloged in the location previously occupied by A.B.C.G0005V00. The old entry is removed from the catalog, to make room for the newer version, and may or may not be scratched depending on what limit processing options are specified for the generation data group (GDG) base. For system-managed data sets, if scratch was specified, the older version will be scratched from the volume. If noscratch was specified, or if the attempt to scratch

the DSCB failed, the older version will not be scratched and the general data sets (GDS) will be recataloged as a non-VSAM data set with the GnnnnVnn name not associated with the GDG base. For non-system-managed data sets, the older version is likewise governed by the GDG base limit processing options. If noscratch was specified for the base, the older GDS version is not scratched. To scratch the old version and make its space available for reallocation, include a DD statement, describing the data set to be deleted, with DISP=(OLD,DELETE) when the data set is to be replaced by the new version.

Relative Generation Number

As an alternative to using absolute generation and version numbers when cataloging or referring to a generation, you can use a relative generation number. To specify a relative number, use the generation data group name followed by a negative integer, a positive integer, or a 0, enclosed in parentheses. For example, A.B.C(−1). A.B.C(+1), or A.B.C(0).

The value of the specified integer tells the operating system what generation number to assign to a new generation, or it tells the system the location of an entry representing a previously cataloged generation.

When you use a relative generation number to catalog a generation, the operating system assigns an absolute generation number and a version number of V00 to represent that generation. The absolute generation number assigned depends on the number last assigned and the value of the relative generation number that you are now specifying. For example if, in a previous job generation, A.B.C.G0005V00 was the last generation cataloged, and you specify A.B.C(+1), the generation now cataloged is assigned the number G0006V00.

Though any positive relative generation number can be used, a number greater than 1 can cause absolute generation numbers to be skipped. For example, if you have a single step job, and the generation being cataloged is a +2, one generation number is skipped. However, in a multiple step job, one step might have a +1 and a second step a +2, in which case no numbers are skipped.

Programming Considerations for Multiple Step Job

One reason for using generation data groups is to allow the system to maintain a given number of related cataloged data sets. If you attempt to delete or uncatalog any but the oldest of the data sets of a generation data group in a multiple step job, catalog management can lose orientation within the data group. This can cause the wrong data set to be deleted, uncataloged, or retrieved when referring to a specified generation. The rule is, if you delete a generation data set in a multiple step job, do not refer to any older generation in subsequent job steps.

Also, in a multiple step job, you should catalog or uncatalog data sets using JCL rather than IEHPROGM or a user program. Because data set allocation and unallocation monitors data sets during job execution and is not aware of the functions performed by IEHPROGM or user programs, data set orientation might be lost or conflicting functions might be performed in subsequent job steps.

When you use a relative generation number to refer to a generation that was previously cataloged, the relative number has the following meaning:
- A.B.C(0) refers to the latest existing cataloged entry.
- A.B.C(−1) refers to the next-to-the-latest entry, and so forth.

Processing Generation Data Groups

When cataloging is requested using JCL, all actual cataloging occurs at step termination, but the relative generation number remains the same throughout the job. Because this is so:

- A relative number used in the JCL refers to the same generation throughout a job.
- A job step that ends abnormally can be deferred for a later step restart. If the step cataloged a generation data set successfully in its generation data group using JCL, you must change all relative generation numbers in the next steps using JCL before resubmitting the job.

For example, if the next steps contained the following relative generation numbers:
- A.B.C(+1) refers to the entry cataloged in the terminated job step, or
- A.B.C(0) refers to the next to the latest entry, or
- A.B.C(−1) refers to the latest entry, before A.B.C(0).

You must change A.B.C(+1) to A.B.C(0), A.B.C(0) to A.B.C(−1), and A.B.C(−1) to A.B.C(−2) before restarting the step.

Naming Generation Data Groups for ISO/ANSI Version 3 or Version 4 Labels

In a Version 3 or Version 4 ISO/ANSI label (LABEL=(,AL)), the generation number and version number are maintained separately from the file identifier. Label processing removes the generation number and version number from the generation data set name. The generation number is placed in the generation number field (file label 1 positions 36 through 39), and the version number is placed in its position on the same label (position 40 and 41). The file identifier portion of a Version 3 or Version 4 ISO/ANSI label contains the generation data set name without the generation number and version number.

For Version 3 or Version 4 labels, you must observe the following specifications created by the generation data group naming convention.

- Data set names whose last 9 characters are of the form `.GnnnnVnn` (n is 0 through 9) can only be used to specify GDG data sets. When a name ending in `.GnnnnVnn` is found, it is automatically processed as a GDG. The generation number `Gnnnn` and the version number `Vnn` are separated from the rest of the data set name and placed in the generation number and version number fields.
- Tape data set names for GDG files are expanded from a maximum of 8 user-specified characters to 17 user-specified characters. (The tape label file identifier field has space for 9 additional user-specified characters because the generation number and version number are no longer contained in this field.)
- A generation number of all zeros is not valid, and will be treated as an error during label validation. The error appears as a “RANG” error in message IEC512I (IECIEUNK) during the label validation installation exit.
- In an MVS system-created GDG name, the version number will always be 0. (MVS will not increase the version number by 1 for subsequent versions.) To obtain a version number other than 0, you must explicitly specify the version number (for example, A.B.C.G0004V03) when the data set is allocated. You must also explicitly specify the version number to retrieve a GDG with a version number other than 0.
- Because the generation number and version number are not contained on the identifier of HDR1, generations of the same GDG will have the same name. Therefore, an attempt to place more than one generation of a GDG on the same

volume will result in an ISO/ANSI standards violation in a system supporting Version 3 and MVS will enter the validation installation exit.

Creating a New Generation

To allocate a new generation data set, you must first allocate space for the generation, then catalog the generation. This section also discusses passing a generation, rolling in a generation, and allocating an indexed sequential data set as part of a generation data group.

Allocating a Generation Data Set

The allocation can be patterned after a previously allocated generation in the same group, by specifying DCB attributes for the new generation, described as follows.

If you are using absolute generation and version numbers, DCB attributes for a generation can be supplied directly in the DD statement defining the generation to be created and cataloged.

If you are using relative generation numbers to catalog generations, DCB attributes can be supplied:

1. By referring to a cataloged data set for the use of its attributes.
2. By creating a model DSCB on the volume on which the index resides (the volume containing the catalog). Attributes can be supplied before you catalog a generation, when you catalog it, or at both times.

   Restriction: You cannot use a model DSCB for system-managed generation data sets.
3. By using the DATACLAS and LIKE keywords in the DD statement for both system-managed and non-system-managed generation data sets. The generation data sets can be on either tape or DASD.
4. Through the assignment of a data class to the generation data set by the data class ACS routine.

Referring to a Cataloged Data Set

You do not need to create a model DSCB if you can refer to a cataloged data set whose attributes are identical to those you desire. You can refer to the cataloged data set's DCB attributes by referring to its DCB or to the DD statement that allocated it.

To refer to a cataloged data set for the use of its attributes, you can specify one of the following on the DD statement that creates and catalogs your generation:

- DCB=(dsname), where dsname is the name of the cataloged data set
- LIKE=dsname, where dsname is the name of the cataloged data set
- REFDD=ddname, where ddname is the name of a DD statement that allocated the cataloged data set.

Examples: An example of allocating a generation data set by supplying its DCB attributes using DATACLAS is:

```
//DD1 DD DSN=GDG(+1),DISP=(NEW,CATLG),DATACLAS=ALLOCL01
```

The DCB attributes allocated to the new data set depend on the attributes defined in data class ALLOCL01. Your storage administrator can provide information on the attributes specified by the data classes available to your installation.

An example of referring to a cataloged data set by referring to its DD statement is:

```
//DD2 DD DSN=GDG(+1),DISP=(NEW,CATLG),REFDD=DD1
```

The new generation data set will have the same attributes as the data set defined in the first example.

You can also refer to an existing model DSCB for which you can supply overriding attributes.

Restriction: You cannot use a model DSCB with system-managed generation data sets.

To refer to an existing model, specify DCB=(*modeldscbname, your attributes*) on the DD statement that creates and catalogs your generation. Assume that you have a generation data group base name ICFUCAT8.GDGBASE and its model DSCB name is ICFUCAT8.GDGBASE.

You can specify:

```
//DD1  DD DSN=ICFUCAT8.GDGBASE(+1),DISP=(NEW,CATLG),
//        UNIT=3380,SPACE=(TRK,(5)),VOL=SER=338001
```

Creating a Model DSCB

You can create a model DSCB on the volume on which your index resides.

Restriction: You cannot use a model DSCB for system-managed generation data sets.

You can provide initial DCB attributes when you create your model; however, you need not provide any attributes now. Because only the attributes in the data set label are used, allocate the model data set with SPACE=(TRK,0) to conserve direct access space. You can supply initial or overriding attributes creating and cataloging a generation. To create a model DSCB, include the following DD statement in the job step that builds the index or in any other job step that precedes the step in which you create and catalog your generation:

```
//name DD DSNAME=datagrpname,DISP=(,KEEP),SPACE=(TRK,(0)),
//        UNIT=yyyy,VOLUME=SER=xxxxxx,
//        DCB=(applicable subparameters)
```

Recommendation: Only one model DSCB is necessary for any number of generations. If you plan to use only one model, do not supply DCB attributes when you create the model. When you subsequently create and catalog a generation, include necessary DCB attributes in the DD statement referring to the generation. In this manner, any number of generation data groups can refer to the same model. The catalog and model data set label are always located on a direct access volume, even for a magnetic tape generation data group.

In the preceding example, *datagrpname* is the common name that identifies each generation, and *xxxxxx* is the serial number of the volume that contains the catalog. If you do not want any DCB subparameters initially, you need not code the DCB parameter.

The model DSCB must reside on the catalog volume. If you move a catalog to a new volume, you also will need to move or create a new model DSCB on this new volume. If you split or merge a catalog and the catalog remains on the same volume as the existing model DSCB, you will not have to move or create a new model DSCB.

Using DATACLAS and LIKE Keywords

You can use the DATACLAS and LIKE keywords in the DD statement for both system-managed and non-system-managed generation data sets. For non-system-managed generation data sets, DATACLAS and LIKE can be used in place of a model DSCB. The data sets can be on either tape or DASD. See *z/OS DFSMS: Using Magnetic Tapes* about using data class with tape data sets.

The LIKE keyword specifies the allocation attributes of a new data set by copying the attributes of a cataloged model data set. The cataloged data set referred to in LIKE=*dsname* must be on DASD.

Recommendation: You can still use model DSCBs if they are present on the volume, even if LIKE and DATACLAS are also used for a non-system-managed generation data set. If you use model DSCBs, you do not need to change the JCL (to scratch the model DSCB) when migrating the data to system-managed storage or migrating from system-managed storage. If you do not specify DATACLAS and LIKE in the JCL for a non-system-managed generation data set, and there is no model DSCB, the allocation fails.

An example of allocating a non-system-managed generation data set by supplying its DCB attributes using DATACLAS and LIKE follows. This example would also work for system-managed generation data sets.

```
//DDNAME DSN=HLQ.----.LLQ(+1),DISP=(NEW,CATLG),DATACLAS=dc_name

//DDNAME DSN=HLQ.----.LLQ(+1),DISP=(NEW,CATLG),LIKE=dsn
```

For more information on the JCL keywords used to allocate a generation data set, see *z/OS MVS JCL Reference*.

The new generation data set is cataloged at allocation time, and rolled into the generation data group at the end-of-job step. If your job ends after allocation but before the end-of-job step, the generation data set is cataloged in a deferred roll-in state. You can resubmit your job to roll the new generation data set into the generation data group. For more information about rolling in generation data sets see “Rolling In a Generation Data Set” on page 476.

Passing a Generation Data Set

A new generation can be passed when created. That generation can then be cataloged in a succeeding job step, or deleted at the end of the job as in normal disposition processing when DISP=(,PASS) is specified on the DD statement.

However, after a generation has been created with DISP=(NEW,PASS) specified on the DD statement, another new generation for that data group must not be cataloged until the passed version has been deleted or cataloged. To catalog another generation would cause the wrong generation to be used when referencing the passed generation data set. If that data set was later cataloged, a bad generation would be cataloged and a good one lost.

For example, if A.B.C(+1) was created with DISP=(NEW,PASS) specified on the DD statement, then A.B.C.(+2) must not be created with DISP=(NEW,CATLG) until A.B.C(+1) has been cataloged or deleted.

By using the proper JCL, the advantages to this support are:

- JCL will not have to be changed to rerun the job.
- The lowest generation version will not be deleted from the index until a valid version is cataloged.

Rolling In a Generation Data Set

For system-managed generation data groups, a new generation is cataloged in a deferred roll-in state at allocation time when the generation data set is allocated as DISP=(NEW,CATLG). At the end-of-job step, the deferred generation data set will be rolled into the generation data group. It becomes an active generation data set. Generation data sets can be in a deferred roll-in state if the job never reached the end-of-job step or if they were allocated as DISP=(NEW,KEEP) and the data set is not system managed. Generation data sets in a deferred roll-in state can be referred to by their absolute generation numbers. You can use the access method services command ALTER ROLLIN to roll in these generation data sets.

The attributes specified for the generation data group determines what happens to the older generations when a new generation is rolled. The access method services command DEFINE GENERATIONDATAGROUP creates a generation data group. It also specifies the limit (the maximum number of active generation data sets) for a generation data group, and specifies whether all or only the oldest generation data sets should be rolled off when the limit is reached.

When a generation data group contains its maximum number of active generation data sets, and a new generation data set is rolled in at the end-of-job step, the oldest generation data set is rolled off and is no longer active. If a generation data group is defined using DEFINE GENERATIONDATAGROUP EMPTY, and is at its limit, then, when a new generation data set is rolled in, all the currently active generation data sets are rolled off.

The parameters you specify on the DEFINE GENERATIONDATAGROUP command determines what happens to rolled off generation data sets. For example, if you specify the SCRATCH parameter, the generation data set is scratched when it is rolled off. If you specify the NOSCRATCH parameter, the rolled off generation data set is recataloged as rolled off and is disassociated with its generation data group.

The access method services command ALTER LIMIT can increase or reduce the limit for an existing generation data group. If a limit is reduced, the oldest active generation data sets are automatically rolled off as needed to meet the decreased limit. If a change in the limit causes generations to be rolled off, then the rolled off data sets are listed with their disposition (uncataloged, recataloged, or deleted). If a limit is increased, and there are generation data sets in a deferred roll-in state, these generation data sets are not rolled into the generation data group. The access method services command ALTER ROLLIN can be used to roll the generation data sets into the generation data group in active status.

For more information about using the access method services commands DEFINE GENERATIONDATAGROUP and ALTER see *z/OS DFSMS Access Method Services*.

Controlling Expiration of a Rolled-Off Generation Data Set

Three variables control the expiration of a rolled-off generation data set, in the following order:

1. Expiration date coded
2. Base SCRATCH or NOSCRATCH
3. Management class EXPIRE/MIGRATE.

Creating an ISAM Data Set as Part of a Generation Data Group

To allocate an indexed sequential data set as part of a generation data group, you must: (1) allocate the indexed sequential data set separately from the generation group and (2) use IEHPROGM to put the indexed sequential data set into the generation group. An ISAM generation data set cannot be system managed.

In catalogs, use access method services commands to catalog the data set. For example, if MASTER is the name of the generation data group, and `GggggVvv` is the absolute generation name, you would code the following:

```
RENAME DSNAME=ISAM,VOL=3380=SCRTH1,NEWNAME=MASTER.GggggVvv
CATLG DSNAME=MASTER.GggggVvv,VOL=3380=SCRTH1
```

Retrieving a Generation Data Set

You can retrieve a generation using job control language procedures. Any operation that can be applied to a nongeneration data set can be applied to a generation data set. For example, a generation data set can be updated and reentered in the catalog, or it can be copied, printed, punched, or used in the creation of new generation or nongeneration data sets.

You can retrieve a generation data set by using either relative generation numbers or absolute generation and version numbers.

Processing Generation Data Groups

Refer to generation data sets that are in a deferred roll-in state by their relative number, such as (+1), within the job that allocates it. Refer to generation data sets that are in a deferred roll-in state by their absolute generation number (G*xxxx*V*yy*) in subsequent jobs.

Because two or more jobs can compete for the same resource, be careful when updating generation data groups, as follows:

- No two jobs running concurrently should refer to the same generation data group. As a partial safeguard against this situation, use absolute generation and version numbers when cataloging or retrieving a generation in a multiprogramming environment.
- Even when using absolute generation and version numbers, a job running concurrently might catalog a new version of a generation, or perhaps delete the generation you wanted to retrieve. So, some degree of control should be maintained over the execution of job steps referring to generation data groups.

Building a Generation Data Group Index

A generation data group contained in a catalog is managed through access method services. The access method services DEFINE command can be used to allocate a generation data group and to specify how to handle older and obsolete generations.

A generation data group is managed using the information found in a generation index.

Rule: An alias name cannot be assigned to the highest level of a generation index.

The BLDG function of IEHPROGM builds the index. The BLDG function also indicates how older or obsolete generations are to be handled when the index is full. For example, when the index is full, you might want to empty it, scratch existing generations, and begin cataloging a new series of generations. After the index is built, a generation can be cataloged by its generation data group name, and by either an absolute generation and version number or a relative generation number.

Examples of how to build a generation data group index are found in *z/OS DFSMS Access Method Services* and in *z/OS DFSMSdfp Utilities*.

Chapter 30. Using I/O Device Control Macros

This chapter explains how to use the z/OS operating system's macros for controlling I/O devices. *z/OS DFSMS Macro Instructions for Data Sets* explains how to invoke each of these time-saving features. To varying degrees, each macro is device dependent, so you must exercise care if you want to achieve device independence. You can adapt your application to various device types by issuing the DEVTYPE macro. See *z/OS DFSMSdfp Advanced Services*.

This chapter covers the following topics.

Topic	Page
Using the CNTRL Macro to Control an I/O Device	479
Using the PRTOV Macro to Test for Printer Overflow	480
Using the SETPRT Macro to Set Up the Printer	480
Using the BSP Macro to Backspace a Magnetic Tape or Direct Access Volume	481
Using the NOTE Macro to Return the Relative Address of a Block	481
Using the POINT Macro to Position to a Block	482
Using the SYNCDEV Macro to Synchronize Data	483

When you use the queued access method, only unit record equipment can be controlled directly. When using the basic access method, limited device independence can be achieved between magnetic tape and direct access storage devices. With BSAM you must check all read or write operations before issuing a device control macro.

Using the CNTRL Macro to Control an I/O Device

The CNTRL macro performs these device-dependent control functions:

- Card reader stacker selection (SS)
- Printer line spacing (SP)
- Printer carriage control (SK)
- Magnetic tape backspace (BSR) over a specified number of blocks
- Magnetic tape backspace (BSM) past a tape mark and forward space over the tape mark
- Magnetic tape forward space (FSR) over a specified number of blocks
- Magnetic tape forward space (FSM) past a tape mark and a backspace over the tape mark

Backspacing moves the tape toward the load point; forward spacing moves the tape away from the load point.

Restriction: The CNTRL macro cannot be used with an input data set containing variable-length records on the card reader.

If you specify OPTCD=H in the DCB parameter field of the DD statement, you can use the CNTRL macro to position VSE tapes even if they contain embedded checkpoint records. The CNTRL macro cannot be used to backspace VSE 7-track tapes that are written in data convert mode and contain embedded checkpoint records.

© Copyright IBM Corp. 1987, 2002

Using the PRTOV Macro to Test for Printer Overflow

The PRTOV macro tests for channel 9 or 12 of the printer carriage control tape or the forms control buffer (FCB). An overflow condition causes either an automatic skip to channel 1 or, if specified, transfer of control to your routine for overflow processing. If you specify an overflow exit routine, set DCBIFLGS to X'00' before issuing another PRTOV.

If the device specified on the DD statement is not for a directly allocated printer, no action is taken.

Using the SETPRT Macro to Set Up the Printer

The SETPRT macro controls how information is printed. It is used with the IBM 3800 Printing Subsystem, IBM 3900 Printing Subsystem, with various other universal character set (UCS) printers, and SYSOUT data sets.

For printers that are allocated to your program, the SETPRT macro is used to initially set or dynamically change the printer control information. For more information about using the SETPRT macro, see *z/OS DFSMS Macro Instructions for Data Sets*.

For printers that have a universal character set (UCS) buffer and optionally, a forms control buffer (FCB), the SETPRT macro is used to specify the UCS or FCB images to be used. Note that universal character sets for the various printers are not compatible. The three formats of FCB images (the FCB image for the 3800 Printing Subsystem, the 4248 format FCB, and the 3211 format FCB) are incompatible. The 3211 format FCB is used by the 3203, 3211, 4248, 3262 Model 5, and 4245 printers.

IBM-supplied UCS images, UCS image tables, FCB images, and character arrangement table modules are included in the SYS1.IMAGELIB at system initialization time. For 1403, 3203, 3211, 3262 Model 5, 4245, and 4248 printers, user-defined character sets can be added to SYS1.IMAGELIB.

Related reading:

- For a description of how images are added to SYS1.IMAGELIB and how band names/aliases are added to image tables see *z/OS DFSMSdfp Advanced Services*.
- For the 3800 and 3900, user-defined character arrangement table modules, FCB modules, graphic character modification modules, copy modification modules, and library character sets can be added to SYS1.IMAGELIB as described for IEBIMAGE in *z/OS DFSMSdfp Utilities*.
- For information on building a 4248 format FCB (which can also be used for the IBM 3262 Model 5 printer), see *z/OS DFSMSdfp Utilities*.

The FCB contents can be selected from the system library (or an alternate library if you are using a 3800 or 3900), or defined in your program through the exit list of the DCB macro. For information about the DCB exit list see “DCB Exit List” on page 501.

For a non-3800 or non-3900 printer, the specified UCS or FCB image can be found in one of the following:
- SYS1.IMAGELIB
- Image table (UCS image only)
- DCB exit list (FCB image only)

If the image is not found, the operator is asked to specify an alternate image name or cancel the request.

For a printer that has no carriage control tape, you can use the SETPRT macro to select the FCB, to request operator verification of the contents of the buffer, or to allow the operator to align the paper in the printer.

For a SYSOUT data set, the specified images must be available at the destination of the data set, which can be JES2, JES3, VM, or other type of system.

Using the BSP Macro to Backspace a Magnetic Tape or Direct Access Volume

The BSP macro backspaces one block on the magnetic tape or direct access volume being processed. The block can then be reread or rewritten. An attempt to rewrite the block destroys the contents of the remainder of the tape or track. See “Using the BSP Macro to Backspace a Physical Record” on page 446 for information on using the BSP macro to process PDSEs.

The direction of movement is toward the load point or the beginning of the extent. You can not use the BSP macro if the track overflow option was specified or if the CNTRL, NOTE, or POINT macro is used. The BSP macro should be used only when other device control macros could not be used for backspacing.

Any attempt to backspace across the beginning of the data set on the current volume results in return code X'04' in register 15, and your tape or direct access volume is positioned before the first block. You cannot issue a successful backspace command after your EODAD routine is entered unless you first reposition the tape or direct access volume into your data set. CLOSE TYPE=T can position you at the end of your data set.

You can use the BSP macro to backspace VSE tapes containing embedded checkpoint records. If you use this means of backspacing, you must test for and bypass the embedded checkpoint records. You cannot use the BSP macro for VSE 7-track tapes written in translate mode.

Using the NOTE Macro to Return the Relative Address of a Block

The NOTE macro requests the relative address of the first logical record of the block just read or written. In a multivolume non-extended-format data set, the address is relative to the beginning of the data set on the volume currently being processed. In a striped data set, the address is always relative to the beginning of the data set. Your program later uses the address in positioning operations.

If a NOTE macro is issued after an automatic volume switch occurs, and before a READ or WRITE macro is issued to the next volume, NOTE returns a relative block address of zero except for extended format data sets.

For magnetic tape, the address is in the form of a 4-byte relative block address. If TYPE=REL is specified or defaults, the address provided by the operating system is returned in register 1. If TYPE=ABS is specified, the physical block identifier of a data block on tape is returned in register 0. Later you can use the relative block address or the block identifier as a search argument for the POINT macro.

For non-extended-format data sets on direct access storage devices, the address is in the form of a 4-byte relative track record address. For extended format data sets, the address is in the form of a block locator token (BLT). The BLT is essentially the relative block number (RBN) within the current logical volume of the data set where the first block has an RBN of 1. The user sees a multistriped data set as a single logical volume; therefore, for a multistriped data set, the RBN is relative to the beginning of the data set and incorporates all stripes. For PDSEs, the address is in the form of a record locator token. The address provided by the operating system is returned in register 1. For non-extended-format data sets and partitioned data sets, NOTE returns the track balance in register 0 if the last I/O operation was a WRITE, or returns the track capacity if the NOTE follows a READ or POINT. For PDSEs, extended format data sets and HFS files, NOTE returns X'7FFF' in register 0.

See “Using the NOTE Macro to Provide Relative Position” on page 455 for information about using the NOTE macro to process PDSEs.

Using the POINT Macro to Position to a Block

The POINT macro repositions a magnetic tape or direct access volume to a specified block. The next read or write operation begins at this block. See “Using the POINT Macro to Position to a Block” on page 455 for information on using the POINT macro to process PDSEs.

In a multivolume sequential data set you must ensure that the volume referred to is the volume currently being processed. The user sees a multistriped extended-format data set as a single logical volume; therefore, no special positioning is needed. However, a single-striped multivolume extended-format data set does require you to be positioned at the correct volume.

For disk, if a write operation follows the POINT macro, all of the track following the write operation is erased, unless the data set is opened for UPDAT. Closing the data set after such a write logically truncate ends the data set. POINT is not meant to be used before a WRITE macro when a data set is opened for UPDAT.

If you specify OPTCD=H in the DCB parameter field of the DD statement, you can use the POINT macro to position VSE tapes even if they contain embedded checkpoint records. The POINT macro cannot be used to backspace VSE 7-track tapes that are written in data convert mode and that contain embedded checkpoint records.

If you specify TYPE=ABS, you can use the physical block identifier as a search argument to locate a data block on tape. The identifier can be provided from the output of a prior execution of the NOTE macro.

When using the POINT macro for a direct access storage device that is opened for OUTPUT, OUTIN, OUTINX, or INOUT, and the record format is not fixed standard, the number of blocks per track might vary slightly.

Using the SYNCDEV Macro to Synchronize Data

Data still in the buffer might not yet reside on the final recording medium. This is called data that is not synchronized. Data synchronization is the process by which the system ensures that data previously given to the system via WRITE, PUT, and PUTX macros is written to the storage medium.

The SYNCDEV macro performs data synchronization for the following:
- Magnetic tape cartridge devices supporting buffered write mode
- PDSEs to DASD
- Compressed format data sets to DASD.

You can do the following for a magnetic tape cartridge device:
- Request information regarding synchronization, or
- Demand that synchronization occur based on a specified number of data blocks that are allowed to be buffered. If zero is specified, synchronization will always occur.

When SYNCDEV completes successfully (return code 0), a value is returned that shows the number of data blocks remaining in the control unit buffer. For PDSEs and compressed format data sets, the value returned is always zero. For PDSEs and compressed format data sets, requests for synchronization information or for partial synchronization cause complete synchronization. Specify Guaranteed Synchronous Write through storage class to ensure that data is synchronized to DASD at the completion of each CHECK macro. However, this degrades performance. This produces the same result as issuing the SYNCDEV macro after each CHECK macro. See *z/OS DFSMSdfp Storage Administration Reference* for information about how the storage administrator specifies guaranteed synchronous write.

Chapter 31. Identifying Non-VSAM User-Written Exit Routines

This chapter covers the following topics.

Topic	Page
General Guidance	485
EODAD End-of-Data-Set Exit Routine	493
SYNAD Synchronous Error Routine Exit	494
DCB Exit List	501
Allocation Retrieval List	504
DCB ABEND Exit	505
DCB OPEN Exit	509
Defer Nonstandard Input Trailer Label Exit List Entry	510
Block Count Unequal Exit	510
EOV Exit for Sequential Data Sets	511
FCB Image Exit	511
JFCB Exit	513
JFCBE Exit	513
Open/Close/EOV Standard User Label Exit	514
Open/EOV Nonspecific Tape Volume Mount Exit	518
Open/EOV Volume Security and Verification Exit	521
QSAM Parallel Input Exit	523
User Totaling for BSAM and QSAM	523

General Guidance

You can identify user-written exit routines for use with non-VSAM access methods. These user-written exit routines can perform a variety of functions for non-VSAM data sets, including error analysis, requesting user totaling, performing I/O operations for data sets, and creating your own data set labels. These functions are not for use with VSAM data sets. Similar VSAM functions are described in “Chapter 17. Coding VSAM User-Written Exit Routines” on page 241.

The DCB and DCBE macros can be used to identify the locations of exit routines:

- The routine that performs end-of-data procedures (the EODAD parameter of DCB or DCBE)
- The routine that supplements the operating system’s error recovery routine (the SYNAD parameter of DCB or DCBE)
- The list that contains addresses of special exit routines (the EXLST parameter of DCB).

The exit addresses can be specified in the DCB or DCBE macro, or you can complete the DCB or DCBE fields before they are needed. Table 32 on page 486 summarizes the exits that you can specify either explicitly in the DCB or DCBE, or implicitly by specifying the address of an exit list in the DCB.

© Copyright IBM Corp. 1987, 2002

Identifying Non-VSAM User-Written Exit Routines

Table 32. DCB Exit Routines

Exit Routine	When Available	Page
End-of-data-set	When no more sequential records or blocks are available	493
Error analysis	After an uncorrectable input/output error	494
Allocation retrieval list	When issuing an RDJFCB macro instruction	501
Block count	After unequal block count comparison by end-of-volume routine	501
DCB abend	When an abend condition occurs in OPEN, CLOSE, or end-of-volume routine	501
DCB open	When opening a data set	501
End-of-volume	When changing volumes	501
FCB image	When opening a data set or issuing a SETPRT macro	501
JFCB	When opening a data set with TYPE=J and reading the JFCB	501
Standard user label (physical sequential or direct organization)	When opening, closing, or reaching the end of a data set, and when changing volumes	501
JFCB extension (JFCBE)	When opening a data set for the IBM 3800	501
Open/EOV nonspecific tape volume mount	When a scratch tape is requested during OPEN or EOV routines	501
Open/EOV volume security/verification	When a scratch tape is requested during OPEN or EOV routines	501
QSAM parallel processing	Opening a data set	501
User totaling (for BSAM and QSAM)	When creating or processing a data set with user labels	501

Programming Considerations

Most exit routines described in this chapter must return to their caller. The only two exceptions are the end-of-data and error analysis routines.

Status Information Following an Input/Output Operation

Following an I/O operation with a DCB, the control program makes certain status information available to the application program. This status information is a 2 byte exception code, or a 16 byte field of standard status indicators, or both.

Exception codes are provided in the data control block (QISAM), or in the data event control block (BISAM and BDAM). The data event control block is described below, and the exception code lies within the block as shown in Table 33 on page 487. If a DCBD macro instruction is coded, the exception code in a data control block can be addressed as two 1-byte fields, DCBEXCD1 and DCBEXCD2. QISAM exception codes are described in Table 38 on page 495. The other exception codes are described in Table 34 on page 488, Table 36 on page 491, and Table 38 on page 495.

Status indicators are available only to the error analysis routine designated by the SYNAD entry in the data control block or the data control block extension. Or, they are available after I/O completion from BSAM or BPAM until the next WAIT or CHECK for the DCB. A pointer to the status indicators is provided either in the data event control block (BSAM, BPAM, and BDAM), or in register 0 (QISAM and QSAM). The contents of registers on entry to the SYNAD exit routine are shown in Table 39 on page 497, Table 40 on page 498, and Table 41 on page 498. The status indicators for BSAM, BPAM, BDAM, and QSAM are shown in Figure 107 on page 492.

Data Event Control Block

A data event control block is constructed as part of the expansion of READ and WRITE macro instructions and is used to pass parameters to the control program, help control the read or write operation, and receive indications of the success or failure of the operation. The data event control block is named by the READ or WRITE macro instruction, begins on a fullword boundary, and contains the information shown in Table 33.

Table 33. Data Event Control Block

Offset from DECB Address (Bytes)	Field Contents BSAM and BPAM	BISAM	BDAM
0	ECB	ECB	ECB[1]
+4	Type	Type	Type
+6	Length	Length	Length
+8	DCB address	DCB address	DCB address
+12	Area address	Area address	Area address
+16	Address of status indicators[3]	Logical record address	Address of status indicators[3]
+20		Key address	Key address
+24		Exception code (2 bytes)[2]	Block address
+28			Next address

Notes:
1. The control program returns exception codes in bytes +1 and +2 of the ECB.
2. See Table 34 on page 488.
3. See Figure 107 on page 492.

For BISAM, exception codes are returned by the control program after the corresponding WAIT or CHECK macro instruction is issued, as indicated in Table 34 on page 488.

Table 34. Exception Code Bits—BISAM

Exception Code Bit in DECB	READ	WRITE	Condition If On
0	X	Type K	Record not found
1	X	X	Record Length Check
2		Type KN	Space not found
3	X	Type K	Invalid request
4	X	X	Uncorrectable I/O error
5	X	X	Unreachable block
6	X		Overflow record
7		Type KN	Duplicate record
8-15			Reserved for control program use

Descriptions of the conditions in Table 34 follow:

- **Record Not Found:** The logical record with the specified key is not found in the data set if the specified key is higher than the highest key in the highest-level index or if the record is not in either the prime area or the overflow area of the data set.
- **Record Length Check:** For READ and update WRITE macro instructions, an overriding length is specified and (1) the record format is blocked, (2) the record format is unblocked but the overriding length is greater than the length known to the control program, or (3) the record is fixed length and the overriding length does not agree with the length known to the control program. This condition is reported for the add WRITE macro instruction if an overriding length is specified.

  When blocked records are being updated, the control program must find the high key in the block to write the block. (The high key is not necessarily the same as the key supplied by the application program.) The high key is needed for writing because the control unit for direct access devices permits writing only if a search on equal is satisfied; this search can be satisfied only with the high key in the block. If the user were permitted to specify an overriding length shorter than the block length, the high key might not be read; then, a subsequent write request could not be satisfied. In addition, failure to write a high key during update would make a subsequent update impossible.
- **Space Not Found:** No room exists for adding a new record to the data set in either the appropriate cylinder overflow area or the independent overflow area when a new record. The data set is not changed in any way in this situation.
- **Invalid Request:** This condition occurs for either of two reasons:
  - Because the application program altered the contents of byte 25 of the data event control block, byte 25 could indicate that this request is an update WRITE macro instruction corresponding to a READ (for update) macro instruction, but the I/O block (IOB) for the READ instruction is not in the update queue.
  - A READ or WRITE macro instruction specifies dynamic buffering (that is, ‘S’ in the area address operand), but the DCBMACRF field of the data control block does not specify dynamic buffering.
- **Uncorrectable Input/Output Error:** The control program’s error recovery procedures encounter an uncorrectable error in transferring data.

- **Unreachable Block:** An uncorrectable I/O error occurs during a search of the indexes or following an overflow chain. This condition is also posted if the data field of an index record contains an improper address (that is, points to the wrong cylinder or track or is not valid).
- **Overflow Record:** The record just read is an overflow record. The SYNAD exit routine is entered only if the CHECK macro is issued after the READ macro, and bit 0, 4, 5, or 7 is also on. (See the section on direct retrieval and update in "Appendix D. Processing Indexed Sequential Data Sets" on page 545 for considerations during BISAM updating.)
- **Duplicate Record Presented for Inclusion in the Data Set:** The new record to be added has the same key as a record in the data set. However, if the delete option was specified and the record in the data set is marked for deletion, this condition is not reported. Instead, the new record replaces the existing record.

  If the record format is blocked and the relative key position is zero, the new record cannot replace an existing record that is of equal key and is marked for deletion.

Event Control Block

The ECB is located in the first word of the DECB. An event control block is the subject of WAIT and POST macro instructions. See Table 35.

Table 35. Event Control Block

Offset	Bytes	Bit Value	Hex Code	Description
00	1	10xx xxxx		W—Waiting for completion of an event.
		01xx xxxx		C—The event has completed.
				One of the following completion codes will appear at the completion of the operation that was initiated by the READ or WRITE macro:
				Access Methods other than BDAM
		0111 1111	7F	Channel program has terminated without error. (The status indicators in Figure 107 on page 492 are valid.)
		0100 0001	41	Channel program has terminated with permanent error. (The status indicators in Figure 107 on page 492 are valid.)
		0100 0010	42	Channel program has terminated because a direct access extent address has been violated. (The status indicators in Figure 107 on page 492 are not valid.)
		0100 0011	43	Abend condition occurred in the error recovery routine. (The status indicators in Figure 107 on page 492 are not valid.)
		0100 0100	44	Channel program has been intercepted because of permanent error associated with device end for previous request. You may reissue the intercepted request. (The status indicators in Figure 107 on page 492 are not valid.)

Table 35. Event Control Block (continued)

Offset	Bytes	Bit Value	Hex Code	Description
		0100 1000	48	The channel program was purged. (The status indicators in Figure 107 on page 492 are not valid.)
		0100 1011	4B	One of the following errors occurred during tape error recovery processing: • The CSW command address was zeros. • An unexpected load point was encountered. (The status indicators in Figure 107 on page 492 are not valid.)
		0100 1111	4F	Error recovery routines have been entered because of direct access error but are unable to read home addresses or record 0. (The status indicators in Figure 107 on page 492 are not valid.)
		0101 0000	50	Channel program terminated with error. Input block was a VSE-embedded checkpoint record. (The status indicators in Figure 107 on page 492 are not valid.)
1	3			Contains the address of the RB issuing the WAIT macro if the ECB has the WAIT bit on. Once the event has completed and ECB is posted the C bit is set with other bits in byte 0 and these 3 bytes (1-3) are zero, for all access methods except BDAM. For BDAM, only the C bit is set in byte zero and the exception codes are returned in bytes 1 and 2 or the ECB for BDAM.

Table 36 on page 491 shows the exception bit codes for BDAM.

Table 36. Exception Code Bits—BDAM

Exception Code Bit	READ	WRITE	Condition If On
0	X	X	Record not found. (This Record Not Found condition is reported if the search argument is not found in the data set.)
1	X	X	Record length check. (This Record Length Check condition occurs for READ and WRITE (update) and WRITE (add). For WRITE (update) variable-length records only, the length in the BDW does not match the length of the record to be updated. For all remaining READ and WRITE (update) conditions, the BLKSIZE, when **S** is specified in the READ or WRITE macro, or the length given with these macros does not agree with the actual length of the record. For WRITE (add), fixed-length records, the BLKSIZE, when **S** is specified in the WRITE macro, or the length given with this macro does not agree with the actual length of the record. For WRITE (add), all other conditions, no error can occur.)
2		X	Space not found. (This Space Not Found for Adding a Record condition occurs if either there is no dummy record when adding an F-format record, or there is no space available when adding a V- or U-format record.)
3	X	X	Invalid request—see bits 9-15
4	X	X	Uncorrectable I/O error. (This Uncorrectable Input/Output Error condition is reported if the control program's error recovery procedures encounter an uncorrectable error in transferring data between real and secondary storage.)
5	X	X	End of data. (This End of Data only occurs as a result of a READ (type DI, DIF, or DIX) when the record requested is an end-of-data record.)
6	X	X	Uncorrectable error. (Same conditions as for bit 4.)
7		X	Not read with exclusive control. (A WRITE, type DIX or DKX, has occurred for which there is no previous corresponding READ with exclusive control.)
8			Not used
9		X	A WRITE was attempted for an input data set.
10	X	X	An extended search was requested, but LIMCT was zero.
11	X	X	The relative block or relative track requested was not in the data set.
12		X	Writing a capacity record (R0) was attempted.
13	X	X	A READ or WRITE with key was attempted, but either KEYLEN equaled zero or the key address was not supplied.
14	X	X	The READ or WRITE macro request options conflict with the OPTCD or MACRF parameters.
15		X	A WRITE (add) with fixed length was attempted with the key beginning with X'FF'.

Figure 107 lists status indicators for BDAM, BPAM, BSAM, and QSAM.

```
Offset in
status indicator
area
Byte   Bit   Meaning                            Name

-12    -     Word containing length that was read  Valid only when reading with LBI
+2     0     Command reject                     Sense byte 0
       1     Intervention required
       2     Bus-out check
       3     Equipment check
       4     Data check
       5     Overrun
       6,7   Device-dependent information;
             see the appropriate device
             manual
+3     0-7   Device-dependent information;      Sense byte 1
             see the appropriate device
             manual

The following bytes make up the low-order seven bytes of
a simulated channel status word (CSW):
+9     —    Command address pointing after last executed CCW

The ending CCW may have the indirect data addressing bit and/or data chaining bit on
+12    0     Attention                          Status byte 0
       1     Status modifier                    (Unit)
       2     Control unit end
       3     Busy
       4     Channel end
       5     Device end
       6     Unit check—must be on for
             sense bytes to be significant
       7     Unit exception

+13    0     Program-controlled interrupt       Status byte 1
       1     Incorrect length                   (Channel)
       2     Program check
       3     Protection check
       4     Channel data check
       5     Channel control check
       6     Interface control check
       7     Chaining check

+14    —    Count field (2 bytes)      Not valid with LBI
```

Figure 107. Status Indicators—BDAM, BPAM, BSAM, and QSAM

If the sense bytes are X'10FE', the control program has set them to this invalid combination because sense bytes could not be obtained from the device because of recurrence of unit checks.

EODAD End-of-Data-Set Exit Routine

The EODAD parameter of the DCB or DCBE macro specifies the address of your end-of-data-set routine, which can perform any final processing on an input data set. The EODAD routine generally is not regarded as being a subroutine. This routine is entered when your program does any of the following:

- Issues a CHECK macro (for a READ macro) or a GET macro and there are no more records or blocks to be retrieved.
- Issues an FEOV macro while reading on the last or only volume.

With sequential concatenation these events cause entry to your EODAD routine only if you are reading the end of the last data set. For a BSAM data set that is opened for UPDAT, this routine is entered at the end of each volume. This lets you issue WRITE macros before an FEOV macro is issued.

Register Contents

Table 37 shows the contents of the registers when control is passed to the EODAD routine.

Table 37. Contents of Registers at Entry to EODAD Exit Routine

Register	Contents
0-1	Reserved
2-13	Contents before execution of GET, CHECK, FEOV or EOV (EXCP)
14	Contains the address after a GET or CHECK as these macros generate a branch and link to the access method routines. FEOV is an SVC. Register 14 will contain what is contained at the time the FEOV was issued.
15	Reserved

Programming Considerations

You can treat your EODAD routine as a subroutine (and end by branching on register 14) or as a continuation of the routine that issued the CHECK, GET or FEOV macro.

The EODAD routine generally is not regarded as being a subroutine. After control passes to your EODAD routine, you can continue normal processing, such as repositioning and resuming processing of the data set, closing the data set, or processing another data set.

For BSAM, you must first reposition the data set that reached end-of-data if you want to issue a BSP, READ, or WRITE macro. You can reposition your data set by issuing a CLOSE TYPE=T macro instruction. If a READ macro is issued before the data set is repositioned, unpredictable results occur.

For BPAM, you may reposition the data set by issuing a FIND or POINT macro. (CLOSE TYPE=T with BPAM results in no operation performed.)

Identifying Non-VSAM User-Written Exit Routines

For QISAM, you can continue processing the input data set that reached end-of-data by first issuing an ESETL macro to end the sequential retrieval, then issuing a SETL macro to set the lower limit of sequential retrieval.You can then issue GET macros to the data set.

Your task will abnormally end under either of the following conditions:

- No exit routine is provided.
- A GET macro is issued in the EODAD routine to the DCB that caused this routine to be entered (unless the access method is QISAM).

For BSAM, BPAM, and QSAM your EODAD routine is entered with the addressability (24- or 31-bit) of when you issued the macro that caused entry to EODAD. This typically is a CHECK, GET, or FEOV macro. DCB EODAD identifies a routine that resides below the line (RMODE is 24). DCBE EODAD identifies a routine that may reside above the line. If it resides above the line, then all macros that might detect an end-of-data must be issued in 31-bit mode. If both the DCB and DCBE specify EODAD, the DCBE routine is used.

SYNAD Synchronous Error Routine Exit

The SYNAD parameter of the DCB or DCBE macro specifies the address of an error routine that is to be given control when an input/output error occurs. You can use this routine to analyze exceptional conditions or uncorrectable errors. I/O errors usually occur asynchronously to your program, but the access method calls your SYNAD routine synchronously to macros that your program issues.

If an I/O error occurs during data transmission, standard error recovery procedures that are provided by the operating system try to correct the error before returning control to your program. These error recovery procedures generally are asynchronous to your program. An uncorrectable error usually causes an abnormal termination of the task. However, if you specify the address of an error analysis routine (called a SYNAD routine) in the DCB or DCBE macro, that routine can try to correct or ignore the error and prevent an abnormal termination. The routine is given control when the application program issues the access method macro that requires the buffer that received the uncorrectable error. For the queued access methods this generally means after enough PUT or GET macros to fill BUFNO-1 buffers past the failing block. For the basic access methods this means when your program issues a CHECK macro for the failing DECB.

For BDAM, BSAM, BPAM, and QSAM, the control program provides a pointer to the status indicators shown in Figure 107 on page 492. The block being read or written can be accepted or skipped, or processing can be terminated.

Table 38 on page 495 shows the exception code bits for QISAM.

Table 38. Exception Code Bits—QISAM

Exception Field	Code Bit	CLOSE	Code GET	Set PUT	by PUTX	SETL	Condition If On
DCBEXCD1	0					Type K	Record Not Found
	1					Type I	Invalid actual address for lower limit
	2	X					Space not found for adding a record
	3					X	Invalid request
	4		X				Uncorrectable input error
	5	X		X	X		Uncorrectable output error
	6		X			X	Block could not be reached (input)
	7	X	X				Block could not be reached (update)
DCBEXCD2	0			X			Sequence check
	1			X			Duplicate record
	2	X					Data control block closed when error routine entered
	3		X				Overflow record
	4			X			Incorrect record length
	5-7						Reserved for future use

Descriptions of the conditions in Table 38 follow:

- **Record Not Found:** The logical record with the specified key is not found in the data set. This happens if the specified key is higher than the highest key in the highest-level index or if the record is not in either the primary area or the overflow area of the data set.
- Invalid Actual Address for Lower Limit condition is reported if the specified lower limit address is outside the space allocated to the data set.
- **Space Not Found for Adding a Record:** The space allocated to the data set is already filled. In locate mode, a buffer segment address is not provided. In move mode, data is not moved.
- **Invalid Request:** (1) The data set is already being referred to sequentially by the application program, (2) the buffer cannot contain the key and the data, or (3) the specified type is not also specified in the DCBMACRF field of the data control block.
- **Uncorrectable Input Error:** The control program's error recovery procedures encounter an uncorrectable error when transferring a block from secondary storage to an input buffer. The buffer address is placed in register 1, and the SYNAD exit routine is given control when a GET macro instruction is issued for the first logical record.
- **Uncorrectable Output Error:** The control program's error recovery procedures encounter an uncorrectable error when transferring a block from an output buffer to secondary storage. If the error is encountered during closing of the data control block, bit 2 of DCBEXCD2 is set to *1* and the SYNAD exit routine is given control immediately. Otherwise, control program action depends on whether load mode or scan mode is being used.

If a data set is being created (load mode), the SYNAD exit routine is given control when the next PUT or CLOSE macro instruction is issued. If a failure to write a data block occurs, register 1 contains the address of the output buffer, and register 0 contains the address of a work area containing the first 16 bytes of the IOB; for other errors, the contents of register 1 are meaningless. After appropriate analysis, the SYNAD exit routine should close the data set or end the job step. If records are to be subsequently added to the data set using the queued indexed sequential access method (QISAM), the job step should be terminated by issuing an abend macro instruction. (Abend closes all open data sets. However, an ISAM data set is only partially closed, and it can be reopened in a later job to add additional records by using QISAM.) Subsequent execution of a PUT macro instruction would cause reentry to the SYNAD exit routine, because an attempt to continue loading the data set would produce unpredictable results.

If a data set is being processed (scan mode), the address of the output buffer in error is placed in register 1, the address of a work area containing the first 16 bytes of the IOB is placed in register 0, and the SYNAD exit routine is given control when the next GET macro instruction is issued. Buffer scheduling is suspended until the next GET macro instruction is reissued.

- Block Could Not Be Reached (Input) condition is reported if the control program's error recovery procedures encounter an uncorrectable error in searching an index or overflow chain. The SYNAD exit routine is given control when a GET macro instruction is issued for the first logical record of the unreachable block.
- **Block Could Not Be Reached (Update):** The control program's error recovery procedures encounter an uncorrectable error in searching an index or overflow chain.

  If the error is encountered during closing of the data control block, bit 2 of DCBEXCD2 is set to 1 and the SYNAD exit routine is given control immediately. Otherwise, the SYNAD exit routine is given control when the next GET macro instruction is issued.
- **Sequence Check:** A PUT macro instruction refers to a record whose key has a smaller numeric value than the key of the record previously referred to by a PUT macro instruction. The SYNAD exit routine is given control immediately; the record is not transferred to secondary storage.
- **Duplicate Record:** A PUT macro instruction refers to a record whose key duplicates the record previously referred to by a PUT macro instruction. The SYNAD exit routine is given control immediately; the record is not transferred to secondary storage.
- **Data Control Block Closed When Error Routine Entered:** The control program's error recovery procedures encounter an uncorrectable output error during closing of the data control block. Bit 5 or 7 of DCBEXCD1 is set to 1, and the SYNAD exit routine is immediately given control. After appropriate analysis, the SYNAD routine must branch to the address in return register 14 so that the control program can finish closing the data control block.
- **Overflow Record:** The input record is an overflow record. The SYNAD exit routine is entered only if bit 4, 5, 6, or 7 of DCBEXCD1 is also on.
- **Incorrect Record Length:** The length of the record as specified in the record-descriptor word (RDW) is larger than the value in the DCBLRECL field of the data control block.

Register Contents

Table 39 shows the register contents on entry to the SYNAD routine for BDAM, BPAM, BSAM, and QSAM.

Table 39. Register Contents on Entry to SYNAD Routine—BDAM, BPAM, BSAM, and QSAM

Register	Bits	Meaning
0	0-7	Value to be added to the status indicator's address to provide the address of the first CCW (QSAM only). Value may be zero, meaning unavailable, if LBI is used.
	8-31	Address of the associated data event control block for BDAM, BPAM, and BSAM unless bit 2 of register 1 is on; address of the status indicators shown in Figure 107 on page 492 for QSAM. If bit 2 of register 1 is on, the failure occurred in CNTRL, POINT, or BSP and this field contains the address on an internal BSAM ECB.
1	0	Bit is on for error caused by input operation.
	1	Bit is on for error caused by output operation.
	2	Bit is on for error caused by BSP, CNTRL, or POINT macro instruction (BPAM AND BSAM only).
	3	Bit is on if error occurred during update of existing record or if error did not prevent reading of the record. Bit is off if error occurred during creation of a new record or if error prevented reading of the record.
	4	Bit is on if the request was invalid. The status indicators pointed to in the data event control block are not present (BDAM, BPAM, and BSAM only).
	5	Bit is on if an invalid character was found in paper tape conversion (BSAM and QSAM only).
	6	Bit is on for a hardware error (BDAM only).
	7	Bit is on if no space was found for the record (BDAM only).
	8-31	Address of the associated data control block.
2-13	0-31	Contents that existed before the macro instruction was issued.
14	0-7	Reserved.
	8-31	Return address.
15	0-31	Address of the error analysis routine.

Table 40 on page 498 shows the register contents on entry to the SYNAD routine for BISAM.

Table 40. Register Contents on Entry to SYNAD Routine—BISAM

Register	Bits	Meaning
0	0-7	Reserved.
	8-31	Address of the first of two sense bytes. (Sense information is valid only when associated with a unit check condition.)
1	0-7	Reserved.
	8-31	Address of the DECB. See Table 33 on page 487.
2-13	0-31	Contents that existed before the macro instruction was issued.
14	0-7	Reserved.
	8-31	Return address.
15	0-7	Reserved.
	8-31	Address of the SYNAD exit routine.

Table 41 shows the register contents on entry to the SYNAD routine for QISAM.

Table 41. Register Contents on Entry to SYNAD Routine—QISAM

Register	Bits	Meaning
0	0	Bit 0=1 indicates that bits 8-31 hold the address of the key in error (only set for a sequence error). If bit 0=1—address of key that is out of sequence. If bit 0=0—address of a work area.
	1-7	Reserved.
	8-31	Address of a work area containing the first 16 bytes of the IOB (after an uncorrectable I/O error caused by a GET, PUT, or PUTX macro instruction; original contents destroyed in other cases). If the error condition was detected before I/O was started, register 0 contains all zeros.
1	0-7	Reserved.
	8-31	Address of the buffer containing the error record (after an uncorrectable I/O error caused by a GET, PUT, or PUTX macro instruction while attempting to read or write a data record; in other cases, this register contains 0).
2-13	0-31	Contents that existed before the macro instruction was issued.
14	0-7	Reserved.
	8-31	Return address. This address is either an address in the control program's CLOSE routine (bit 2 of DCBEXCD2 is on), or the address of the instruction following the expansion of the macro instruction that caused the SYNAD exit routine to be given control (bit 2 of DCBEXCD2 is off).
15	0-7	Reserved.
	8-31	Address of the SYNAD exit routine.

Programming Considerations

For BSAM, BPAM, and QSAM your SYNAD routine is entered with the addressability (24- or 31-bit) of when you issued the macro that caused entry to SYNAD. This typically is a CHECK, GET, or PUT macro. DCB SYNAD identifies a routine that resides below the line (RMODE is 24). DCBE SYNAD identifies a routine that may reside above the line. If it resides above the line, then all macros that might detect an I/O error must be issued in 31-bit mode. If both the DCB and DCBE specify SYNAD, the DCBE routine will be used.

You can write a SYNAD routine to determine the cause and type of error that occurred by examining:

- The contents of the general registers
- The data event control block (see “Status Information Following an Input/Output Operation” on page 486)
- The exceptional condition code
- The standard status and sense indicators

You can use the SYNADAF macro to perform this analysis automatically. This macro produces an error message. Your program can use a PUT, WRITE, or WTO macro to print the message.

Your SYNAD routine can act as an exit routine and return to its caller, or the SYNAD routine can continue in your main program with restrictions on the DCB. In other words, branch elsewhere in your program and, after the analysis is complete, you can return control to the operating system or close the data set. If you close the data set, you cannot use the temporary close (CLOSE TYPE=T) option in the SYNAD routine. To continue processing the same data set, you must first return control to the control program by a RETURN macro. The control program then transfers control to your processing program, subject to the conditions described below. Never attempt to reread or rewrite the record, because the system has already attempted to recover from the error.

You should not use the FEOV macro against the data set for which the SYNAD routine was entered, within the SYNAD routine.

Queued Access Methods

When you are using GET and PUT to process a sequential data set, the operating system provides three automatic error options (EROPT) to be used if there is no SYNAD routine or if you want to return control to your program from the SYNAD routine:

ACC—accept the erroneous block
SKP—skip the erroneous block
ABE—abnormally terminate the task

These options are applicable only to data errors, because control errors result in abnormal termination of the task. Data errors affect only the validity of a block of data. Control errors affect information or operations necessary for continued processing of the data set. These options are not applicable to a spooled data set, a subsystem data set, or output errors, except output errors on a real printer. If the EROPT and SYNAD fields are not complete, ABE is assumed.

Because EROPT applies to a physical block of data, and not to a logical record, use of SKP or ACC may result in incorrect assembly of spanned records.

Basic Access Methods

When you use READ and WRITE macros, errors are detected when you issue a CHECK macro. If you are processing a direct data set, sequential data set, or PDS and you return to the control program from your SYNAD routine, the operating system assumes that you have accepted the bad block. If you are creating a direct data set and you return to the control program from your SYNAD routine, your task ends abnormally. If you are processing a direct data set, make the return to the control program through register 14 to make an internal system control block available for reuse in a READ or WRITE macro.

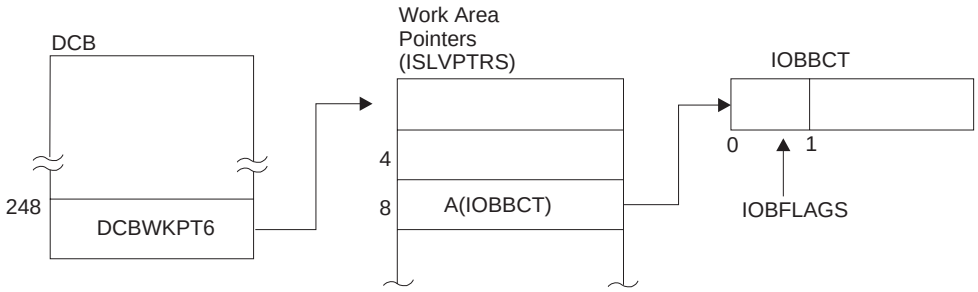
Returning from the SYNAD routine

Your SYNAD routine can end by branching to another routine in your program, such as a routine that closes the data set. It can also end by returning control to the control program. The control program then returns control to the next sequential instruction (after the macro) in your program, after a basic access method or if the queued access method is honoring EROPT=AC or EROPT=SKP. If your routine returns control, the conventions for saving and restoring the contents of registers are as follows:

- The SYNAD routine must preserve the contents of registers 13 and 14. The routine must also preserve the contents of registers 2 through 12 if the logic of your program requires their preservation. On return to your program, the contents of registers 2 through 12 will be the same as on return to the control program from the SYNAD routine.
- Register 13 contains the address of a save area that the control program uses. The SYNAD routine must not use this save area. If the routine saves and restores register contents, it must provide its own save area.
- If the SYNAD routine calls another routine or issues supervisor or data management macros, it must provide its own save area or issue a SYNADAF macro. The SYNADAF macro provides a save area for its own use and then makes this area available to the SYNAD routine. A SYNADRLS macro must remove such a save area from the save area chain before control is returned to the control program.

ISAM

If the error analysis routine receives control from the CLOSE routine when indexed sequential data sets are being created (the DCB is opened for QISAM load mode), bit 3 of the IOBFLAGS field in the load mode buffer control table (IOBBCT) is set to 1. The DCBWKPT6 field in the DCB contains an address of a list of work area pointers (ISLVPTRS). The pointer to the IOBBCT is at offset 8 in this list as shown in the following diagram:



If the error analysis routine receives control from the CLOSE routine when indexed sequential data sets are being processed using QISAM scan mode, bit 2 of the DCB field DCBEXCD2 is set to 1.

For information about QISAM error conditions and the meanings they have when the ISAM interface to VSAM is being used, see “Appendix E. Using ISAM Programs with VSAM Data Sets” on page 579.

DCB Exit List

The EXLST parameter of the DCB macro specifies the address of a DCB exit list. The DCB exit list may contain the addresses of special processing routines, a forms control buffer (FCB) image, a user totaling area, an area for a copy of the JFCB, and an allocation retrieval list. A DCB exit list must be created if user label, data control block, end-of-volume, block count, JFCBE, or DCB abend exits are used, or if a PDAB macro or FCB image is defined in the processing program.

The DCB exit list must begin on a fullword boundary and each entry in the list requires one fullword. Each exit list entry is identified by a code in the high-order byte, and the address of the routine, image, or area is specified in the 3 low-order bytes. Codes and addresses (including the information location) for the exit list entries are shown in Table 42.

Identifying Non-VSAM User-Written Exit Routines

Table 42. DCB Exit List Format and Contents

Entry Type	Hex Code	3 Byte Address—Purpose	Page
Inactive entry	00	Ignore the entry; it is not active.	
Input header label exit	01	Process a user input header label.	514
Output header label exit	02	Create a user output header label.	514
Input trailer label exit	03	Process a user input trailer label.	514
Output trailer label exit	04	Create a user output trailer label.	514
Data control block OPEN exit	05	Take an exit during OPEN processing.	509
End-of-volume exit	06	Take an end-of-volume exit.	511
JFCB exit	07	JFCB address for RDJFCB and OPEN TYPE=J macros.	513
	08	Reserved.	
	09	Reserved.	
User totaling area	0A	Address of beginning of user's totaling area.	523
Block count unequal exit	0B	Process tape block count discrepancy.	510
Defer input trailer label	0C	Defer processing of a user input trailer label from end-of-data until closing.	514
Defer nonstandard input trailer label	0D	Defer processing of a nonstandard input trailer label on magnetic tape unit from end-of-data until closing (no exit routine address).	510
	0E-0F	Reserved.	
FCB image	10	Define an FCB image.	511
DCB abend exit	11	Examine the abend condition and select one of several options.	505
QSAM parallel input	12	Address of the PDAB for which this DCB is a member.	523
Allocation retrieval list	13	Retrieve allocation information for one or more data sets with the RDJFCB macro.	504
	14	Reserved.	
JFCBE exit	15	Take an exit during OPEN to let a user examine JCL-specified setup requirements for a 3800 printer.	513
	16	Reserved.	
OPEN/EOV nonspecific tape volume mount	17	Option to specify a tape volume serial number.	518
OPEN/EOV volume security/verification	18	Verify a tape volume and some security checks.	521
	1A-7F	Reserved.	
Last entry	80	Treat this entry as the last entry in the list. This code can be specified with any of the above but must always be specified with the last entry.	

You can activate or deactivate any entry in the list by placing the required code in the high-order byte. Care must be taken, however, not to destroy the last entry indication. The operating system routines scan the list from top to bottom, and the first active entry found with the proper code is selected.

You can shorten the list during execution by setting the high-order bit to 1, and extend it by setting the high-order bit to 0.

Exit routines identified in a DCB exit list are entered in 24-bit mode even if the rest of your program is executing in 31-bit mode. *z/OS DFSMS Macro Instructions for Data Sets* has an example showing how to build a 24-bit routine in an area below the 16 MB line that acts as a glue routine and branches to your 31-bit routine above the line.

Register Contents for Exits from EXLST

When control is passed to an exit routine, the registers contain the following information:

Register	Contents
0	Variable; see exit routine description.
1	The 3 low-order bytes contain the address of the DCB currently being processed, except when the user-label exits (X'01' - X'04' and X'0C'), user totaling exit (X'0A'), DCB abend exit (X'11'), nonspecific tape volume mount exit (X'17'), or the tape volume security/verification exit (X'18') is taken, when register 1 contains the address of a parameter list. The contents of the parameter list are described in the explanation of each exit routine.
2-13	Contents before execution of the macro.
14	Return address (must not be altered by the exit routine).
15	Address of exit routine entry point.

The conventions for saving and restoring register contents are as follows:

- The exit routine must preserve the contents of register 14. It need not preserve the contents of other registers. The control program restores the contents of registers 2 to 13 before returning control to your program.
- The exit routine must not use the save area whose address is in register 13, because this area is used by the control program. If the exit routine calls another routine or issues supervisor or data management macros, it must provide the address of a new save area in register 13.
- The exit routine must not issue an access method macro that refers to the DCB for which the exit routine was called, unless otherwise specified in the individual exit routine descriptions that follow.

Serialization

During any of the exit routines described in this section, the system might hold an enqueue on the SYSZTIOT resource. The resource represents the TIOT and DSAB chain and holding it or being open to the DD are the only ways to ensure that dynamic unallocation in another task does not eliminate those control blocks while they are being examined. If the system holds the SYSZTIOT resource, your exit routine cannot use certain system functions that might need the resource. Those functions include LOCATE, OBTAIN, SCRATCH, CATALOG, OPEN, CLOSE, FEOV, and dynamic allocation. Whether or not the system holds that resource is part of system logic and IBM might change it in a future release. IBM recommends that your exit routine not depend on the system holding or not holding SYSZTIOT. One

example of your exit routine depending on the system holding SYSZTIOT is your routine testing control blocks for DDs outside the concatenation.

Allocation Retrieval List

The RDJFCB macro uses the DCB exit list entry with code X'13' to retrieve allocation information (JFCBs and volume serial numbers). When you issue RDJFCB, the JFCBs for the specified data sets, including concatenated data sets, and their volume serial numbers are placed in the area located at the address specified in the allocation retrieval list. The DCB exit list entry contains the address of the allocation retrieval list. The RDJFCB macro passes the following return codes in register 15:

Return Code	Meaning
0 (X'00')	RDJFCB has completed the allocation retrieval list successfully.
4 (X'04')	One or more DCBs had one of the following conditions and were skipped: • DCB currently being processed by O/C/EOV or a similar function. • No data set is allocated with the ddname that is in the DCB. • The DCB is not open and its ddname is blank. DCBs that were not skipped were handled successfully.
8 (X'08')	One or more DCBs had an allocation retrieval list which could not be handled. Each allocation retrieval list contains a reason code to describe its status. One or more DCBs may have an error described by return code 4, in which case their allocation retrieval lists will not have a reason code.

For more information about the RDJFCB macro, see *z/OS DFSMSdfp Advanced Services*.

Programming Conventions

The allocation retrieval list must be below the 16 MB line, but the allocation return area can be above the 16 MB line.

When you are finished obtaining information from the retrieval areas, free the storage with a FREEMAIN or STORAGE macro.

You can use the IHAARL macro to generate and map the allocation retrieval list. For more information about the IHAARL macro see *z/OS DFSMSdfp Advanced Services*.

Restrictions

When OPEN TYPE=J is issued, the X'13' exit has no effect. The JFCB exit at X'07' can be used instead (see “JFCB Exit” on page 513).

DCB ABEND Exit

The DCB ABEND exit is provided to give you some options regarding the action you want the system to take when a condition occurs that may result in abnormal termination of your task. This exit can be taken any time an abend condition occurs during the process of opening, closing, or handling an end-of-volume condition for a DCB associated with your task. The exit is taken only for determinate errors that the system can associate with the DCB.

When an abend condition occurs, a write-to-programmer message about the abend is issued and your DCB abend exit is given control, provided there is an active DCB abend exit routine address in the exit list contained in the DCB being processed. If STOW called the end-of-volume routines to get secondary space to write an end-of-file mark for a PDS, or if the DCB being processed is for an indexed sequential data set, the DCB abend exit routine is not given control if an abend condition occurs. When your exit routine is entered the contents of the registers are the same as for other DCB exit list routines, except that the 3 low-order bytes of register 1 contain the address of the parameter list described in Figure 108.

Offset	Length or Bit Pattern	Description
0(0)	2	System completion code in first 12 bits.
2(2)	1	Return code associated with system completion code. For example, with abend 213-30, this byte will have X'30'.
Input to exit:		
3(3)	1	Option mask.
	xxxx ...x	Reserved.
	.... 1...	Okay to recover.
	.... .1..	Okay to ignore.
	.... ..1.	Okay to delay.
Output from exit:		
3(3)	1	Option. See Table 43 on page 506
Input to exit:		
4(4)	4	Address of DCB.
8(8)	4	For system diagnostic use.
12(C)	4	Address of recovery work area. Must be below 16 MB.

Figure 108. Parameter List Passed to DCB Abend Exit Routine

Your ABEND exit routine can choose one of four options:

1. To terminate your task immediately
2. To delay the abend until all the DCBs in the same OPEN or CLOSE macro are opened or closed
3. To ignore the abend condition and continue processing without making reference to the DCB on which the abend condition was encountered, or
4. To try to recover from the error.

Not all of these options are available for each abend condition. Your DCB ABEND exit routine must determine which option is available by examining the contents of the option mask byte (byte 3) of the parameter list. The address of the parameter

list is passed in register 1. Figure 108 shows the contents of the parameter list and the possible settings of the option mask when your routine receives control.

When your DCB ABEND exit routine returns control to the system control program (this can be done using the RETURN macro), the option mask byte must contain the setting that specifies the action you want to take. These actions and the corresponding settings of the option mask byte are in Table 43.

Table 43. Option Mask Byte Settings

Decimal Value	Action
0	Abnormally terminate the task immediately.
4	Ignore the abend condition.
8	Delay the abend until the other DCBs being processed concurrently are opened or closed.
12	Make an attempt to recover.

Your exit routine must inspect bits 4, 5, and 6 of the option mask byte (byte 3 of the parameter list) to determine which options are available. If a bit is set to 1, the corresponding option is available. Indicate your choice by inserting the appropriate value in byte 3 of the parameter list, overlaying the bits you inspected. If you use a value that specifies an option that is not available, the abend is issued immediately.

If the contents of bits 4, 5, and 6 of the option mask are 0, you must not change the option mask. This unchanged option mask results in a request for an immediate abend.

If bit 5 of the option mask is set to 1, you can ignore the abend by placing a value of 4 in byte 3 of the parameter list. Processing on the current DCB stops and bit DCBOFOPN is off. There is no need to issue CLOSE. If you subsequently attempt to use this DCB other than to issue CLOSE or FREEPOOL, the results are unpredictable. If you ignore an error in end-of-volume, the DCB is closed and control is returned to your program at the point that caused the end-of-volume condition (unless the end-of-volume routines were called by the CLOSE routines). If the end-of-volume routines were called by the CLOSE routines, an ABEND macro is issued even though the IGNORE option was selected.

If bit 6 of the option mask is set to 1, you can delay the abend by placing a value of 8 in byte 3 of the parameter list. All other DCBs being processed by the same OPEN or CLOSE invocation will be processed before the abend is issued. For end-of-volume, however, you can’t delay the abend because the end-of-volume routine never has more than one DCB to process.

If bit 4 of the option mask is set to 1, you can attempt to recover. Place a value of 12 in byte 3 of the parameter list and provide information for the recovery attempt.

Table 44 on page 507 lists the abend conditions for which recovery can be attempted. See *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)*; and *z/OS MVS System Codes*.

Table 44. Conditions for Which Recovery Can Be Attempted

Completion Code	Return Code	Description of Error
117	X'38'	An I/O error occurred during execution of a read block ID command issued to establish tape position.
214	X'10'	DCB block count did not agree with the calculated data block count for the tape data set.
137	X'24'	A specific volume serial number was specified for the second or subsequent volume of an output data set on magnetic tape. During EOV processing, it was discovered that the expiration date (from the HDR1 label of the first data set currently on the specified volume) had not passed. When requested to specify if the volume could be used despite the expiration date, the operator did not reply **U**.
214	X'0C'	An I/O error occurred during execution of a read block ID command issued to establish tape position.
237	X'04'	Block count in DCB does not agree with block count in trailer label.
	X'0C'	DCB block count did not agree with the calculated block count on a cartridge.
413	X'18'	Data set was opened for input and no volume serial number was specified.
	X'24'	**LABEL=(n)** was specified, where **n** was greater than 1 and **vol=ser** was not specified for a tape data set.
613	X'08'	I/O error occurred during reading of tape label.
	X'0C'	Invalid tape label was read.
	X'10'	I/O error occurred during writing of tape label.
	X'14'	I/O error occurred during writing of tape label.
713	X'04'	A data set on magnetic tape was opened for INOUT, but the volume contained a data set whose expiration date had not been reached and the operator denied permission.
717	X'10'	I/O error occurred during reading of trailer label 1 to update block count in DCB.
737	X'28'	The EOV DA module was passed an error return code in register 15 after issuing the IEFSSREQ macro instruction. This indicates the subsystem (JES3) discovered a functional or logical error that it could not process.
813	X'04'	Data set name on header label does not match data set name on DD statement.

Recovery Requirements

For most types of recoverable errors, you should supply a recovery work area (see Figure 109) with a new volume serial number for each volume associated with an error.

```
                Length or
Offset      Bit Pattern        Description

0           2                  Length of this work area.
2           1                  Option byte.
            1... ....          Free this work area.
            .1.. ....          Volume serial numbers provided.
            ..xx xxxx          Reserved.
3           1                  Subpool number.
4           1                  Number of volumes that follow.
5           n*6                New volume serial numbers (six bytes each)
```

Figure 109. Recovery Work Area

If no new volumes are supplied for such errors, recovery will be attempted with the existing volumes, but the likelihood of successful recovery is greatly reduced.

If you request recovery for system completion code 117, return code 3C, or system completion code 214, return code 0C, or system completion code 237, return code 0C, you do not need to supply new volumes or a work area. The condition that caused the abend is disagreement between the DCB block count and the calculated count from the hardware. To permit recovery, this disagreement is ignored and the value in the DCB is used.

If you request recovery for system completion code 237, return code 04, you don't need to supply new volumes or a work area. The condition that caused the abend is the disagreement between the block count in the DCB and that in the trailer label. To permit recovery, this disagreement is ignored.

If you request recovery for system completion code 717, return code 10, you don't need to supply new volumes or a work area. The abend is caused by an I/O error during updating of the DCB block count. To permit recovery, the block count is not updated. So, an abnormal termination with system completion code 237, return code 04, may result when you try to read from the tape after recovery. You may attempt recovery from the abend with system completion code 237, return code 04, as explained in the preceding paragraph.

System completion codes and their associated return codes are described in *z/OS MVS System Codes*.

The work area that you supply for the recovery attempt must begin on a halfword boundary and can contain the information described in Figure 109. Place a pointer to the work area in the last 3 bytes of the parameter list pointed to by register 1 and described in Figure 108 on page 505.

If you acquire the storage for the work area by using the GETMAIN macro, you can request that it be freed by a FREEMAIN macro after all information has been extracted from it. Set the high-order bit of the option byte in the work area to 1 and place the number of the subpool from which the work area was requested in byte 3 of the recovery work area.

Only one recovery attempt per data set is permitted during OPEN, CLOSE, or end-of-volume processing. If a recovery attempt is unsuccessful, you can not request another recovery. The second time through the exit routine you may request only one of the other options (if allowed): Issue the abend immediately, ignore the abend, or delay the abend. If at any time you select an option that is not permitted, the abend is issued immediately.

If recovery is successful, you still receive an abend message on your listing. This message refers to the abend that would have been issued if the recovery had not been successful.

DCB Abend Installation Exit

The DCB abend installation exit gives your installation an additional option for handling error situations that result in an abend. This exit is taken any time an abend condition occurs during the process of opening, closing, or handling an end-of-volume condition for a DCB. An IBM-supplied installation exit routine gives your installation the option to retry tape positioning when you receive a 613 system

completion code, return code 08 or 0C. For more information about the DCB abend installation exit, see *z/OS DFSMS Installation Exits*.

DCB OPEN Exit

You can specify in an exit list the address of a routine that completes or modifies a DCB and does any additional processing required before the data set is completely open. The routine is entered during the opening process after the JFCB has been used to supply information for the DCB. “Filling in the DCB” on page 319 describes other functions performed by OPEN before and after the DCB OPEN exit. The routine can determine data set characteristics by examining fields completed from the data set labels. When your DCB exit routine receives control, the 3 low-order bytes of register 1 will contain the address of the DCB currently being processed. See “Changing and Testing the DCB and DCBE” on page 330.

When opening a data set for output and the record format is fixed or variable, you can force the system to calculate an optimal block size by setting the block size in the DCB or DCBE to zero before returning from this exit. The system uses DCB block size if it is not using the large block interface (LBI). See “Large Block Interface (LBI)” on page 323. If the zero value you supply is not changed by the DCB OPEN installation exit, OPEN determines a block size when OPEN takes control after return from the DCB OPEN installation exit. See “System-Determined Block Size” on page 324.

As with label processing routines, the contents of register 14 must be preserved and restored if any macros are used in the routine. Control is returned to the operating system by a RETURN macro; no return code is required.

This exit is mutually exclusive with the JFCBE exit. If you need both the JFCBE and DCB OPEN exits, you must use the JFCBE exit to pass control to your routines.

The DCB OPEN exit is intended for modifying or updating the DCB. System functions should not be attempted in this exit before returning to OPEN processing. In particular, dynamic allocation, OPEN, CLOSE, EOV, and DADSM functions should not be invoked because of an existing OPEN enqueue on the SYSZTIOT resources.

Calls to DCB OPEN Exit for Sequential Concatenation

If your program uses *like* sequential concatenation processing, the system calls your DCB OPEN exit only for the first data set and calls your EOV exit for the beginning of each subsequent data set and for each disk or tape volume after reading the first volume of each data set. If your program uses *unlike* sequential concatenation, the system calls your DCB OPEN exit at the beginning of each data set and calls your EOV exit only for each volume of each disk or tape data set after the first volume of the data set.

Installation DCB OPEN Exit

After the system calls your application’s optional DCB OPEN exit or JFCBE exit, it calls the installation DCB OPEN exit. That exit can augment or override your application’s DCB OPEN exit. See *z/OS DFSMS Installation Exits*.

Defer Nonstandard Input Trailer Label Exit List Entry

In an exit list, you can specify a code that indicates that you want to defer nonstandard input trailer label processing from end-of-data until the data set is closed. The address portion of the entry is not used by the operating system. This exit list entry has an effect only when reading magnetic tape that has nonstandard labels. You specified LABEL=(x,NSL) on the DD statement.

An end-of-volume condition exists in several situations. Two examples are: (1) when the system reads a tape mark at the end of a volume of a multivolume data set but that volume is not the last, and (2) when the system reads a tape mark at the end of a data set. The first situation is referred to here as an end-of-volume condition, and the second as an end-of-data condition, although it, too, can occur at the end of a volume.

For an end-of-volume (EOV) condition, the EOV routine passes control to your installation's nonstandard input trailer label routine, whether or not this exit code is specified. For an end-of-data condition when this exit code is specified, the EOV routine does not pass control to your installation's nonstandard input trailer label routine. Instead, the CLOSE routine passes control to your installation's nonstandard input trailer label.

Block Count Unequal Exit

In an exit list you can specify the address of a routine that lets you abnormally terminate the task or continue processing when the EOV routine finds an unequal block count condition. When you are using IBM standard or ISO/ANSI standard labeled input tapes, the EOV function compares the block count in the trailer label with the block count in the DCB. The count in the trailer label reflects the number of blocks written when the data set was created. The number of blocks read when the tape is used as input is contained in the DCBBLKCT field of the DCB.

When the system reads or writes any kind of cartridge tape, it calls the block-count-unequal exit if the DCB block count does not match the block count calculated for the cartridge. The EOV and CLOSE functions perform these comparisons for cartridges, even for unlabeled tapes and for writes. The result can be a 117-3C or 237-0C ABEND, but the system calls your optional DCB ABEND exit.

The routine is entered during EOV processing. The trailer label block count is passed in register 0. You can gain access to the count field in the DCB by using the address passed in register 1 plus the proper displacement, which is shown in *z/OS DFSMS Macro Instructions for Data Sets*. If the block count in the DCB differs from that in the trailer label when no exit routine is provided or your exit gives return code 0, the system calls your optional DCB abend exit and possibly your installation's DCB abend exit. If these exits do not exist or they allow abnormal end, the task is abnormally terminated. The routine must terminate with a RETURN macro and a return code that indicates what action is to be taken by the operating system, as shown in Table 45.

Table 45. System Response to Block Count Exit Return Code

Return Code	System Action
0 (X'00')	The task is to be abnormally terminated with system completion code 237, return code 4.
4 (X'04')	Normal processing is to be resumed.

As with other exit routines, the contents of register 14 must be saved and restored if any macros are used.

EOV Exit for Sequential Data Sets

You can specify in an exit list the address of a routine that is entered when end of volume is reached in processing of a physical sequential data set and the system finds either of these conditions:

- There is another tape or DASD volume for the data set.
- You reached the end of the data set, another is concatenated and your program did not have on the DCB unlike-attributes bit.

When you concatenate data sets with unlike attributes, no EOV exits are taken when beginning each data set.

The system treats the volumes of a striped extended format data set as if they were one volume. For such a data set your EOV exit is called only when the end of the data set is reached and it is part of a like sequential concatenation.

When the EOV routine is entered, register 0 contains 0 unless user totaling was specified. If you specified user totaling in the DCB macro (by coding OPTCD=T) or in the DD statement for an output data set, register 0 contains the address of the user totaling image area.

The routine is entered after the next volume has been positioned and all necessary label processing has been completed. If the volume is a reel or cartridge of magnetic tape, the tape is positioned after the tape mark that precedes the beginning of the data.

You can use the EOV exit routine to take a checkpoint by issuing the CHKPT macro (see *z/OS DFSMS Checkpoint/Restart*). If a checkpointed job step terminates abnormally, it can be restarted from the EOV checkpoint. When the job step is restarted, the volume is mounted and positioned as on entry to the routine. Restart becomes impossible if changes are made to the link pack area (LPA) library between the time the checkpoint is taken and the job step is restarted. When the EOV exit is entered, register 1 contains the address of the DCB. Registers 2 - 13 contain the contents when your program issued the macro that resulted in the EOV condition. Register 14 has the return address. When the step is restarted, pointers to EOV modules must be the same as when the checkpoint was taken.

The EOV exit routine returns control in the same manner as the DCB exit routine. The contents of register 14 must be preserved and restored if any macros are used in the routine. Control is returned to the operating system by a RETURN macro; no return code is required.

FCB Image Exit

You can specify in an exit list the address of a forms control buffer (FCB) image. This FCB image can be loaded into the forms control buffer of the printer control unit. The FCB controls the movement of forms in printers that do not use a carriage control tape.

Identifying Non-VSAM User-Written Exit Routines

Multiple exit list entries in the exit list can define FCBs. The OPEN and SETPRT routines search the exit list for requested FCBs before searching SYS1.IMAGELIB.

The first 4 bytes of the FCB image contain the image identifier. To identify the FCB, this image identifier is specified in the FCB parameter of the DD statement, by your JFCBE exit, by the SETPRT macro, or by the system operator in response to message IEC127D or IEC129D.

For an IBM 3203, 3211, 3262, 4245, or 4248 Printer, the image identifier is followed by the FCB image described in *z/OS DFSMSdfp Advanced Services*.

You can create, modify, and list FCB images in libraries with the IEBIMAGE utility and the CIPOPS utility. IEBIMAGE is described in *z/OS DFSMSdfp Utilities*.

The system searches the DCB exit list for an FCB image only when writing to a printer that is allocated to the job step. The system does not search the DCB exit list with a SYSOUT data set. Figure 110 shows one way the exit list can be used to define an FCB image.

```
                ...
                DCB    ..,EXLST=EXLIST
                ...
EXLIST          DS     0F
                DC     X'10'           Flag code for FCB image
                DC     AL3(FCBIMG)     Address of FCB image
                DC     X'80000000'     End of EXLST and a null entry
FCBIMG          DC     CL4'IMG1'       FCB identifier
                DC     X'00'           FCB is not a default
                DC     AL1(67)         Length of FCB
                DC     X'90'           Offset print line
* 16 line character positions to the right
                DC     X'00'           Spacing is 6 lines per inch
                DC     5X'00'          Lines 2-6, no channel codes
                DC     X'01'           Line 7, channel 1
                DC     6X'00'          Lines 8-13, no channel codes
                DC     X'02'           Line (or Lines) 14, channel 2
                DC     5X'00'          Line (or Lines) 15-19, no channel codes
                DC     X'03'           Line (or Lines) 20, channel 3
                DC     9X'00'          Line (or Lines) 21-29, no channel codes
                DC     X'04'           Line (or Lines) 30, channel 4
                DC     19X'00'         Line (or Lines) 31-49, no channel codes
                DC     X'05'           Line (or Lines) 50, channel 5
                DC     X'06'           Line (or Lines) 51, channel 6
                DC     X'07'           Line (or Lines) 52, channel 7
                DC     X'08'           Line (or Lines) 53, channel 8
                DC     X'09'           Line (or Lines) 54, channel 9
                DC     X'0A'           Line (or Lines) 55, channel 10
                DC     X'0B'           Line (or Lines) 56, channel 11
                DC     X'0C'           Line (or Lines) 57, channel 12
                DC     8X'00'          Line (or Lines) 58-65, no channel codes
                DC     X'10'           End of FCB image
                ...
                END
//ddname        DD     UNIT=3211,FCB=(IMG1,VERIFY)
/*
```

Figure 110. Defining an FCB Image for a 3211

JFCB Exit

This exit list entry does not define an exit routine. It is used with the RDJFCB macro and OPEN TYPE=J. The RDJFCB macro uses the address specified in the DCB exit list entry at X'07' to place a copy of the JFCB for each DCB specified by the RDJFCB macro.

The area is 176 bytes and must begin on a fullword boundary. It must be located in the user's address space. This area must be located below 16 MB virtual. The DCB can be either open or closed when the RDJFCB macro is executed.

If RDJFCB fails while processing a DCB associated with your RDJFCB request, your task is abnormally terminated. You cannot use the DCB abend exit to recover from a failure of the RDJFCB macro. See *z/OS DFSMSdfp Advanced Services*.

JFCBE Exit

JCL-specified setup requirements for the IBM 3800 and 3900 Printing Subsystem cause a JFCB extension (JFCBE) to be created to reflect those specifications. Your JFCBE exists if BURST, MODIFY, CHARS, FLASH, or any copy group is coded on the DD statement. The JFCBE exit can examine or modify those specifications in the JFCBE.

Although use of the JFCBE exit is still supported, its use is not recommended.

Place the address of the routine in an exit list. The device allocated does not have to be a printer. This exit is taken during OPEN processing and is mutually exclusive with the DCB OPEN exit. If you need both the JFCBE and DCB OPEN exits, you must use the JFCBE exit to pass control to your routines. Everything that you can do in a DCB OPEN exit routine can also be done in a JFCBE exit. See "DCB OPEN Exit" on page 509. When you issue the SETPRT macro to a SYSOUT data set, the JFCBE is further updated from the information in the SETPRT parameter list.

When control is passed to your exit routine, the contents of register 1 will be the address of the DCB being processed.

The area pointed to by register 0 will contain a 176 byte JFCBE followed by the 4 byte FCB identification that is obtained from the JFCB. If the FCB operand was not coded on the DD statement, this FCB field will be binary zeros.

If your exit routine modifies your copy of the JFCBE, you should indicate this by turning on bit JFCBEOPN (X'80' in JFCBFLAG) in the JFCBE copy. On return to OPEN, this bit indicates if the system copy is to be updated. The 4-byte FCB identification in your area is used to update the JFCB regardless of the bit setting. Checkpoint/restart also interrogates this bit to determine which version of the JFCBE to use at restart time. If this bit is not on, the JFCBE generated by the restart JCL is used.

Open/Close/EOV Standard User Label Exit

When you create a data set with physical sequential or direct organization, you can provide routines to create your own data set labels to augment the system's labels. You can also provide routines to verify these labels when you use the data set as input. Each label is 80 characters long, with the first four characters UHL1,UHL2, through UHL8 for a header label or UTL1,UTL2,...,UTL8 for a trailer label. User labels are not permitted on partitioned, indexed sequential, spooled, or extended format data sets or HFS files.

The physical location of the labels on the data set depends on the data set organization. For direct (BDAM) data sets, user labels are placed on a separate user label track in the first volume. User label exits are taken only during execution of the OPEN and CLOSE routines. Thus you can create or examine as many as eight user header labels only during execution of OPEN and as many as eight trailer labels only during execution of CLOSE. Because the trailer labels are on the same track as the header labels, the first volume of the data set must be mounted when the data set is closed.

For physical sequential (BSAM or QSAM) data sets on DASD or tape with IBM standard labels, you can create or examine as many as eight header labels and eight trailer labels on each volume of the data set. For ISO/ANSI tape label data sets, you can create an unlimited number of user header and trailer labels. The user label exits are taken during OPEN, CLOSE, and EOV processing.

To create or verify labels, you must specify the addresses of your label exit routines in an exit list as shown in Table 42 on page 502. Thus you can have separate routines for creating or verifying header and trailer label groups. Care must be taken if a magnetic tape is read backward, because the trailer label group is processed as header labels and the header label group is processed as trailer labels.

When your routine receives control, the contents of register 0 are unpredictable. Register 1 contains the address of a parameter list. The contents of registers 2 to 13 are the same as when the macro instruction was issued. However, if your program does not issue the CLOSE macro, or abnormally ends before issuing CLOSE, the CLOSE macro will be issued by the control program, with control-program information in these registers.

The parameter list pointed to by register 1 is a 16 byte area aligned on a fullword boundary. Figure 111 shows the contents of the area:

0		Address of 80-byte label buffer area
4	EOF flag	Address of DCB being processed
8	Error flags	Address of status information
12		Address of user totaling image area

Figure 111. Parameter List Passed to User Label Exit Routine

The first address in the parameter list points to an 80-byte label buffer area. The format of a user label is described in “User Label Groups” on page 530. For input, the control program reads a user label into this area before passing control to the label routine. For output, your user label exit routine builds labels in this area and returns to the control program, which writes the label. When an input trailer label routine receives control, the EOF flag (high-order byte of the second word in the parameter list) is set as follows:

```
Bit 0 = 0: Entered at EOV
Bit 0 = 1: Entered at end-of-file
Bits 1-7:  Reserved
```

When a user label exit routine receives control after an uncorrectable I/O error has occurred, the third word of the parameter list contains the address of the standard status indicators. The error flag (high-order byte of the third word in the parameter list) is set as follows:

```
Bit 0 = 1: Uncorrectable I/O error
Bit 1 = 1: Error occurred during writing of updated label
Bits 2-7:  Reserved
```

The fourth entry in the parameter list is the address of the user totaling image area. This image area is the entry in the user totaling save area that corresponds to the last record physically written on the volume. (The image area is discussed in “User Totaling for BSAM and QSAM” on page 523.)

Each routine must create or verify one label of a header or trailer label group, place a return code in register 15, and return control to the operating system. The operating system responds to the return code as shown in Table 46 on page 516.

You can create user labels only for data sets on magnetic tape volumes with IBM standard labels or ISO/ANSI labels and for data sets on direct access volumes. When you specify both user labels and IBM standard labels in a DD statement by specifying LABEL=(,SUL) and there is an active entry in the exit list, a label exit is always taken. Thus, a label exit is taken even when an input data set does not contain user labels, or when no user label track has been allocated for writing labels on a direct access volume. In either case, the appropriate exit routine is entered with the buffer area address parameter set to 0. On return from the exit routine, normal processing is resumed; no return code is necessary.

Table 46. System Response to a User Label Exit Routine Return Code

Routine Type	Return Code	System Response
Input header or trailer label	0 (X'00')	Normal processing is resumed. If there are any remaining labels in the label group, they are ignored.
	4 (X'04')	The next user label is read into the label buffer area and control is returned to the exit routine. If there are no more labels in the label group, normal processing is resumed.
	8[1] (X'08')	The label is written from the label buffer area and normal processing is resumed.
	12[1] (X'0C')	The label is written from the label area, the next label is read into the label buffer area, and control is returned to the label processing routine. If there are no more labels, processing is resumed.
Output header or trailer label	0 (X'00')	Normal processing is resumed; no label is written from the label buffer area.
	4 (X'04')	User label is written from the label buffer area. Normal processing is resumed.
	8 (X'08')	User label is written from the label buffer area. If fewer than eight labels have been created, control is returned to the exit routine, which then creates the next label. If eight labels have been created, normal processing is resumed.

Note:

1. Your input label routines can return these codes only when you are processing a physical sequential data set opened for UPDAT or a direct data set opened for OUTPUT or UPDAT. These return codes let you verify the existing labels, update them if necessary, and then request that the system write the updated labels.

Label exits are not taken for system output (SYSOUT) data sets, or for data sets on volumes that do not have standard labels. For other data sets, exits are taken as follows:

- When an input data set is opened, the input header label exit 01 is taken. If the data set is on tape being opened for RDBACK, user trailer labels will be processed.
- When an output data set is opened, the output header label exit 02 is taken. However, if the data set already exists and DISP=MOD is coded in the DD statement, the input trailer label exit 03 is taken to process any existing trailer labels. If the input trailer label exit 03 does not exist, then the deferred input trailer label exit 0C is taken if it exists; otherwise, no label exit is taken. For tape, these trailer labels will be overwritten by the new output data or by EOV or close processing when writing new standard trailer labels. For direct access devices, these trailer labels will still exist unless rewritten by EOV or close processing in an output trailer label exit.
- When an input data set reaches EOV, the input trailer label exit 03 is taken. If the data set is on tape opened for RDBACK, header labels will be processed. The input trailer label exit 03 is not taken if you issue an FEOV macro. If a defer input trailer label exit 0C is present, and an input trailer label exit 03 is not present, the 0C exit is taken. After switching volumes, the input header label exit 01 is taken. If the data set is on tape opened for RDBACK, trailer labels will be processed.
- When an output data set reaches EOV, the output trailer label exit 04 is taken. After switching volumes, output header label exit 02 is taken.
- When an input data set reaches end-of-data, the input trailer label exit 03 is taken before the EODAD exit, unless the DCB exit list contains a defer input trailer label exit 0C.
- When an input data set is closed, no exit is taken unless the data set was previously read to end-of-data and the defer input trailer label exit 0C is present. If so, the defer input trailer label exit 0C is taken to process trailer labels, or if the tape is opened for RDBACK, header labels.
- When an output data set is closed, the output trailer label exit 04 is taken.

To process records in reverse order, a data set on magnetic tape can be read backward. When you read backward, header label exits are taken to process trailer labels, and trailer label exits are taken to process header labels. The system presents labels from a label group in ascending order by label number, which is the order in which the labels were created. If necessary, an exit routine can determine label type (UHL or UTL) and number by examining the first four characters of each label. Tapes with IBM standard labels and direct access devices can have as many as eight user labels. Tapes with ISO/ANSI labels can have an unlimited number of user labels.

After an input error, the exit routine must return control with an appropriate return code (0 or 4). No return code is required after an output error. If an output error occurs while the system is opening a data set, the data set is not opened (DCB is flagged) and control is returned to your program. If an output error occurs at any other time, the system attempts to resume normal processing.

Open/EOV Nonspecific Tape Volume Mount Exit

This user exit gives you the option of identifying a specific tape volume to be requested in place of a nonspecific (scratch) tape volume. An X'17' in the DCB exit list (EXLST) activates this exit (see “DCB Exit List” on page 501). This exit, which supports only IBM standard labeled tapes, was designed to be used with the Open/EOV volume security and verification user exit. However, this exit can be used by itself.

Open or EOV calls this exit when either must issue mount message IEC501A or IEC501E to request a scratch tape volume. Open issues the mount message if you specify the DEFER parameter with the UNIT option, and either you did not specify a volume serial number in the DD statement or you specified 'VOL=SER=SCRTCH'. EOV always calls this exit for a scratch tape volume request.

This user exit gets control in the key and state of the program that issued the OPEN or EOV, and no locks are held. This exit must provide a return code in register 15.

Return Code	Meaning
00 (X'00')	Continue with the scratch tape request as if this exit had not been called.
04 (X'04')	Replace the scratch tape request with a specific volume serial number. Register 0 contains the address of a 6-byte volume serial number.

Note: A value other than 0 or 4 in register 15 is treated as a 0.

If OPEN or EOV finds that the volume pointed to by register 0 is being used either by this or by another job (an active ENQ on this volume), it calls this exit again and continues to do so until you either specify an available volume serial number or request a scratch volume. If the volume you specify is available but is rejected by OPEN or EOV for some other reason (I/O errors, expiration date, password check, and so forth), this exit is not called again.

When this exit gets control, register 1 points to the parameter list described by the IECOENTE macro. Figure 112 on page 519 shows this parameter list.

Offset	Length or Bit Pattern	Description	
0	4	OENTID	PLIST ID ('OENT')
4	1	OENTFLG	FLAG BYTES
	1...	OENTOEOV	Set to 0 if OPEN called this exit; set to 1 if EOV or FEOV called this exit
	1	OENTNTRY	Set to 1 if this is not the first time this exit was called because the requested tape volume is being used by this or any other job
5	1	OENTOPTN	Contains the options from the OPEN parameter list (OUTPUT, INPUT, OUTIN, INOUT, and so forth). For EOV processing, the options byte in the DCB parameter list indicates how EOV is processings this volume. For example, if you open a tape volume for OUTIN and EOV is called during an output operation on this tape volume, OENTOPTN is set to indicate OUTPUT. Possible values follow: xxxx 0000 INPUT or reading at EOV with INOUT, OUTIN, or OUTINX xxxx 0001 RDBACK xxxx 1111 OUTPUT or EXTEND or writing at EOV with INOUT, OUTIN, or OUTINX xxxx 0011 INOUT during OPEN xxxx 0111 OUTIN or OUTINX during OPEN
	xxxx	RESERVED	
	0000 1111	OENTMASK	TO MASK OFF UNNECESSARY BITS
6	2	OENTRSVD	RESERVED
8	4	OENTDCBA	ADDRESS OF USER DCB
12(X'C')	4	OENTVSRA	Points to the last volume serial number you requested in this exit but was in use either by this or another job. OENTVSRA is set to 0 the first time this exit is called.
16(X'10')	4	OENTJFCB	Points to the OPEN or EOV copy of the JFCB. The high order bit is always on, indicating that this is the end of the parameter list.
		OENTLENG	PLIST LENGTH (Current value is 20.)

Figure 112. IECOENTE Macro Parameter List

Identifying Non-VSAM User-Written Exit Routines

When this user exit is entered, the general registers contain the information in Table 47 for saving and restoring.

Table 47. Saving and Restoring General Registers

Register	Contents
0	Variable
1	Address of the parameter list for this exit
2-13	Contents of the registers before the OPEN, FEOV, or EOV was issued
14	Return address (you must preserve the contents of this register in this user exit)
15	Entry point address to this user exit

You do not have to preserve the contents of any register other than register 14. The operating system restores the contents of registers 2 through 13 before it returns to OPEN or EOV and before it returns control to the original calling program.

Do not use the save area pointed to by register 13; the operating system uses it. If you call another routine, or issue a supervisor or data management macro in this user exit, you must provide the address of a new save area in register 13.

Open/EOV Volume Security and Verification Exit

This user exit lets you verify that the volume that is currently mounted is the one you want. You can also use it to bypass the OPEN or EOV expiration date, password, and data set name security checks. An X'18' in the DCB exit list (EXLST) activates this exit (see “DCB Exit List” on page 501). This exit, which supports IBM standard label tapes, was designed to be used with the OPEN/EOV nonspecific tape volume mount user exit, but you can use this exit by itself (see “Open/EOV Nonspecific Tape Volume Mount Exit” on page 518).

This exit is available only for APF-authorized programs.

This user exit gets control in the key and state of the program that issued the OPEN or EOV request, and no locks are held. This exit must provide a return code in register 15.

Return Code	Meaning
00 (X'00')	Use this tape volume. Return to OPEN or EOV as if this exit had not been called.
04 (X'04')	Reject this volume and: • Output – If the data set is the first data set on the volume, request a scratch tape. This causes OPEN or EOV to issue demount message IEC502E for the rejected tape volume, and mount message IEC501A or IEC501E for a scratch tape volume. If the nonspecific tape volume mount exit is active, it is called. – If the data set is other than the first one on the volume, process this return code as if it were return code 08. • Input – Treat this return code as if it were return code 08.
08 (X'08')	Abnormally terminate OPEN or EOV unconditionally; no scratch tape request is issued. OPEN abnormally terminates with a 913-34 ABEND code, and EOV terminates with a 937-44 ABEND code.
12 (X'0C')	Use this volume without checking the data set’s expiration date. Password, RACF authority, and data set name checking still occurs.
16 (X'10')	Use this volume. A conflict with the password, label expiration date, or data set name does not prevent the new data set from writing over the current data set if it is the first one on the volume. To write over other than the first data set, the new data set must have the same level of security protection as the current data set.

When this exit gets control, register 1 points to the parameter list described by the IECOEVSE macro. The parameter list is shown in Figure 113 on page 522.

Offset	Length or Bit Pattern	Description	
0		OEVSE	DSECT name
0	4	OEVSID	ID field = "OEVS"
4	1	OEVSFLG	A flag field
	1...	OEVSEOV	Set to 0 if OPEN called this exit and set to 1 if EOV called this exit
	1	OEVSFILE	Set to 0 if the first data set on the volume is to be written and set to 1 if this is not the first data set on the volume to be written. This bit is always 0 for INPUT processing.
	.xxx xxx.		Bits 1 through 6 reserved
5	1	OEVSOPTN	OPEN options from the DCB parameter list (OUTPUT, INPUT,INOUT, and so forth). For EOV processing, this byte indicates how EOV is processing this volume. For example, if you opened a tape volume for OUTIN and EOV is called during an output operation on the tape volume, the DCB parameter list and OEVSOPTN are set to indicate OUTPUT. Possible values follow: xxxx 0000 INPUT or reading at EOV with INOUT, OUTIN, or OUTINX xxxx 0001 RDBACK xxxx 1111 OUTPUT or EXTEND or writing at EOV with INOUT, OUTIN, or OUTINX xxxx 0011 INOUT during OPEN xxxx 0111 OUTIN or OUTINX during OPEN
	0000 1111	OEVSMASK	Mask
6	2	OEVSRSVD	Reserved
8	4	OEVSDCBA	Address of user DCB
12(X'C')	4	OEVSVSRA	A pointer to the current volume serial number that OPEN or EOV is processing
16(X'10')	4	OEVSHDR1	A pointer to an HDR1 label, if one exists, or to an EOF1 label, if you are creating other than the first data set on this volume
20(X'14')	4	OEVSJFCB	A pointer to the OPEN, CLOSE, or EOV copy of the JFCB. The high-order bit is always on, indicating that this is the end of the parameter list.
	24	OEVSLENG	OEVSID PLIST LENGTH

Figure 113. IECOEVSE Macro Parameter List

When this user exit is entered, the general registers have the following contents.

Register	Contents
0	Variable
1	Address of the parameter list for this exit.
2-13	Contents of the registers before the OPEN or EOV was issued
14	Return address (you must preserve the contents of this register in this user exit)
15	Entry point address to this user exit

You do not have to preserve the contents of any register other than register 14. The operating system restores the contents of registers 2 through 13 before it returns to OPEN or EOV and before it returns control to the original calling program.

Do not use the save area pointed to by register 13; the operating system uses it. If you call another routine or issue a supervisor or data management macro in this user exit, you must provide the address of a new save area in register 13.

QSAM Parallel Input Exit

QSAM parallel input processing can be used to process two or more input data sets concurrently, such as sorting or merging several data sets at the same time.

A request for parallel input processing is indicated by including the address of a parallel data access block (PDAB) in the DCB exit list. The address must be on a fullword boundary with the first byte of the entry containing X'12' or, if it is the last entry, X'92'. For more information about parallel input processing see “PDAB—Parallel Input Processing (QSAM Only)” on page 360.

User Totaling for BSAM and QSAM

When creating or processing a data set with user labels, you can develop control totals for each volume of the data set and store this information in your user labels. For example, a control total that was accumulated as the data set was created can be stored in your user label and later compared with a total accumulated during processing of the volume. User totaling helps you by synchronizing the control data you create with records physically written on a volume. For an output data set without user labels, you can also develop a control total that is available to your EOV routine.

User totaling is ignored for extended format data sets and HFS files.

To request user totaling, you must specify OPTCD=T in the DCB macro instruction or in the DCB parameter of the DD statement. The area in which you collect the control data (the user totaling area) must be identified to the control program by an entry of X'0A' in the DCB exit list. OPTCD=T cannot be specified for SYSIN or SYSOUT data sets.

The user totaling area, an area in storage that you provide, must begin on a halfword boundary and be large enough to contain your accumulated data plus a 2 byte length field. The length field must be the first 2 bytes of the area and specify the length of the complete area. A data set for which you have specified user totaling (OPTCD=T) will not be opened if either the totaling area length or the address in the exit list is 0, or if there is no X'0A' entry in the exit list.

Identifying Non-VSAM User-Written Exit Routines

The control program establishes a user totaling save area, where the control program preserves an image of your totaling area, when an I/O operation is scheduled. When the output user label exits are taken, the address of the save area entry (user totaling image area) corresponding to the last record physically written on a volume is passed to you in the fourth entry of the user label parameter list. (This parameter list is described in “Open/Close/EOV Standard User Label Exit” on page 514.) When an EOV exit is taken for an output data set and user totaling has been specified, the address of the user totaling image area is in register 0.

When using user totaling for an output data set, that is, when creating the data set, you must update your control data in your totaling area before issuing a PUT or a WRITE macro. The control program places an image of your totaling area in the user totaling save area when an I/O operation is scheduled. A pointer to the save area entry (user totaling image area) corresponding to the last record physically written on the volume is passed to you in your label processing routine. Thus you can include the control total in your user labels.

When subsequently using this data set for input, you can collect the same information as you read each record and compare this total with the one previously stored in the user trailer label. If you have stored the total from the preceding volume in the user header label of the current volume, you can process each volume of a multivolume data set independently and still maintain this system of control.

When variable-length records are specified with the totaling function for user labels, special considerations are necessary. Because the control program determines if a variable-length record fits in a buffer after a PUT or a WRITE is issued, the total you have accumulated can include one more record than is really written on the volume. For variable-length spanned records, the accumulated total includes the control data from the volume-spanning record although only a segment of the record is on that volume. However, when you process such a data set, the volume-spanning record or the first record on the next volume will not be available to you until after the volume switch and user label processing are completed. Thus the totaling information in the user label cannot agree with that developed during processing of the volume.

One way you can resolve this situation is to maintain, when you are creating a data set, control data about each of the last two records and include both totals in your user labels. Then the total related to the last complete record on the volume and the volume-spanning record or the first record on the next volume would be available to your user label routines. During subsequent processing of the data set, your user label routines can determine if there is agreement between the generated information and one of the two totals previously saved.

When the totaling function for user labels is selected with DASD devices and secondary space is specified, the total accumulated can be one less than the actual written.

Part 4. Appendixes

© Copyright IBM Corp. 1987, 2002

Appendix A. Using Direct Access Labels

This appendix covers the following topics.

Topic	Page
Direct Access Storage Device Architecture	527
Volume Label Group	528
Data Set Control Block (DSCB)	530
User Label Groups	530

This chapter is intended to help you understand direct access labels.

Direct Access Storage Device Architecture

Disks reside in direct access storage subsystems. The real disks might have an architecture that differs from what the subsystem presents to the operating system. The operating system sees direct access storage devices (DASDs). This book and other z/OS documentation describe DASDs. Hardware documentation describes internal characteristics of direct access storage subsystems.

As seen by software, each disk or tape is called a volume. Each volume can contain one or more complete data sets and parts of data sets. Each complete or partial data set on a DASD volume has a data set label. Each complete or partial data set on a tape volume has a data set label only if the volume has IBM standard labels or ISO or ANSI standard labels. For information about data sets and labels on magnetic tapes, see “Magnetic Tape Volumes” on page 11.

Only standard label formats are used on direct access volumes. Volume, data set, and optional user labels are used (see Figure 114 on page 528). In the case of direct access volumes, the data set label is the data set control block (DSCB).

The system programmer or storage administrator uses ICKDSF to format tracks, write a volume label, and create a volume table of contents (VTOC). The VTOC contains all the DSCBs. RACF DASDVOL authority is required to create a VTOC. DASDVOL authority is not required to allocate space on volumes. The system controls space on SMS volumes by other means such as the ACS routines, storage group definitions and ISMF commands.

© Copyright IBM Corp. 1987, 2002

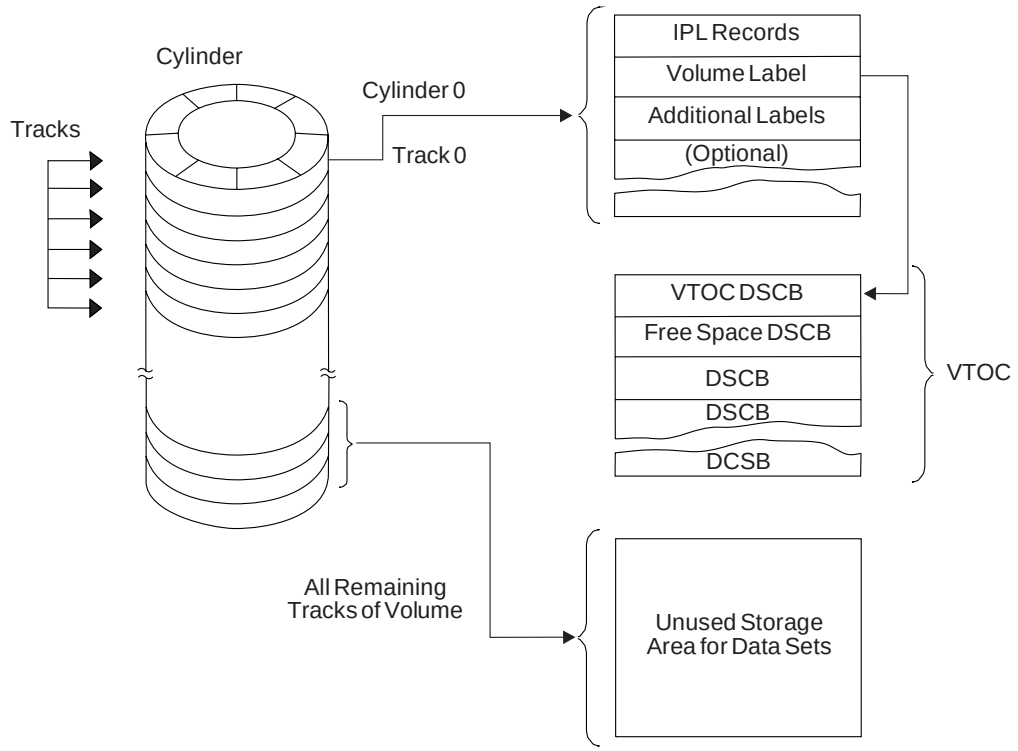


Figure 114. Direct Access Labeling

Volume Label Group

The volume label group immediately follows the first two initial program loading (IPL) records on track 0 of cylinder 0 of the volume. It consists of the initial volume label at record 3 plus a maximum of seven additional volume labels. The initial volume label identifies a volume and its owner, and is used to verify that the correct volume is mounted. It can also be used to prevent use of the volume by unauthorized programs. The additional labels can be processed by an installation routine that is incorporated into the system.

The format of the data portion of the direct access volume label group is shown in Figure 115 on page 529.

(As many as Seven Additional Volume Labels)
80-Byte Physical Record

Field		
1	(3)	Volume Label Identifier (VOL)
2	(1)	Volume Label Number (1)
3	(6)	Volume Serial Number
4	(1)	Volume Security
5	(5)	VTOC Pointer
6	(21)	Reserved (Blank)
7	(14)	Owner Identification
8	(29)	Blank

Figure 115. Initial Volume Label Format

The operating system identifies an initial volume label when, in reading the initial record, it finds that the first 4 characters of the record are VOL1. That is, they contain the volume label identifier and the volume label number. The initial volume label is 80 bytes. The format of an initial volume label are described in the following text.

Volume Label Identifier (VOL). Field 1 identifies a volume label.

Volume Label Number (1). Field 2 identifies the relative position of the volume label in a volume label group. It must be written as X'F1'.

Volume Serial Number. Field 3 contains a unique identification code assigned when the volume enters the system. You can place the code on the external surface of the disk drive for visual identification. The code is any 1 to 6 alphanumeric or national (#, $, @) characters, or a hyphen (X'60'). If this field is fewer than 6 characters, it is padded on the right with blanks.

Volume Security. Field 4 is reserved for use by installations that want to provide security for volumes. Make this field an X'C0' unless you have your own security processing routines.

VTOC Pointer. Field 5 of direct access volume label 1 contains the address of the VTOC in the form of CCHHR.

Reserved. Field 6 is reserved for possible future use, and should be left blank.

Owner Name and Address Code. Field 7 contains an optional identification of the owner of the volume.

Field 8. All the bytes in Field 8 are left blank.

Data Set Control Block (DSCB)

The system automatically constructs a DSCB when space is requested for a data set on a direct access volume. Each data set on a direct access volume has one or more DSCBs to describe its characteristics. The DSCB appears in the VTOC and, in addition to space allocation and other control information, contains operating system data, device-dependent information, and data set characteristics. There are seven kinds of DSCBs, each with a different purpose and a different format number. See *z/OS DFSMSdfp Advanced Services* for an explanation of format-1 through format-7 DSCBs. Format 0 DSCBs are used to show empty space in the VTOC.

User Label Groups

User header and trailer label groups can be included with data sets of physically sequential or direct organization. They are not supported for extended format data sets. The labels in each group have the format shown in Figure 116.

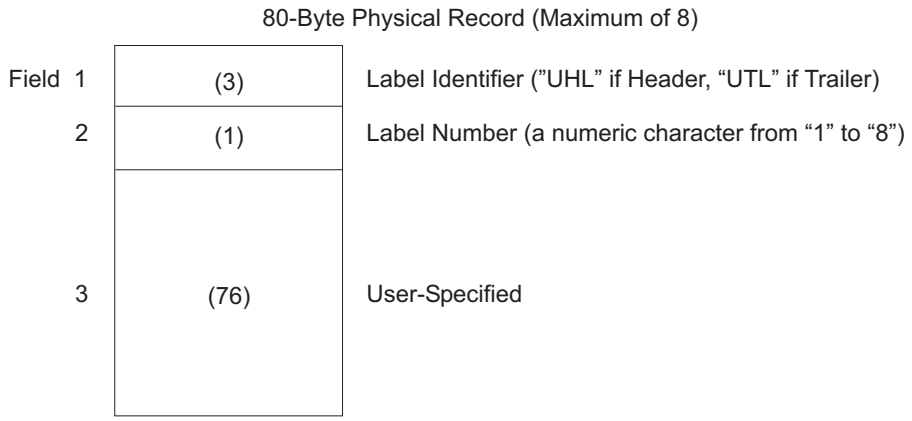


Figure 116. User Header and Trailer Labels on DASD or Tape

Each group can include as many as eight labels, but the space required for both groups must not be more than one track on a direct access storage device. A program becomes device-dependent (among direct access storage devices) when it creates more than eight header labels or eight trailer labels.

If user labels are specified in the DD statement (LABEL=SUL), an additional track is normally allocated when the data set is created. No additional track is allocated when specific tracks are requested (SPACE=(ABSTR,...)). In that case, labels are written on the first track that is allocated.

User Header Label Group. The operating system writes these labels as directed by the processing program recording the data set. The first four characters of the user header label must be UHL1,UHL2, through UHL8; you can specify the remaining 76 characters. When the data set is read, the operating system makes the user header labels available to the application program for processing.

User Trailer Label Group. These labels are recorded (and processed) as explained in the preceding text for user header labels, except that the first four characters must be UTL1,UTL2, through UTL8.

The format of user header and trailer labels follows:

Label Identifier. Field 1 shows the kind of user header label. “UHL” means a user header label; “UTL” means a user trailer label.

Label Number. Field 2 identifies the relative position (1 to 8) of the label within the user label group. It is an EBCDIC character.

User-Specified. Field 3 (76 bytes).

Appendix B. Using the Double-Byte Character Set (DBCS)

This appendix covers the following topics.

Topic	Page
DBCS Character Support	533
Record Length When Using DBCS Characters	533

Double-byte character set (DBCS) support lets you process characters in languages that contain too many characters or symbols for each to be assigned a 1-byte hexadecimal value. You can use DBCS to process languages, such as Japanese and Chinese, that use ideographic characters. In DBCS, two bytes are used to describe each character; this lets you describe more than 35 000 characters. When one byte is used to describe a character, as in EBCDIC, it is called a single-byte character set (SBCS).

DBCS Character Support

DBCS support is not used to create characters; it is used to print and copy DBCS characters already in the data set. To print and copy DBCS characters, use the access method services commands PRINT and REPRO. See *z/OS DFSMS Access Method Services* for information on using PRINT and REPRO with DBCS data.

When the data has a mixture of DBCS and SBCS strings, you must use two special delimiters, *SO* (shift out) and *SI* (shift in), which designate where a DBCS string begins and where it ends. *SO* tells you when you are leaving an SBCS string, and *SI* tells you when you are returning to an SBCS string. Use the PRINT and REPRO commands to insert the SO and SI characters around the DBCS data.

DBCS data must satisfy the following criteria:

1. The data must be bracketed by paired SO and SI characters when used in combination with SBCS data.
2. The number of bytes between the SO and SI characters must be even because each DBCS character requires two bytes to represent it.
3. Each DBCS character must reside within a range of valid character codes. The valid character codes range from X'41' through X'FE' for both the first and second byte. For example, X'41FE' is a valid DBCS character but not X'39FF'. X'4040' is a DBCS space.

Record Length When Using DBCS Characters

This section shows how to define the record length for fixed-length and variable-length records when using DBCS characters.

Fixed-Length Records

Because inserting of SO and SI characters increases the output record length, you must define the output data set with enough space in the output record. The record length of the output data set must be equal to the input data set's record length plus the additional number of bytes necessary to insert the SO and SI pairs. Each SO and SI pair consists of 2 bytes. In the following example for a fixed-length record, the input record length is 80 bytes and consists of one DBCS string surrounded by an SO and SI pair. The output record length would be 82 bytes, which is correct.

© Copyright IBM Corp. 1987, 2002

```
Input record length = 80; number of SO and SI pairs = 1
Output record length = 82 (correct length)
```

An output record length of 84 bytes, for example, would be too large and would result in an error. An output record length of 80 bytes, for example, would be too small because there would not be room for the SO and SI pair. If the output record length is too small or too large, an error message is issued, a return code of 12 is returned from IEBGENER, and the command ends.

Variable-Length Records

Because insertion of SO and SI characters increases the output record length, you must define the output data set with enough space in the output record. The input data set's record length plus the additional number of bytes necessary to insert the SO and SI pairs must not exceed the maximum record length of the output data set. Each SO and SI pair consists of 2 bytes. If the output record length is too small, an error message will be issued, a return code of 12 will be returned from IEBGENER, and the command will be ended.

In the following example for a variable-length record, the input record length is 50 bytes and consists of four DBCS string surrounded by SO and SI pairs. The output record length is 50 bytes which is too small because the SO and SI pairs add eight extra bytes to the record length. The output record length should be at least 58 bytes.

```
Input record length = 50; number of SO and SI pairs = 4
Output record length = 50 (too small; should be at least 58 bytes)
```

Appendix C. Processing Direct Data Sets

This appendix covers the following topics.

Topic	Page
Using the Basic Direct Access Method (BDAM)	535
Processing a Direct Data Set Sequentially	536
Organizing a Direct Data Set	536
Creating a Direct Data Set	537
Referring to a Record	539
Adding or Updating Records	540
Sharing DCBs	544

Use of BDAM is not recommended; use VSAM instead. If you use BDAM, be aware of the following limitations:

- Keyed blocks use hardware keys, which are less efficient than VSAM keys.
- BDAM does not support extended format data sets.
- The use of relative track addressing or actual device addresses easily can lead to program logic that is dependent on characteristics that are unique to certain device types.
- Updating R0 can be inefficient.
- Load mode does not support 31-bit addressing mode.

Using the Basic Direct Access Method (BDAM)

Create a direct data set with the basic sequential access method (BSAM). Use the MACRF=WL parameter in the BSAM DCB macro to create a direct data set.

The application program must synchronize all I/O operations with a CHECK or a WAIT macro.

The application program must block and unblock its own input and output records. (BDAM only reads and writes data blocks.)

You can find data blocks within a data set with one of the following addressing techniques.

Actual device addresses. This specifies the actual location.

Relative track address technique. This locates a track on a direct access storage device starting at the beginning of the data set.

Relative block address technique. This locates a fixed-length data block starting from the beginning of the data set.

BDAM macros can be issued in 24-bit or 31-bit addressing mode.

© Copyright IBM Corp. 1987, 2002

Processing a Direct Data Set Sequentially

Although you can process a direct data set sequentially using either the queued access method or the basic access method, you cannot read record keys using the queued access method. When you use the basic access method, each unit of data transmitted between virtual storage and an I/O device is regarded by the system as a record. If, in fact, it is a block, you must perform any blocking or deblocking required. For that reason, the LRECL field is not used when processing a direct data set. Only BLKSIZE must be specified when you read, add, or update records on a direct data set.

If dynamic buffering is specified for your direct data set, the system will provide a buffer for your records. If dynamic buffering is not specified, you must provide a buffer for the system to use.

The discussion of direct access storage devices shows that record keys are optional. If they are specified, they must be used for every record and must be of a fixed length.

Organizing a Direct Data Set

In a direct data set, there is a relationship between a control number (or identification of each record) and its location on the direct access volume. Therefore, you can access a record without an index search. You determine the actual organization of the data set. If the data set has been carefully organized, location of a particular record takes less time than with an indexed sequential data set.

You can use direct addressing to develop the organization of your data set. When you use direct addresses, the location of each record in the data set is known.

By Range of Keys

If format-F records with keys are being written, the key of each record can be used to identify the record. For example, a data set with keys ranging from 0 to 4999 should be allocated space for 5000 records. Each key relates directly to a location that you can refer to as a relative record number. Therefore, each record should be assigned a unique key.

If identical keys are used, it is possible, during periods of high processor and channel activity, to skip the desired record and retrieve the next record on the track. The main disadvantage of this type of organization is that records might not exist for many of the keys, even though space has been reserved for them.

By Number of Records

Space could be allocated based on the number of records in the data set rather than on the range of keys. Allocating space based on the number of records requires the use of a cross-reference table. When a record is written in the data set, you must note the physical location as a relative block number, an actual address, or as a relative track and record number. The addresses must then be stored in a table that is searched when a record is to be retrieved. Disadvantages are that cross-referencing can be used efficiently only with a small data set; storage is required for the table, and processing time is required for searching and updating the table.

With Indirect Addressing

A more common, but somewhat complex, technique for organizing the data set involves the use of indirect addressing. In indirect addressing, the address of each record in the data set is determined by a mathematical manipulation of the key, also called randomizing or conversion. Because several randomizing procedures could be used, no attempt is made here to describe or explain those that might be most appropriate for your data set.

Creating a Direct Data Set

After the organization of a direct data set has been determined, the process of creating it is almost identical to creating a sequential data set. The BSAM DCB macro should be used with the WRITE macro (the form used to allocate a direct data set). Issue the WRITE and CHECK macros in 24-bit mode. The following parameters must be specified in the DCB macro:

- DSORG=PS or PSU
- DEVD=DA or omitted
- MACRF=WL

The DD statement must specify direct access (DSORG=DA or DAU). If keys are used, a key length (KEYLEN) must also be specified. Record length (LRECL) need not be specified, but can provide compatibility with sequential access method processing of a direct data set.

DSORG and KEYLEN can be specified through data class. For more information about data class see “Chapter 22. Specifying and Initializing Data Control Blocks” on page 313.

Restrictions in Creating a Direct Data Set Using QSAM

It is possible to create a direct data set using QSAM (no keys allowed) or BSAM (with or without keys and the DCB specifies MACRF=W). However, it is not recommended that you access a direct data set using QSAM because you cannot request a function that requires the information in the capacity record (R0) data field. For example, the following restrictions would apply:

- Variable-length or undefined-length spanned record processing is not permitted.
- The WRITE add function with extended search for fixed-length records (with or without track overflow) is not permitted.

If a direct data set is created and updated or read within the same job step, and the OPTCD parameter is used in the creation, updating, or reading of the data set, different DCBs and DD statements should be used.

With Direct Addressing with Keys

If you are using direct addressing with keys, you can reserve space for future format-F records by writing a dummy record. To reserve or truncate a track for format-U, format-V, or format-VS records, write a capacity record.

Format-F records are written sequentially as they are presented. When a track is filled, the system automatically writes the capacity record and advances to the next track.

Rule: Direct data sets whose records are to be identified by relative track address must be limited in size to no more than 65 536 tracks for the entire data set.

With BDAM to Allocate a VIO Data Set

If a VIO data set is opened for processing with the extended search option, BDAM does not search unused tracks. The information needed to determine the data set size is written in the DSCB during the close of the DCB used in the create step. Therefore, if this data set is being allocated and processed by the same program, and the DCB used for creating the data set has not been closed before opening the DCB to be used for processing, the resultant beginning and ending CCHH will be equal.

Example: In the example problem in Figure 117, a tape containing 204-byte records arranged in key sequence is used to allocate a direct data set. A 4-byte binary key for each record ranges from 1000 to 8999, so space for 8000 records is requested.

```
//DAOUTPUT DD     DSNAME=SLATE.INDEX.WORDS,DCB=(DSORG=DA,         C
//            BLKSIZE=200,KEYLEN=4,RECFM=F),SPACE=(204,8000),---
//TAPINPUT DD     ---
           ...
DIRECT     START
           ...
           L      9,=F'1000'
           OPEN   (DALOAD,(OUTPUT),TAPEDCB)
           LA     10,COMPARE
NEXTREC    GET    TAPEDCB
           LR     2,1
COMPARE    C      9,0(2)       Compare key of input against
*                                           control number
           BNE    DUMMY
           WRITE  DECB1,SF,DALOAD,(2)         Write data record
           CHECK  DECB1
           AH     9,=H'1'
           B      NEXTREC
DUMMY      C      9,=F'8999'    Have 8000 records been written?
           BH     ENDJOB
           WRITE  DECB2,SD,DALOAD,DUMAREA     Write dummy
           CHECK  DECB2
           AH     9,=H'1'
           BR     10
INPUTEND   LA     10,DUMMY
           BR     10
ENDJOB     CLOSE  (TAPEDCB,,DALOAD)
           ...
DUMAREA    DS     8F
DALOAD     DCB    DSORG=PS,MACRF=(WL),DDNAME=DAOUTPUT,          C
                  DEVD=DA,SYNAD=CHECKER,---
TAPEDCB    DCB    EODAD=INPUTEND,MACRF=(GL), ---
           ...
```

Figure 117. Creating a Direct Data Set (Tape-to-Disk)

Referring to a Record

You choose among three types of record addressing and you can choose other addressing options.

Record Addressing

After you have determined how your data set is to be organized, you must consider how the individual records will be referred to when the data set is updated or new records are added. You refer to records using one of three forms of addressing:

- **Relative Block Address.** You specify the relative location of the record (block) within the data set as a 3-byte binary number. You can use this type of reference only with format-F records. The system computes the actual track and record number. The relative block address of the first block is 0.
- **Relative Track Address.** You specify the relative track as a 2-byte binary number and the actual record number on that track as a 1-byte binary number. The relative track address of the first track is 0. The number of the first record on each track is 1.

  Direct data sets whose records are to be identified by relative track address must be limited in size to no more than 65 536 tracks for the entire data set.
- **Actual Address.** You supply the actual address in the standard 8-byte form, MBBCCHHR. Remember that using an actual address might force you to specify that the data set is unmovable. In that case the data set is ineligible to be system managed.

In addition to the relative track or block address, you specify the address of a virtual storage location containing the record key. The system computes the actual track address and searches for the record with the correct key.

Extended Search

You request that the system begin its search with a specified starting location and continue for a certain number of records or tracks. You can use the extended search option to request a search for unused space where a record can be added.

To use the extended search option, you must specify in the DCB (DCBLIMCT) the number of tracks (including the starting track) or records (including the starting record) that are to be searched. If you specify a number of records, the system might actually examine more than this number. In searching a track, the system searches the entire track (starting with the first record); it therefore might examine records that precede the starting record or follow the ending record.

If the DCB specifies a number equal to or greater than the number of tracks allocated to the data set or the number of records within the data set, the entire data set is searched in the attempt to satisfy your request.

In addition to the relative track or block address, you specify the address of a virtual storage location containing the record key. The system computes the actual track address and searches for the record with the correct key.

Exclusive Control for Updating

When more than one task is referring to the same data set, exclusive control of the block being updated is required to prevent referring to the same record at the same time. Rather than issuing an ENQ macro each time you update a block, you can request exclusive control through the MACRF field of the DCB and the *type* parameter of the READ macro. The coding example in Figure 119 on page 543 shows the use of exclusive control. After the READ macro is executed, your task has exclusive control of the block being updated. No other task in the system requesting access to the block is given access until the operation started by your WRITE macro is complete. If, however, the block is not to be written, you can release exclusive control using the RELEX macro.

Feedback Option

The feedback option specifies that the system is to provide the address of the record requested by a READ or WRITE macro. This address can be in the same form that was presented to the system in the READ or WRITE macro, or as an 8-byte actual address. You can specify the feedback option in the OPTCD parameter of the DCB and in the READ or WRITE macro. If the feedback option is omitted from the DCB, but is requested in a READ or WRITE macro, an 8-byte actual address is returned to you.

The feedback option is automatically provided for a READ macro requesting exclusive control for updating. This feedback will be in the form of an actual address (MBBCCHHR) unless feedback was specified in the OPTCD field of the DCB. In that case, feedback is returned in the format of the addressing scheme used in the application program (an actual or a relative address). When a WRITE or RELEX macro is issued (which releases the exclusive control for the READ request), the system will assume that the addressing scheme used for the WRITE or RELEX macro is in the same format as the addressing scheme used for feedback in the READ macro.

Adding or Updating Records

The techniques for adding records to a direct data set depend on the format of the records and the organization used.

Format-F with Keys

Essentially, adding a record amounts to updating by record identification. You can refer to the record using either a relative block address or a relative track address.

If you want to add a record passing a relative block address, the system converts the address to an actual track address. That track is searched for a dummy record. If a dummy record is found, the new record is written in place of it. If there is no dummy record on the track, you are informed that the write operation did not take place. If you request the extended search option, the new record will be written in place of the first dummy record found within the search limits you specify. If none is found, you are notified that the write operation could not take place.

In the same way, a reference by relative track address causes the record to be written in place of a dummy record on the referenced track or the first within the search limits, if requested. If extended search is used, the search begins with the first record on the track. Without extended search, the search can start at any record on the track. Therefore, records that were added to a track are not necessarily located on the track in the same sequence they were written in.

Format-F without Keys

Here too, adding a record is really updating a dummy record already in the data set. The main difference is that dummy records cannot be written automatically when the data set is allocated. You will have to use your own method for flagging dummy records. The update form of the WRITE macro (MACRF=W) must be used rather than the add form (MACRF=WA).

You will have to retrieve the record first (using a READ macro), test for a dummy record, update, and write.

Format-V or Format-U with Keys

The technique used to add format-V and -U records with keys depends on whether records are located by indirect addressing or by a cross-reference table. If indirect addressing is used, you must at least initialize each track (write a capacity record) even if no data is actually written. That way the capacity record shows how much space is available on the track. If a cross-reference table is used, you should enter all the actual input and then initialize enough succeeding tracks to contain any additions that might be required.

To add a new record, use a relative track address. The system examines the capacity record to see if there is room on the track. If there is, the new record is written. Under the extended search option, the record is written in the first available area within the search limit.

Format-V or Format-U without Keys

Because a record of this type does not have a key, you can access the record only by its relative track or actual address (direct addressing only). When you add a record to this data set, you must retain the relative track or actual address data (for example, by updating your cross-reference table). The extended search option is not permitted because it requires keys.

Tape-to-Disk Add—Direct Data Set

The example in Figure 118 involves adding records to the data set allocated in Figure 117 on page 538.

```
//DIRADD  DD       DSNAME=SLATE.INDEX.WORDS,---
//TAPEDD  DD       ---
          ...
DIRECTAD  START
          ...
          OPEN     (DIRECT,(OUTPUT),TAPEIN)
NEXTREC   GET      TAPEIN,KEY
          L        4,KEY            Set up relative record number
          SH       4,=H'1000'
          ST       4,REF
          WRITE    DECB,DA,DIRECT,DATA,'S',KEY,REF+1
          WAIT     ECB=DECB
          CLC      DECB+1(2),=X'0000'   Check for any errors
          BE       NEXTREC
```

Check error bits and take required action

```
DIRECT    DCB      DDNAME=DIRADD,DSORG=DA,RECFM=F,KEYLEN=4,BLKSIZE=200,   C
              MACRF=(WA)
TAPEIN    DCB      ---
KEY       DS       F
DATA      DS       CL200
REF       DS       F
          ...
```

Figure 118. Adding Records to a Direct Data Set

The write operation adds the key and the data record to the data set. If the existing record is not a dummy record, an indication is returned in the exception code of the DECB. For that reason, it is better to use the WAIT macro instead of the CHECK macro to test for errors or exceptional conditions.

Tape-to-Disk Update—Direct Data Set

The example in Figure 119 is similar to that in Figure 118, but involves updating a record rather than adding one.

```
//DIRECTDD DD      DSNAME=SLATE.INDEX.WORDS,---
//TAPINPUT DD      ---
           ...
DIRUPDAT   START
           ...
           OPEN    (DIRECT,(UPDAT),TAPEDCB)
NEXTREC    GET     TAPEDCB,KEY
           PACK    KEY,KEY
           CVB     3,KEYFIELD
           SH      3,=H'1'
           ST      3,REF
           READ    DECBRD,DIX,DIRECT,'S','S',0,REF+1
           CHECK   DECBRD
           L       3,DECBRD+12
           MVC     0(30,3),DATA
           ST      3,DECBWR+12
           WRITE   DECBWR,DIX,DIRECT,'S','S',0,REF+1
           CHECK   DECBWR
           B       NEXTREC
           ...
KEYFIELD   DS      0D
           DC      XL3'0'
KEY        DS      CL5
DATA       DS      CL30
REF        DS      F
DIRECT     DCB     DSORG=DA,DDNAME=DIRECTDD,MACRF=(RISXC,WIC),       C
               OPTCD=RF,BUFNO=1,BUFL=100
TAPEDCB    DCB     ---
           ...
```

Figure 119. Updating a Direct Data Set

There is no check for dummy records. The existing direct data set contains 25 000 records whose 5-byte keys range from 00 001 to 25 000. Each data record is 100 bytes long. The first 30 characters are to be updated. Each input tape record consists of a 5-byte key and a 30-byte data area. Notice that only data is brought into virtual storage for updating.

When you are updating variable-length records, you should use the same length to read and write a record.

With User Labels

If you use user labels, they must be created when the data set is allocated. They can be updated, but not added or deleted, during processing of a direct data set. When creating a multivolume direct data set using BSAM, you should turn off the header exit entry after OPEN and turn on the trailer label exit entry just before issuing the CLOSE. Turning off the header exit entry and turning on the trailer label exit entry eliminate the end-of-volume exits. The first volume, containing the user label track, must be mounted when the data set is closed. If you have requested exclusive control, OPEN and CLOSE issues ENQ and DEQ, preventing simultaneous reference to user labels.

Sharing DCBs

BDAM permits several tasks to share the same DCB and several jobs to share the same data set. It synchronizes I/O requests at both levels by maintaining a read-exclusive list.

When several tasks share the same DCB and each asks for exclusive control of the same block, BDAM issues a system ENQ for the block (or in some cases the entire track). It reads in the block and passes it to the first caller while putting all subsequent requests for that block on a wait queue. When the first task releases the block, BDAM moves it into the next caller's buffer and posts that task complete. The block is passed to subsequent callers in the order the request was received.

BDAM not only synchronizes the I/O requests, but also issues only one ENQ and one I/O request for several read requests for the same block.

Because BDAM processing is not sequential and I/O requests are not related, a caller can continue processing other blocks while waiting for exclusive control of the shared block.

Because BDAM issues a system ENQ for each record held exclusively, it permits a data set to be shared between jobs, so long as all callers use BDAM. The system enqueues on BDAM's commonly understood argument.

BDAM supports multiple task users of a single DCB when working with existing data sets. When operating in load mode, however, only one task can use the DCB at a time. The following restrictions and comments apply when more than one task shares the same DCB, or when multiple DCBs are used for the same data set.

- Subpool 0 must be shared.
- You should ensure that a WAIT or CHECK macro has been issued for all outstanding BDAM requests before the task issuing the READ or WRITE macro ends. In case of abnormal termination, this can be done through a STAE/STAI or ESTAE exit.
- FREEDBUF or RELEX macros should be issued to free any resources that could still be held by the terminating task. You can free the resources during or after task termination.

Rule: OPEN, CLOSE, and all I/O must be performed in the same key and state (problem state or supervisor state).

Appendix D. Processing Indexed Sequential Data Sets

This appendix covers the following topics.

Topic	Page
Using the Basic Indexed Sequential Access Method (BISAM)	545
Using the Queued Indexed Sequential Access Method (QISAM)	545
Processing	546
Data Set Organization	546
Creating an ISAM Data Set	550
Allocating Space	553
Calculating Space Requirements	556
Retrieving and Updating	561
Adding Records	566
Maintaining an Indexed Sequential Data Set	569

Recommendation: Do not use BISAM or QISAM to process indexed sequential data sets because it is less efficient and less reliable than VSAM and ISAM and requires 24-bit addressing. Use VSAM data sets instead. You can use BISAM or QISAM to process indexed sequential data sets. Indexed sequential data sets cannot be system managed.

To ease the task of converting programs from ISAM to VSAM, consider using the ISAM interface for VSAM. See “Appendix E. Using ISAM Programs with VSAM Data Sets” on page 579.

Note: The presence of the ISAM system code in your system is optional. The system programmer might have omitted the function from your operating system.

Using the Basic Indexed Sequential Access Method (BISAM)

BISAM cannot be used to create an indexed sequential data set.

BISAM directly retrieves logical records by key, updates blocks of records in-place, and inserts new records in their correct key sequence.

Your program must synchronize all I/O operations with a CHECK or a WAIT macro.

Other DCB parameters are available to reduce I/O operations by defining work areas that contain the highest level master index and the records being processed.

Using the Queued Indexed Sequential Access Method (QISAM)

The characteristics of an indexed sequential data set are established when the data set is created using QISAM. You cannot change them without reorganizing the data set. The DCB parameters that establish these characteristics are: BLKSIZE, CYLOFL, KEYLEN, LRECL, NTM, OPTCD, RECFM, and RKP.

A data set processed with QISAM can have unblocked fixed-length records (F), blocked fixed-length records (FB), unblocked variable-length records (V), or blocked variable-length records (VB).

© Copyright IBM Corp. 1987, 2002

Processing Indexed Sequential Data Sets

QISAM can create an indexed sequential data set (QISAM, load mode), add additional data records at the end of the existing data set (QISAM, resume load mode), update a record in place, or retrieve records sequentially (QISAM, scan mode).

For an indexed sequential data set, you can allocate space on the same or separate volumes for the data set's prime area, overflow area, and cylinder/master index or indexes. For more information about space allocation, see *z/OS MVS JCL User's Guide*.

QISAM automatically generates a track index for each cylinder in the data set and one cylinder index for the entire data set. Specify the DCB parameters NTM and OPTCD to show that the data set requires a master index. QISAM creates and maintains as many as three levels of master indexes.

You can purge records by specifying the OPTCD=L DCB option when you allocate an indexed sequential data set. The OPTCD=L option flags the records you want to purge with a X'FF' in the first data byte of a fixed-length record or the fifth byte of a variable-length record. QISAM ignores these flagged records during sequential retrieval.

You can get reorganization statistics by specifying the OPTCD=R DCB option when an indexed sequential data set is allocated. The application program uses these statistics to determine the status of the data set's overflow areas.

When you allocate an indexed sequential data set, you must write the records in ascending key order.

Processing

The queued access method must be used to allocate an indexed sequential data set. It can also be used to sequentially process or update the data set and to add records to the end of the data set. The basic access method can be used to insert new records between records already in the data set and to update the data set directly.

Because indexed sequential data sets cannot take advantage of system-managed storage, you should consider converting indexed sequential data sets to VSAM data sets. You can use access method services to allocate a VSAM data set and copy the indexed sequential data set into it. For information about converting to VSAM data sets see "Appendix E. Using ISAM Programs with VSAM Data Sets" on page 579.

Data Set Organization

The organization of an indexed sequential data set allows you much flexibility in the operations you can perform. The data set can be read or written sequentially, individual records can be processed in any order, records can be deleted, and new records can be added. The system automatically locates the proper position in the data set for new records and makes any necessary adjustments when records are deleted.

The records in an indexed sequential data set are arranged according to collating sequence by a key field in each record. Each block of records is preceded by a key field that corresponds to the key of the last record in the block.

Processing Indexed Sequential Data Sets

An indexed sequential data set resides on direct access storage devices and can occupy as many as three different areas:

- The prime area, also called the prime data area, contains data records and related track indexes. It exists for all indexed sequential data sets.
- The index area contains master and cylinder indexes associated with the data set. It exists for a data set that has a prime area occupying more than one cylinder.
- The overflow area contains records that overflow from the prime area when new data records are added. It is optional.

The track indexes of an indexed sequential data set are similar to the card catalog in a library. For example, if you know the name of the book or the author, you can look in the card catalog and obtain a catalog number that enables you to locate the book in the book files. You then go to the shelves and go through rows until you find the shelf containing the book. Then you look at the individual book numbers on that shelf until you find the particular book.

ISAM uses the track indexes in much the same way to locate records in an indexed sequential data set.

As the records are written in the prime area of the data set, the system accounts for the records contained on each track in a track index area. Each entry in the track index identifies the key of the last record on each track. There is a track index for each cylinder in the data set. If more than one cylinder is used, the system develops a higher-level index called a cylinder index. Each entry in the cylinder index identifies the key of the last record in the cylinder. To increase the speed of searching the cylinder index, you can request that a master index be developed for a specified number of cylinders, as shown in Figure 120 on page 548.

Rather than reorganize the entire data set when records are added, you can request that space be allocated for additional records in an overflow area.

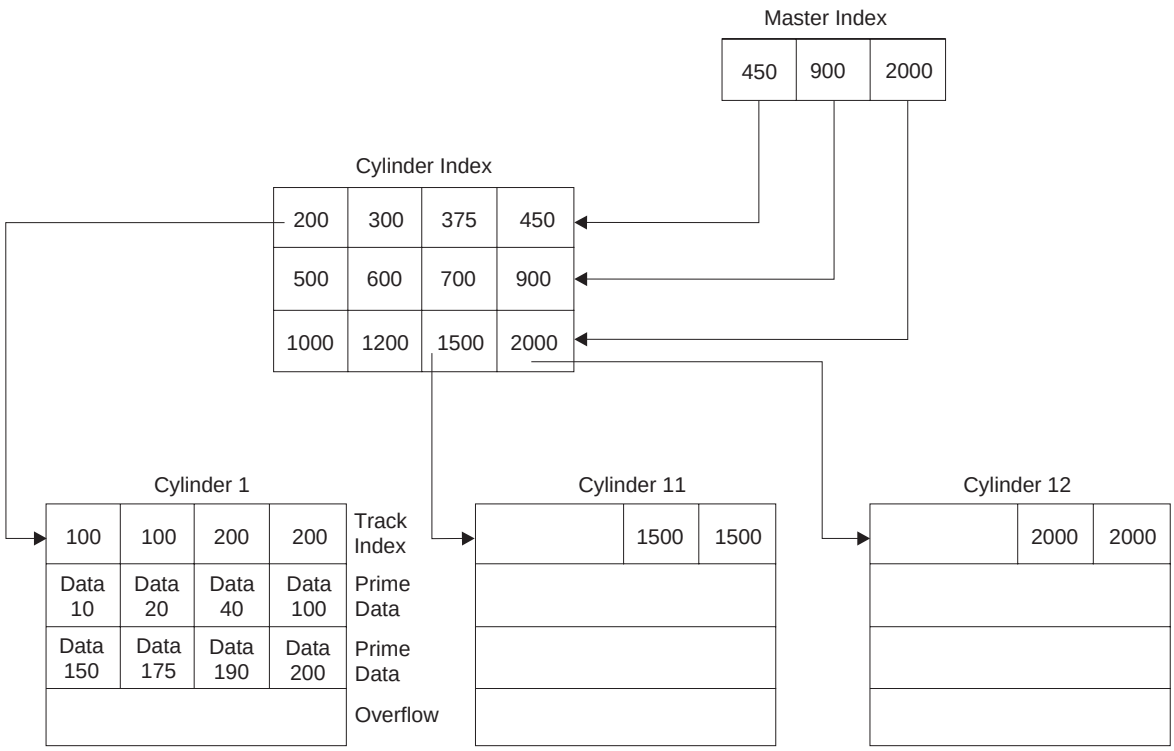


Figure 120. Indexed Sequential Data Set Organization

Prime Area

Records are written in the prime area when the data set is allocated or updated. The last track of prime data is reserved for an end-of-file mark. The portion of Figure 120 labeled cylinder 1 illustrates the initial structure of the prime area. Although the prime area can extend across several noncontiguous areas of the volume, all the records are written in key sequence. Each record must contain a key; the system automatically writes the key of the highest record before each block.

When the ABSTR option of the SPACE parameter of the DD statement is used to generate a multivolume prime area, the VTOC of the second volume, and of all succeeding volumes, must be contained within cylinder 0 of the volume.

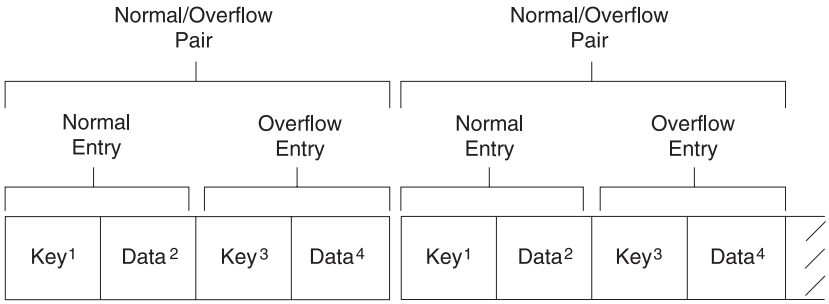
Index Areas

The operating system generates track and cylinder indexes automatically. As many as three levels of master index are created if requested.

Track Index

The track index is the lowest level of index and is always present. There is one track index for each cylinder in the prime area; it is written on the first tracks of the cylinder that it indexes.

The index consists of a series of paired entries, that is, a normal entry and an overflow entry for each prime track. Figure 121 on page 549 shows the format of a track index.



[1] **Normal key** = key of the highest record on the prime data track
[2] **Normal data** = address of the prime data track
[3] **Overflow key** = key of the highest overflow record logically associated with the prime data track
[4] **Overflow data** = address of the lowest overflow record logically associated with the prime data track

Notes:

- If there are no overflow records, overflow key and data entries are the same as normal key and data entries.
- This figure is a logical representation only; that is, it makes no attempt to show the physical size of track index entries.

Figure 121. Format of Track Index Entries

For fixed-length records, each normal entry points to record 0 or to the first data record on a track shared by index and data records. (DCBFIRSH also points to it.) For variable-length records, the normal entry contains the key of the highest record on the track and the address of the last record.

The overflow entry is originally the same as the normal entry. (This is why 100 appears twice on the track index for cylinder 1 in Figure 120.) The overflow entry is changed when records are added to the data set. Then the overflow entry contains the key of the highest overflow record and the address of the lowest overflow record logically associated with the track.

If all the tracks allocated for the prime data area are not used, the index entries for the unused tracks are flagged as inactive. The last entry of each track index is a dummy entry indicating the end of the index. When fixed-length record format has been specified, the remainder of the last track of each cylinder used for a track index contains prime data records, if there is room for them.

Each index entry has the same format as the others. It is an unblocked, fixed-length record consisting of a count, a key, and a data area. The length of the key corresponds to the length of the key area in the record to which it points. The data area is always 10 bytes long. It contains the full address of the track or record to which the index points, the level of the index, and the entry type.

Cylinder Index

For every track index created, the system generates a cylinder index entry. There is one cylinder index for a data set that points to a track index. Because there is one track index per cylinder, there is one cylinder index entry for each cylinder in the prime data area, except for a 1-cylinder prime area. As with track indexes, inactive entries are created for any unused cylinders in the prime data area.

Master Index

As an optional feature, the operating system creates a master index at your request. The presence of this index makes long, serial searches through a large cylinder index unnecessary.

You can specify the conditions under which you want a master index created. For example, if you have specified NTM=3 and OPTCD=M in your DCB macro, a master index is created when the cylinder index exceeds 3 tracks. The master index consists of one entry for each track of cylinder index. If your data set is extremely large, a higher-level master index is created when the first-level master index exceeds three tracks. This higher-level master index consists of one entry for each track of the first-level master index. This procedure can be repeated for as many as three levels of master index.

Overflow Areas

As records are added to an indexed sequential data set, space is required to contain those records that will not fit on the prime data track on which they belong. You can request that a number of tracks be set aside as a cylinder overflow area to contain overflows from prime tracks in each cylinder. An advantage of using cylinder overflow areas is a reduction of search time required to locate overflow records. A disadvantage is that there will be unused space if the additions are unevenly distributed throughout the data set.

Instead of, or in addition to, cylinder overflow areas, you can request an independent overflow area. Overflow from anywhere in the prime data area is placed in a specified number of cylinders reserved solely for overflow records. An advantage of having an independent overflow area is a reduction in unused space reserved for overflow. A disadvantage is the increased search time required to locate overflow records in an independent area.

If you request both cylinder overflow and independent overflow, the cylinder overflow area is used first. It is a good practice to request cylinder overflow areas large enough to contain a reasonable number of additional records, and an independent overflow area to be used as the cylinder overflow areas are filled.

Creating an ISAM Data Set

You can allocate an indexed sequential data set either by writing all records in a single step, or by writing one group of records in one step and writing additional groups of records in subsequent steps. Writing records in subsequent steps is called resume loading.

One-Step Method

To create an indexed sequential data set by the one-step method, take the following actions:

1. Code DSORG=IS or DSORG=ISU and MACRF=PM or MACRF=PL in the DCB macro.
2. Specify the following attributes in the DD statement:
   - DCB attributes DSORG=IS or DSORG=ISU
   - Record length (LRECL)
   - Block size (BLKSIZE)
   - Record format (RECFM)
   - Key length (KEYLEN)
   - Relative key position (RKP)

- Options required (OPTCD)
- Cylinder overflow (CYLOFL)
- Number of tracks for a master index (NTM)
- Space requirements (SPACE)
- To reuse previously allocated space, omit the SPACE parameter and code DISP=(OLD, KEEP)

3. Open the data set for output.
4. Use the PUT macro to place all the records or blocks on the direct access volume.
5. Close the data set.

The records that comprise a newly created data set must be presented for writing in ascending order by key. You can merge two or more input data sets. If you want a data set with no records (a null data set), you must write at least one record when you allocate the data set. You can subsequently delete this record to achieve the null data set.

Recommendations:

- If you unload a data set so that it deletes all existing records in an ISAM data set, at least one record must be written on the subsequent load. If no record is written, the data set will be unusable.
- If the records are blocked, do not write a record with a hexadecimal value of FF and a key of hexadecimal value FF. This value of FF is used for padding. If it occurs as the last record of a block, the record cannot be retrieved. If the record is moved to the overflow area, the record is lost.
- After an indexed sequential data set has been allocated, you cannot change its *cms* characteristics. However, for added flexibility, the system lets you retrieve records by using either the queued access technique with simple buffering or the basic access method with dynamic buffering.

Full-Track-Index Write Option

When creating an indexed sequential data set, you can improve performance by using the full-track-index write option. You do this by specifying OPTCD=U in the DCB. OPTCD=U causes the operating system to accumulate track index entries in virtual storage. The full-track-index write option can be used only for fixed-length records.

If you do not specify full-track-index write, the operating system writes each normal overflow pair of entries for the track index after the associated prime data track has been written. If you do specify full-track-index write, the operating system accumulates track index entries in virtual storage until either (a) there are enough entries to fill a track or (b) end-of-data or end-of-cylinder is reached. Then the operating system writes these entries as a group, writing one group for each track of track index. The OPTCD=U option requires allocation of more storage space (the space in which the track index entries are gathered), but the number of I/O operations required to write the index can be significantly decreased.

When you specify the full-track-index write option, the track index entries are written as fixed-length unblocked records. If the area of virtual storage available is not large enough the entries are written as they are created, that is, in normal overflow pairs.

Example: The example in Figure 122 on page 552 shows the creation of an indexed sequential data set from an input tape containing 60-character records.

```
//INDEXDD DD     DSNAME=SLATE.DICT(PRIME),DCB=(BLKSIZE=240,CYLOFL=1,    C
//              DSORG=IS,OPTCD=MYLR,RECFM=FB,LRECL=60,NTM=6,RKP=19,    C
//              KEYLEN=10),UNIT=3380,SPACE=(CYL,25,,CONTIG),---
//INPUTDD DD     ---
          ...
ISLOAD    START  0
          ...
          DCBD   DSORG=IS
ISLOAD    CSECT
          OPEN   (IPDATA,,ISDATA,(OUTPUT))
NEXTREC   GET    IPDATA                 Locate mode
          LR     0,1                    Address of record in register 1
          PUT    ISDATA,(0)             Move mode
          B      NEXTREC
          ...
CHECKERR  L      3,=A(ISDATA)           Initialize base for errors
          USING  IHADCB,3
          TM     DCBEXCD1,X'04'
          BO     OPERR                  Uncorrectable error
          TM     DCBEXCD1,X'20'
          BO     NOSPACE                Space not found
          TM     DCBEXCD2,X'80'
          BO     SEQCHK                 Record out of sequence
```

Rest of error checking

Error routine

End-of-job routine (EODAD FOR IPDATA)

```
IPDATA    DCB    ---
ISDATA    DCB    DDNAME=INDEXDD,DSORG=IS,MACRF=(PM),SYNAD=CHECKERR
          ...
```

Figure 122. Creating an Indexed Sequential Data Set

The key by which the data set is organized is in positions 20 through 29. The output records will be an exact image of the input, except that the records will be blocked. One track per cylinder is to be reserved for cylinder overflow. Master indexes are to be built when the cylinder index exceeds 6 tracks. Reorganization information about the status of the cylinder overflow areas is to be maintained by the system. The delete option will be used during any future updating.

Multiple-Step Method

To create an indexed sequential data set in more than one step, create the first group of records using the procedure in “one-step method”. This first group of records must contain at least one data record. The remaining records can then be added to the end of the data set in subsequent steps, using resume load. Each group to be added must contain records with successively higher keys. This method lets you allocate the indexed sequential data set in several short time periods rather than in a single long one.

This method also lets you provide limited recovery from uncorrectable output errors. When an uncorrectable output error is detected, do not attempt to continue processing or to close the data set. If you have provided a SYNAD routine, it should issue the ABEND macro to end processing. If no SYNAD routine is provided, the control program will end your processing. If the error shows that space in which to add the record was not found, you must close the data set; issuing subsequent PUT macros can cause unpredictable results. You should begin recovery at the

record following the end of the data as of the last successful close. The rerun time is limited to that necessary to add the new records, rather than to that necessary to re-create the entire data set.

Resume Load

When you extend an indexed sequential data set with resume load, the disposition parameter of the DD statement must specify MOD. To ensure that the necessary control information is in the DSCB before attempting to add records, you should at least open and close the data set successfully on a system that includes resume load. This is necessary only if the data set was allocated on a previous version of the system. Records can be added to the data set by resume load until the space allocated for prime data in the first step has been filled.

During resume load on a data set with a partially filled track or a partially filled cylinder, the track index entry or the cylinder index entry is overlaid when the track or cylinder is filled. Resume load for variable-length records begins at the next sequential track of the prime data set. If resume load abnormally ends after these index entries have been overlaid, a subsequent resume load will result in a sequence check when it adds a key that is higher than the highest key at the last successful CLOSE but lower than the key in the overlaid index entry. When the SYNAD exit is taken for a sequence check, register 0 contains the address of the high key of the data set. However, if the SYNAD exit is taken during CLOSE, register 0 will contain the IOB address.

Allocating Space

An indexed sequential data set has three areas: prime, index, and overflow. Space for these areas can be subdivided and allocated as follows:

- *Prime area*—If you request only a prime area, the system automatically uses a portion of that space for indexes, taking one cylinder at a time as needed. Any unused space in the last cylinder used for index will be allocated as an independent overflow area. More than one volume can be used in most cases, but all volumes must be for devices of the same device type.
- *Index area*—You can request that a separate area be allocated to contain your cylinder and master indexes. The index area must be contained within one volume, but this volume can be on a device of a different type than the one that contains the prime area volume. If a separate index area is requested, you cannot catalog the data set with a DD statement.

  If the total space occupied by the prime area and index area does not exceed one volume, you can request that the separate index area be imbedded in the prime area (to reduce access arm movement) by indicating an index size in the SPACE parameter of the DD statement defining the prime area.

  If you request space for prime and index areas only, the system automatically uses any space remaining on the last cylinder used for master and cylinder indexes for overflow, provided the index area is on a device of the same type as the prime area.
- *Overflow area*—Although you can request an independent overflow area, it must be contained within one volume and must be of the same device type as the prime area. If no specific request for index area is made, then it will be allocated from the specified independent overflow area.

  To request that a designated number of tracks on each cylinder be used for cylinder overflow records, you must use the CYLOFL parameter of the DCB

macro. The number of tracks that you can use on each cylinder equals the total number of tracks on the cylinder minus the number of tracks needed for track index and for prime data. That is:

```
Overflow tracks = total tracks
                - (track index tracks + prime data tracks)
```

When you allocate a 1-cylinder data set, ISAM reserves 1 track on the cylinder for the end-of-file mark. You cannot request an independent index for an indexed sequential data set that has only 1 cylinder of prime data.

When you request space for an indexed sequential data set, the DD statement must follow several rules, as shown below and summarized in Figure 123 on page 555.

- Space can be requested only in cylinders, SPACE=(CYL,(...)), or absolute tracks, SPACE=(ABSTR,(...)). If the absolute track technique is used, the designated tracks must make up a whole number of cylinders.
- Data set organization (DSORG) must be specified as indexed sequential (IS or ISU) in both the DCB macro and the DCB parameter of the DD statement.
- All required volumes must be mounted when the data set is opened; that is, volume mounting cannot be deferred.
- If your prime area extends beyond one volume, you must specify the number of units and volumes to be spanned; for example, UNIT=(3380,3),VOLUME=(,,,3).
- You can catalog the data set using the DD statement parameter DISP=(,CATLG) only if the entire data set is defined by one DD statement; that is, if you did not request a separate index or independent overflow area.

As your data set is allocated, the operating system builds the track indexes in the prime data area. Unless you request a separate index area or an imbedded index area, the cylinder and master indexes are built in the independent overflow area. If you did not request an independent overflow area, the cylinder and master indexes are built in the prime area.

If an error is found during creation of a multivolume data set, the IEHPROGM utility program should be used to scratch the DSCBs on the volumes where the data set was successfully allocated. You can use the IEHLIST utility program to determine whether part of the data set has been allocated. The IEHLIST utility program also determines whether space is available or whether identically named data sets exist before space allocation is attempted for indexed sequential data sets. These utility programs are described in *z/OS DFSMSdfp Utilities*. Figure 123 on page 555 lists the criteria for requesting indexed sequential data sets.

Criteria			Restrictions on Unit Types and Number of Units Requested	Resulting Arrangement of Areas
1. Number of DD Statements	2. Types of DD Statements	3. Index Size Coded?		
3	INDEX PRIME OVFLOW	--	None	Separate index, prime, and overflow areas.
2	INDEX PRIME	--	None	Separate index and prime areas. Any partially used index cylinder is used for independent overflow if the index and prime areas are on the same type of device.
2	PRIME OVFLOW	No	None	Prime area and overflow area with an index at its end.
2	PRIME OVFLOW	Yes	The statement defining the prime area cannot request more than one unit.	Prime area and imbedded index, and overflow area.
1	PRIME	No	None	Prime area with index at its end. Any partially used index cylinder is used for independent overflow.
1	PRIME	Yes	Statement cannot request more than one unit.	Prime area with imbedded index area; independent overflow in remainder of partially used index cylinder.

Figure 123. Requests for Indexed Sequential Data Sets

Prime Data Area

To request that the system allocate space and subdivide it as required, you should code your data definition as follows:

```
//ddname DD DSNAME=dsname,DCB=DSORG=IS,
//          SPACE=(CYL,quantity,,CONTIG),UNIT=unitname,
//          DISP=(,KEEP),---
```

You can accomplish the same type of allocation by qualifying your dsname with the element indication (PRIME). The PRIME element is assumed if it is omitted. It is required only if you request an independent index or an overflow area. To request an imbedded index area when an independent overflow area is specified, you must specify DSNAME=dsname(PRIME). To indicate the size of the imbedded index, you specify SPACE=(CYL,(quantity,,index size)).

Specifying a Separate Index Area

To request a separate index area, other than an imbedded area as described above, you must use a separate DD statement. The element name is specified as (INDEX). The space and unit designations are as required. Notice that only the first DD statement can have a data definition name. The data set name (dsname) must be the same.

```
//ddname DD DSNAME=dsname(INDEX),---
//       DD DSNAME=dsname(PRIME),---
```

Specifying an Independent Overflow Area

A request for an independent overflow area is essentially the same as for a separate index area. Only the element name, OVFLOW, is changed. If you do not request a separate index area, only two DD statements are required.

```
//ddname DD DSNAME=dsname(INDEX),---
//       DD DSNAME=dsname(PRIME),---
//       DD DSNAME=dsname(OVFLOW),---
```

Calculating Space Requirements

To determine the number of cylinders required for an indexed sequential data set, you must consider the number of blocks that will fit on a cylinder, the number of blocks that will be processed, and the amount of space required for indexes and overflow areas. When you make the computations, consider how much additional space is required for device overhead. The IBM publications for storage devices contain device-specific information on device capacities and overhead formulas. Refer to the publication written for your device. In the formulas that follow, the length of the last (or only) block must include device overhead.

```
Blocks =  Track capacity / Length of blocks
```

Use modulo-32 arithmetic when calculating key length and data length terms in your equations. Compute these terms first, then round up to the nearest increment of 32 bytes before completing the equation.

The following eight steps summarize calculation of space requirements for an indexed sequential data set.

Step 1. Number of Tracks Required

After you know how many records will fit on a track and the maximum number of records you expect to create, you can determine how many tracks you will need for your data.

```
Number of tracks required = (Maximum number of blocks / Blocks per track) + 1
```

The ISAM load mode reserves the last prime data track for the file mark.

Example: Assume that a 200,000 record parts-of-speech dictionary is stored on an IBM 3380 Disk Storage as an indexed sequential data set. Each record in the dictionary has a 12-byte key that contains the word itself and an 8-byte data area that contains a parts-of-speech code and control information. Each block contains 50 records; LRECL=20 and BLKSIZE=1000. Using the following formula, you can calculate that each track can contain 26 blocks, or 1300 records, and a total of 155 tracks is required for the dictionary.

```
Blocks = 47968/(256+((12+267)/32)(32)+((1000+267)/32)(32))
       = 47968/1824 = 26

Records per track = (26 blocks)(50 records per block) = 1300

Prime data tracks required = (200000 records / 1300 records per track) + 1 = 155
```

Step 2. Overflow Tracks Required

You will want to anticipate the number of tracks required for cylinder overflow areas. The computation is the same as for prime data tracks, but you must remember that overflow records are unblocked and a 10-byte link field is added. Remember also that, if you exceed the space allocated for any cylinder overflow area, an independent overflow area is required. Those records are not placed in another cylinder overflow area.

```
Overflow records per track = Track capacity / Length of overflow records
```

Example: Approximately 5000 overflow records are expected for the data set described in step 1. Because 55 overflow records will fit on a track, 91 overflow tracks are required. There are 91 overflow tracks for 155 prime data tracks, or approximately 1 overflow track for every 2 prime data tracks. Because the 3380 disk pack for a 3380 Model AD4 has 15 tracks per cylinder, it would probably be best to allocate 5 tracks per cylinder for overflow.

```
Overflow  = 47968/(256+((12+267)/32)(32)+((30+267)/32)(32))
records   = 47968/864
per track = 55

Overflow tracks required = 5000 records / 55 records per track
                         = 91

Overflow tracks per cylinder = 5
```

Step 3. Index Entries Per Track

You will have to set aside space in the prime area for track index entries. There will be two entries (normal and overflow) for each track on a cylinder that contains prime data records. The data field of each index entry is always 10 bytes long. The key length corresponds to the key length for the prime data records. How many index entries will fit on a track?

```
Index entries per track = Track capacity / Length of index entries
```

Example: Again assuming a 3380 Model AD4 disk pack and records with 12-byte keys, 57 index entries fit on a track.

```
Index     = 47968/(256+((12+267)/32)(32)+((10+267)/32)(32))
entries   = 47968/832
per track = 57
```

Step 4. Determine Unused Space

Unused space on the last track of the track index depends on the number of tracks required for track index entries, which in turn depends upon the number of tracks per cylinder and the number of track index entries per track. You can use any unused space for any prime data records that will fit.

```
Unused space = (Number of index entries per track)
               - (2 (Number of tracks per cylinder
               - Number of overflow tracks per cyl.) + 1)
               (Number of bytes per index)
```

For variable-length records, or when a prime data record will not fit on the last track of the track index, the last track of the track index is not shared with prime data records. In this case, if the remainder of the division is less than or equal to 2, drop the remainder. In all other cases, round the quotient up to the next integer.

Example: The 3380 disk pack from the 3380 Model AD4 has 15 tracks per cylinder. You can fit 57 track index entries into one track. Therefore, you need less than 1 track for each cylinder.

```
Number of trk index = (2 (15 - 5) + 1) / (57 + 2)
trks per cylinder   = 21 / 59
```

The space remaining on the track is 47968 − (21 (832)) = 30496 bytes.

This is enough space for 16 blocks of prime data records. Because the normal number of blocks per track is 26, the blocks use 16/26ths of the track, and the effective number of track index tracks per cylinder is therefore 1 − 16/26 or 0.385.

Space is required on the last track of the track index for a dummy entry to show the end of the track index. The dummy entry consists of an 8-byte count field, a key field the same size as the key field in the preceding entries, and a 10-byte data field.

Step 5. Calculate Tracks for Prime Data Records

Next you have to calculate the number of tracks available on each cylinder for prime data records. You cannot include tracks set aside for cylinder overflow records.

```
Prime data tracks = Tracks per cylinder - Overflow tracks per cylinder
per cylinder        - Index tracks per cylinder
```

Example: If you set aside 5 cylinder overflow tracks, and you need 0.385ths of a track for the track index, 9.615 tracks are available on each cylinder for prime data records.

```
Prime data tracks per cylinder  = 15 - 5 - (0.385) = 9.615
```

Step 6. Cylinders Required

The number of cylinders required to allocate prime space is determined by the number of prime data tracks required divided by the number of prime data tracks available on each cylinder. This area includes space for the prime data records, track indexes, and cylinder overflow records.

```
Number of cylinders needed  = Prime data tracks needed
                              / Prime data tracks per cylinder needed
```

Example: You need 155 tracks for prime data records. You can use 9.615 tracks per cylinder. Therefore, you need 17 cylinders for your prime area and cylinder overflow areas.

```
Number of cylinders required = (155) / (9.615) = 16.121  (round up to 17)
```

Step 7. Space for Cylinder Indexes and Track Indexes

You will need space for a cylinder index and track indexes. There is a cylinder index entry for each track index (for each cylinder allocated for the data set). The size of each entry is the same as the size of the track index entries; therefore, the number of entries that will fit on a track is the same as the number of track index entries. Unused space on a cylinder index track is not shared.

```
Number of tracks  = (Track indexes + 1)
required for      / (Index entries per track cylinder index)
```

Example: You have 17 track indexes (from Step 6). Because 57 index entries fit on a track (from Step 3), you need 1 track for your cylinder index. The remaining space on the track is unused.

```
Number of tracks required for cyl. index = (17 + 1) / 57 = 18 / 57 = 0.316 < 1
```

Every time a cylinder index crosses a cylinder boundary, ISAM writes a dummy index entry that lets ISAM chain the index levels together. The addition of dummy entries can increase the number of tracks required for a given index level. To determine how many dummy entries will be required, divide the total number of tracks required by the number of tracks on a cylinder. If the remainder is 0, subtract 1 from the quotient. If the corrected quotient is not 0, calculate the number of tracks these dummy entries require. Also consider any additional cylinder boundaries crossed by the addition of these tracks and by any track indexes starting and stopping within a cylinder.

Step 8. Space for Master Indexes

If you have a data set large enough to require master indexes, you will want to calculate the space required according to the number of tracks for master indexes (NTM parameter) you specified in the DCB macro or the DD statement.

If the cylinder index exceeds the NTM specification, an entry is made in the master index for each track of the cylinder index. If the master index itself exceeds the NTM specification, a second-level master index is started. As many as three levels of master indexes are created if required.

The space requirements for the master index are computed in the same way as those for the cylinder index.

Calculate the number of tracks for master indexes as follows:

```
# Tracks for master indexes = (# Cylinder index tracks + 1) / Index entries per track
```

If the number of cylinder indexes is greater than NTM, calculate the number of tracks for a first level master index as follows:

```
# Tracks for first level master index =
(Cylinder track indexes + 1) / Index entries per track
```

If the number of first level master indexes is greater than NTM, calculate the number of tracks for a second level master index as follows:

```
# Tracks for second level master index =
(First level master index + 1) / Index entries per track
```

If the number of second level master indexes is greater than NTM, calculate the number of tracks for a third level master index as follows:

```
# Tracks for second level master index =
(Second level master index + 1) / Index entries per track
```

Example: Assume that your cylinder index will require 22 tracks. Because large keys are used, only 10 entries will fit on a track. If NTM was specified as 2, 3 tracks will be required for a master index, and two levels of master index will be created.

```
Number of tracks required for master indexes = (22 + 1) / 10 = 2.3
```

Summary of Indexed Sequential Space Requirements Calculations

Indexed sequential space requirement calculations can be summarized as follows:

1. How many blocks will fit on a track?

   ```
   Blocks = Track capacity / Length of blocks
   ```

2. How many overflow records will fit on a track?

   ```
   Overflow records = Track capacity / Length of overflow records per track
   ```

3. How many index entries will fit on a track?

   ```
   Index entries per track = Track capacity / Length of index entries
   ```

4. How much space is left on the last track of the track index?

   ```
   Unused    = (Number of index entries per track)
   space     - (2 (Number of tracks per cylinder
             - Number of overflow tracks per cylinder) + 1)
               (Number of bytes per index)
   ```

5. How many tracks on each cylinder can you use for prime data records?

   ```
   Prime data  = Tracks per cylinder
   tracks per  - Overflow tracks per cylinder
               - Index tracks per cylinder
   ```

6. How many cylinders do you need for the prime data area?

   ```
   Number of cylinders needed = Prime data tracks needed / Prime data tracks per cylinder
   ```

7. How many tracks do you need for the cylinder index?

   ```
   Number of tracks required = (Track indexes + 1) / Index entries per track
   for cylinder index
   ```

8. How many tracks do you need for master indexes?

   ```
   Number of tracks required = (Number of cylinder index tracks + 1)
   for master indexes        / Index entries per track
   ```

Retrieving and Updating

Retrieving and updating an indexed sequential data set can be accomplished either sequentially or directly, as described in this section.

Sequential Retrieval and Update

To sequentially retrieve and update records in an indexed sequential data set, take the following actions:

1. Code DSORG=IS or DSORG=ISU to agree with what you specified when you allocated the data set, and MACRF=GL, MACRF=SK, or MACRF=PU in the DCB macro.
2. Code a DD statement for retrieving the data set. The data set characteristics and options are as defined when the data set was allocated.
3. Open the data set.
4. Set the beginning of sequential retrieval (SETL).
5. Retrieve records and process as required, marking records for deletion as required.
6. Return records to the data set.
7. Use ESETL to end sequential retrieval as required and reset the starting point.
8. Close the data set to end all retrieval.

Using the data set allocated in Figure 122 on page 552, assume that you are to retrieve all records whose keys begin with 915. Those records with a date (positions 13 through 16) before the current date are to be deleted. The date is in the standard form as returned by the system in response to the TIME macro, that is, packed decimal 0cyyddds. Overflow records can be logically deleted even though they cannot be physically deleted from the data set.

Figure 124 on page 562 shows how to update an indexed sequential data set sequentially.

```
//INDEXDD   DD       DSNAME=SLATE.DICT,---
            ...
ISRETR      START    0
            DCBD     DSORG=IS
ISRETR      CSECT
            ...
            USING    IHADCB,3
            LA       3,ISDATA
            OPEN     (ISDATA)
            SETL     ISDATA,KC,KEYADDR    Set scan limit
            TIME     ,                    Today's date in register 1
            ST       1,TODAY
NEXTREC     GET      ISDATA               Locate mode
            CLC      19(10,1),LIMIT
            BNL      ENDJOB
            CP       12(4,1),TODAY        Compare for old date
            BNL      NEXTREC
            MVI      0(1),X'FF'           Flag old record for
                                           deletion
            PUTX     ISDATA               Return delete record
            B        NEXTREC
TODAY       DS       F
KEYADDR     DC       C'915'               Key prefix
            DC       XL7'0'               Key padding
LIMIT       DC       C'916'
            DC       XL7'0'
            ...
CHECKERR
```

Test DCBEXCD1 and DCBEXDE2 for error indication: Error Routines

```
ENDJOB      CLOSE    (ISDATA)
            ...
ISDATA      DCB      DDNAME=INDEXDD,DSORG=IS,MACRF=(GL,SK,PU),      C
            ...      SYNAD=CHECKRR
```

Figure 124. Sequentially Updating an Indexed Sequential Data Set

Direct Retrieval and Update

By using the basic indexed sequential access method (BISAM) to process an indexed sequential data set, you can directly access the records in the data set for:

- Direct retrieval of a record by its key
- Direct update of a record
- Direct insertion of new records.

Because the operations are direct, there is no anticipatory buffering. However, if ‘S’ is specified on the READ macro, the system provides dynamic buffering each time a read request is made. (See Figure 125 on page 565.)

Ensuring a Record is in Virtual Storage

To ensure that the requested record is in virtual storage before you start processing, you must issue a WAIT or CHECK macro. If you issue a WAIT macro, you must test the exception code field of the DECB. If you issue a CHECK macro, the system tests the exception code field in the DECB.

If an error analysis routine has not been specified and a CHECK is issued, and an error situation exists, the program abnormally ends with a system completion code of X'001'. For both WAIT and CHECK, if you want to determine whether the record is an overflow record, you should test the exception code field of the DECB.

After you test the exception code field, you need not set it to 0. If you have used a READ KU (read an updated record) macro, and if you plan to use the same DECB again to rewrite the updated record using a WRITE K macro, you should not set the field to 0. If you do, your record might not be rewritten properly.

Updating Existing Records

To update existing records, you must use the READ KU and WRITE K combination. Because READ KU implies that the record will be rewritten in the data set, the system retains the DECB and the buffer used in the READ KU and uses them when the record is written. If you decide not to write the record, you should use the same DECB in another READ or WRITE macro, or if dynamic buffering was used, issue a FREEDBUF macro. If you issue several READ KU or WRITE K macros before checking the first one, you could destroy some of your updated records unless the records are from different blocks.

When you are using scan mode with QISAM and you want to issue PUTX, issue an ENQ on the data set before processing it and a DEQ after processing is complete. ENQ must be issued before the SETL macro, and DEQ must be issued after the ESETL macro. When you are using BISAM to update the data set, do not modify any DCB fields or issue a DEQ until you have issued CHECK or WAIT.

Sharing a BISAM DCB between Related Tasks

If there is the possibility that your task and another task will be accessing the same data set simultaneously, or the same task has two or more DCBs opened for the same data set, use data set sharing. You specify data set sharing by coding DISP=SHR in your DD statement. When a data set is shared, the DCB fields are maintained for your program to process the data set correctly. If you do not use DISP=SHR, and more than one DCB is open for updating the data set, the results are unpredictable.

If you specify DISP=SHR, you must also issue an ENQ for the data set before each I/O request and a DEQ on completion of the request. All users of the data set must use the same *qname* and *rname* operands for ENQ. For example, you might use the data set name as the *qname* operand. For more information about using ENQ and DEQ, see *z/OS MVS Programming: Assembler Services Reference ABE-HSP* and *z/OS MVS Programming: Assembler Services Guide*.

Subtasking

For subtasking, I/O requests should be issued by the task that owns the DCB or a task that will remain active while the DCB is open. If the task that issued the I/O request ends, the storage used by its data areas (such as IOBs) can be freed, or queuing switches in the DCB work area can be left on, causing another task issuing an I/O request to the DCB to program check or to enter the wait state.

For example, if a subtask issues and completes a READ KU I/O request, the IOB created by the subtask is attached to the DCB update queue. (READ KU means the record retrieved is to be updated.) If that subtask ends, and subpool zero is not shared with the subtask owning the DCB, the IOB storage area is freed and the integrity of the ISAM update queue is destroyed. A request from another subtask, attempting to use that queue, could cause unpredictable abends. As another example, if a WRITE KEY NEW is in process when the subtask ends, a 'WRITE-KEY-NEW-IN-PROCESS' bit is left on. If another I/O request is issued to the DCB, the request is queued but cannot proceed.

Direct Updating with Exclusive Control

In the example shown in Figure 125 on page 565, the previously described data set is to be updated directly with transaction records on tape. The input tape records

are 30 characters long, the key is in positions 1 through 10, and the update information is in positions 11 through 30. The update information replaces data in positions 31 through 50 of the indexed sequential data record.

Exclusive control of the data set is requested, because more than one task might be referring to the data set at the same time. Notice that, to avoid tying up the data set until the update is completed, exclusive control is released after each block is written.

Using FREEDBUF: Note the use of the FREEDBUF macro in Figure 125 on page 565. Usually, the FREEDBUF macro has two functions:

- To indicate to the ISAM routines that a record that has been read for update will not be written back
- To free a dynamically obtained buffer.

In Figure 125, because the read operation was unsuccessful, the FREEDBUF macro frees only the dynamically obtained buffer.

The first function of FREEDBUF lets you read a record for update and then decide not to update it without performing a WRITE for update. You can use this function even when your READ macro does not specify dynamic buffering, if you have included S (for dynamic buffering) in the MACRF field of your READ DCB.

You can cause an automatic FREEDBUF merely by reusing the DECB; that is, by issuing another READ or a WRITE KN to the same DECB. You should use this feature whenever possible, because it is more efficient than FREEDBUF. For example, in Figure 125, the FREEDBUF macro could be eliminated, because the WRITE KN addressed the same DECB as the READ KU.

```
//INDEXDD   DD       DSNAME=SLATE.DICT,DCB=(DSORG=IS,BUFNO=1,...),---
//TAPEDD    DD       ---
            ...
ISUPDATE    START    0
            ...
NEXTREC     GET      TPDATA,TPRECORD
            ENQ      (RESOURCE,ELEMENT,E,,SYSTEM)
            READ     DECBRW,KU,,'S',MF=E    Read into dynamically
*                                           obtained buffer
            WAIT     ECB=DECBRW
            TM       DECBRW+24,X'FD'        Test for any condition
            BM       RDCHECK                but overflow
            L        3,DECBRW+16            Pick up pointer to
*                                           record
            MVC      ISUPDATE-ISRECORD      Update record
            WRITE    DECBRW,K,MF=E
            WAIT     ECB=DECBRW
            TM       DECBRW+24,X'FD'        Any errors?
            BM       WRCHECK
            DEQ      (RESOURCE,ELEMENT,,SYSTEM)
            B        NEXTREC
RDCHECK     TM       DECBRW+24,X'80'        No record found
            BZ       ERROR                  If not, go to error
*                                           routine
            FREEDBUF DECBRW,K,ISDATA        Otherwise, free buffer
            MVC      ISKEY,KEY              Key placed in ISRECORD
            MVC      ISUPDATE,UPDATE        Updated information
*                                           placed in ISRECORD
            WRITE    DECBRW,KN,,WKNAREA,'S',MF=E Add record to data set
            WAIT     ECB=DECBRW
            TM       DECBRW+24,X'FD'        Test for errors
            BM       ERROR
            DEQ      (RESOURCE,ELEMENT,,SYSTEM)  Release exclusive
*                                          control
            B        NEXTREC
WKNAREA     DS       4F                     BISAM WRITE KN work field
ISRECORD    DS       0CL50                  50-byte record from ISDATA
            DS       CL19                   DCB First part of ISRECORD
ISKEY       DS       CL10                   Key field of ISRECORD
            DS       CL1                    Part of ISRECORD
ISUPDATE    DS       CL20                   Update area of ISRECORD
            ORG      ISUPDATE               Overlay ISUPDATE with
TPRECORD    DS       0CL30                  TPRECORD 30-byte record
KEY         DS       CL10                   from TPDATA DCB Key
*                                            for locating
UPDATE      DS       CL20                   ISDATA record update
RESOURCE    DC       CL8'SLATE'             information or new data
ELEMENT     DC       C'DICT'
            READ     DECBRW,KU,ISDATA,'S','S',KEY,MF=L
ISDATA      DCB      DDNAME=INDEXDD,DSORG=IS,MACRF=(RUS,WUA),       C
                     MSHI=INDEX,SMSI=2000
TPDATA      DCB      ---
INDEX       DS       2000C
            ...
```

Figure 125. Directly Updating an Indexed Sequential Data Set

Using Other Updating Methods: For an indexed sequential data set with variable-length records, you can make three types of updates by using the basic access method. You can read a record and write it back with no change in its length, simply updating some part of the record. You do this with a READ KU, followed by a WRITE K, the same way you update fixed-length records.

Two other methods for updating variable-length records use the WRITE KN macro and lets you change the record length. In one method, a record read for update (by a READ KU) can be updated in a manner that will change the record length and then be written back with its new length by a WRITE KN (key new). In the second method, you can replace a record with another record having the same key and possibly a different length using the WRITE KN macro. To replace a record, it is not necessary to have first read the record.

In either method, when changing the record length, you must place the new length in the DECBLGTH field of the DECB before issuing the WRITE KN macro. If you use a WRITE KN macro to update a variable-length record that has been marked for deletion, the first bit (no record found) of the exceptional condition code field (DECBEXC1) of the DECB is set on. If this condition is found, the record must be written using a WRITE KN with nothing specified in the DECBLGTH field.

Recommendation: Do not try to use the DECBLGTH field to determine the length of a record read because DECBLGTH is for use with writing records, not reading them.

If you are reading fixed-length records, the length of the record read is in DCBLRECL, and if you are reading variable-length records, the length is in the record descriptor word (RDW).

Direct Update with Variable-Length Records

In Figure 126 on page 567, an indexed sequential data set with variable-length records is updated directly with transaction records on tape. The transaction records are of variable length and each contains a code identifying the type of transaction. Transaction code 1 means that an existing record is to be replaced by one with the same key; code 2 means that the record is to be updated by appending additional information, thus changing the record length; code 3 or greater means that the record is to be updated with no change to its length.

For this example, the maximum record length of both data sets is 256 bytes. The key is in positions 6 through 15 of the records in both data sets. The transaction code is in position 5 of records on the transaction tape. The work area (REPLAREA) size is equal to the maximum record length plus 16 bytes.

Adding Records

You can use either the queued access method or the basic access method to add records to an indexed sequential data set. To insert a record between existing records in the data set, you must use the basic access method and the WRITE KN (key new) macro. Records added to the end of a data set (that is, records with successively higher keys), can be added to the prime data area or the overflow area by the basic access method using WRITE KN, or they can be added to the prime data area by the queued access method using the PUT macro.

Figure 126 on page 567 shows an example of directly updating an indexed sequential data set with variable-length records.

```
//INDEXDD   DD     DSNAME=SLATE.DICT,DCB=(DSORG=IS,BUFNO=1,...),---
//TAPEDD    DD     ---
            ...
ISUPDVLR    START  0
            ...
NEXTREC     GET    TPDATA,TRANAREA
            CLI    TRANCODE,2                Determine if replacement or
*                                            other transaction
            BL     REPLACE                   Branch if replacement
            READ   DECBRW,KU,,'S','S',MF=E   Read record for update
            CHECK  DECBRW,DSORG=IS           Check exceptional conditions
            CLI    TRANCODE,2                Determine if change or append
            BH     CHANGE                    Branch if change
            ...
            ...
* CODE TO MOVE RECORD INTO REPLACEA+16 AND APPEND DATA FROM TRANSACTION
* RECORD
            ...
            MVC    DECBRW+6(2),REPLAREA+16    Move new length from RDW
*                                             into DECBLGTH (DECB+6)
            WRITE  DECBRW,KN,,REPLAREA,MF=E   Rewrite record with
*                                             changed length
            CHECK  DECBRW,DSORG=IS
            B      NEXTREC
CHANGE      ...
            ...
* CODE TO CHANGE FIELDS OR UPDATE FIELDS OF THE RECORD
            ...
            WRITE  DECBRW,K,MF=E               Rewrite record with no
*                                              change of length
            CHECK  DECBRW,DSORG=IS
            B      NEXTREC
REPLACE     MVC    DECBRW+6(2),TRANAREA        Move new length from RDW
*                                              into DECBLGTH (DECB+6)
            WRITE  DECBRW,KN,,TRANAREA-16,MF=E Write transaction record
*                                              as replacement for record
*                                              with the same key
            CHECK  DECBRW,DSORG=IS
            B      NEXTREC
CHECKERR    ...                  SYNAD routine
            ...
REPLAREA    DS     CL272
TRANAREA    DS     CL4
TRANCODE    DS     CL1
KEY         DS     CL10
TRANDATA    DS     CL241
            READ   DECBRW,KU,ISDATA,'S','S',KEY,MF=L
ISDATA      DCB    DDNAME=INDEXDD,DSORG=IS,MACRF=(RUSC,WUAC),SYNAD=CHECKERR
TPDATA      DCB    ---
            ...
```

Figure 126. Directly Updating an Indexed Sequential Data Set with Variable-Length Records

Inserting New Records

As you add records to an indexed sequential data set, the system inserts each record in its proper sequence according to the record key. The remaining records on the track are then moved up one position each. If the last record does not fit on the track, it is written in the first available location in the overflow area. A 10-byte link field is added to the record put in the overflow area to connect it logically to the correct track. The proper adjustments are made to the track index entries. This procedure is illustrated in Figure 127 on page 569.

Subsequent additions are written either on the prime track or as part of the overflow chain from that track. If the addition belongs after the last prime record on a track but before a previous overflow record from that track, it is written in the first available location in the overflow area. Its link field contains the address of the next record in the chain.

For BISAM, if you add a record that has the same key as a record in the data set, a “duplicate record” condition is shown in the exception code. However, if you specified the delete option and the record in the data set is marked for deletion, the condition is not reported and the new record replaces the existing record. For more information about exception codes, see *z/OS DFSMS Macro Instructions for Data Sets*.

Adding New Records to the End of a Data Set

Records added to the end of a data set (that is, records with successively higher keys), can be added by the basic access method using WRITE KN, or by the queued access method using the PUT macro (resume load). In either case, records can be added to the prime data area.

When you use the WRITE KN macro, the record being added is placed in the prime data area only if there is room for it on the prime data track containing the record with the highest key currently in the data set. If there is not sufficient room on that track, the record is placed in the overflow area and linked to that prime track, even though additional prime data tracks originally allocated have not been filled.

When you use the PUT macro, records are added to the prime data area until the space originally allocated is filled. After this allocated prime area is filled, you can add records to the data set using WRITE KN, in which case they will be placed in the overflow area. Resume load is discussed in more detail under “Creating an ISAM Data Set” on page 550.

To add records with successively higher keys using the PUT macro:

- The key of any record to be added must be higher than the highest key currently in the data set.
- The DD statement must specify DISP=MOD or specify the EXTEND option in the OPEN macro.
- The data set must have been successfully closed when it was allocated or when records were previously added using the PUT macro.

You can continue to add fixed-length records in this manner until the original space allocated for prime data is exhausted.

When you add records to an indexed sequential data set using the PUT macro, new entries are also made in the indexes. During resume load on a data set with a partially filled track or a partially filled cylinder, the track index entry or the cylinder index entry is overlaid when the track or cylinder is filled. If resume load abnormally ends after these index entries have been overlaid, a subsequent resume load will get a sequence check when adding a key that is higher than the highest key at the last successful CLOSE but lower than the key in the overlaid index entry. When the SYNAD exit is taken for a sequence check, register 0 contains the address of the highest key of the data set. Figure 127 on page 569 graphically represents how records are added to an indexed sequential data set.

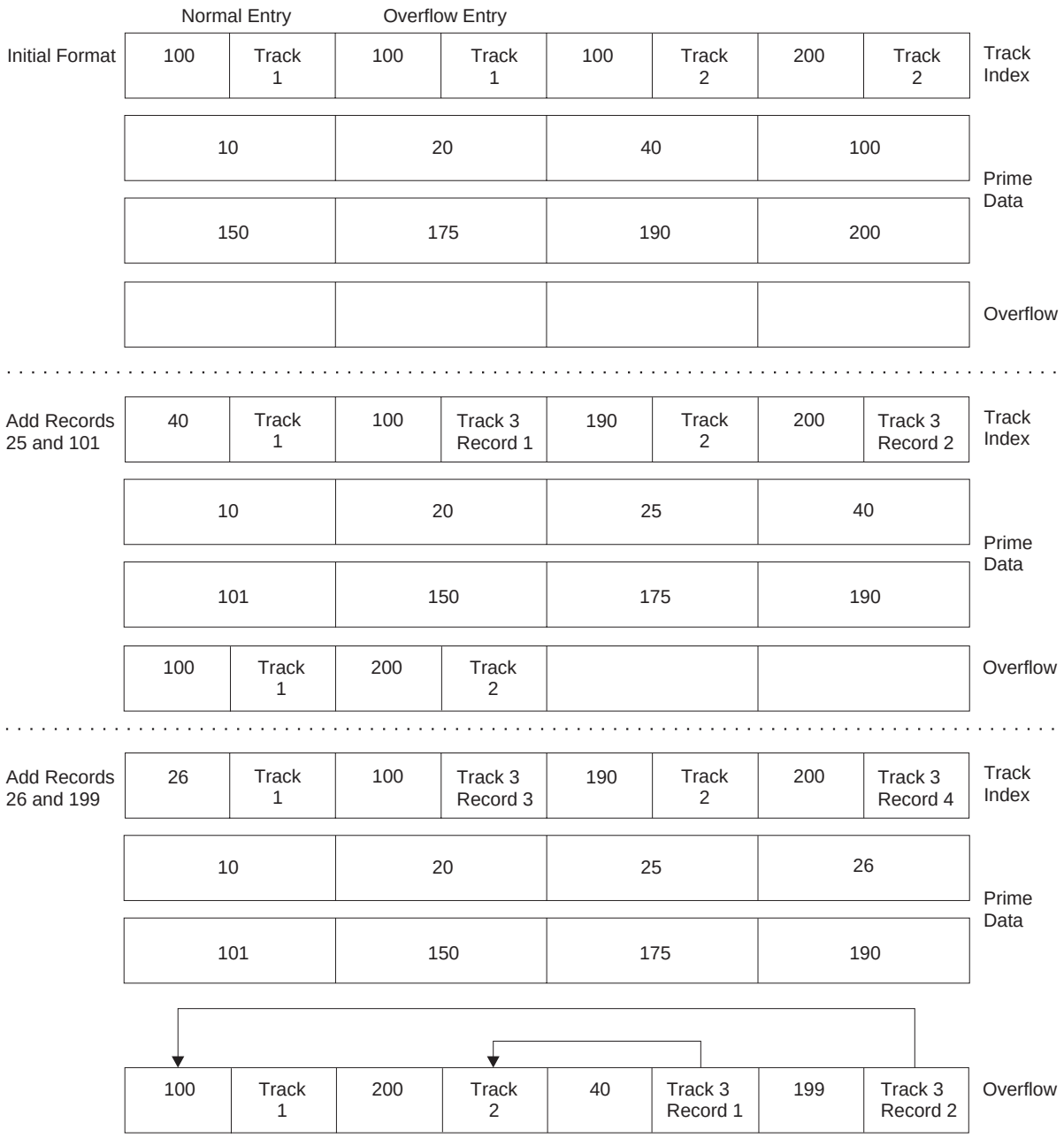


Figure 127. Adding Records to an Indexed Sequential Data Set

Maintaining an Indexed Sequential Data Set

An indexed sequential data set must be reorganized occasionally for two reasons: The overflow area eventually fill. Additions increase the time required to locate records directly. The frequency of reorganization depends on the activity of the data set and on your timing and storage requirements. There are two ways to reorganize the data set:

- In two passes by writing it sequentially into another area of direct access storage or magnetic tape and then re-creating it in the original area.
- In one pass by writing it directly into another area of direct access storage. In this case, the area occupied by the original data set cannot be used by the reorganized data set.

The operating system maintains statistics that are pertinent to reorganization. The statistics, written on the direct access volume and available in the DCB for checking, include the number of cylinder overflow areas, the number of unused tracks in the independent overflow area, and the number of references to overflow records other than the first. They appear in the RORG1, RORG2, and RORG3 fields of the DCB.

When creating or updating the data set, if you want to be able to flag records for deletion during updating, set the delete code (the first byte of a fixed-length record or the fifth byte of a variable-length record) to X'FF'. Figure 128 describes the process for deleting indexed data set records, and how a flagged record will not be rewritten in the overflow area after it has been forced off its prime track (unless it has the highest key on that cylinder) during a subsequent update.

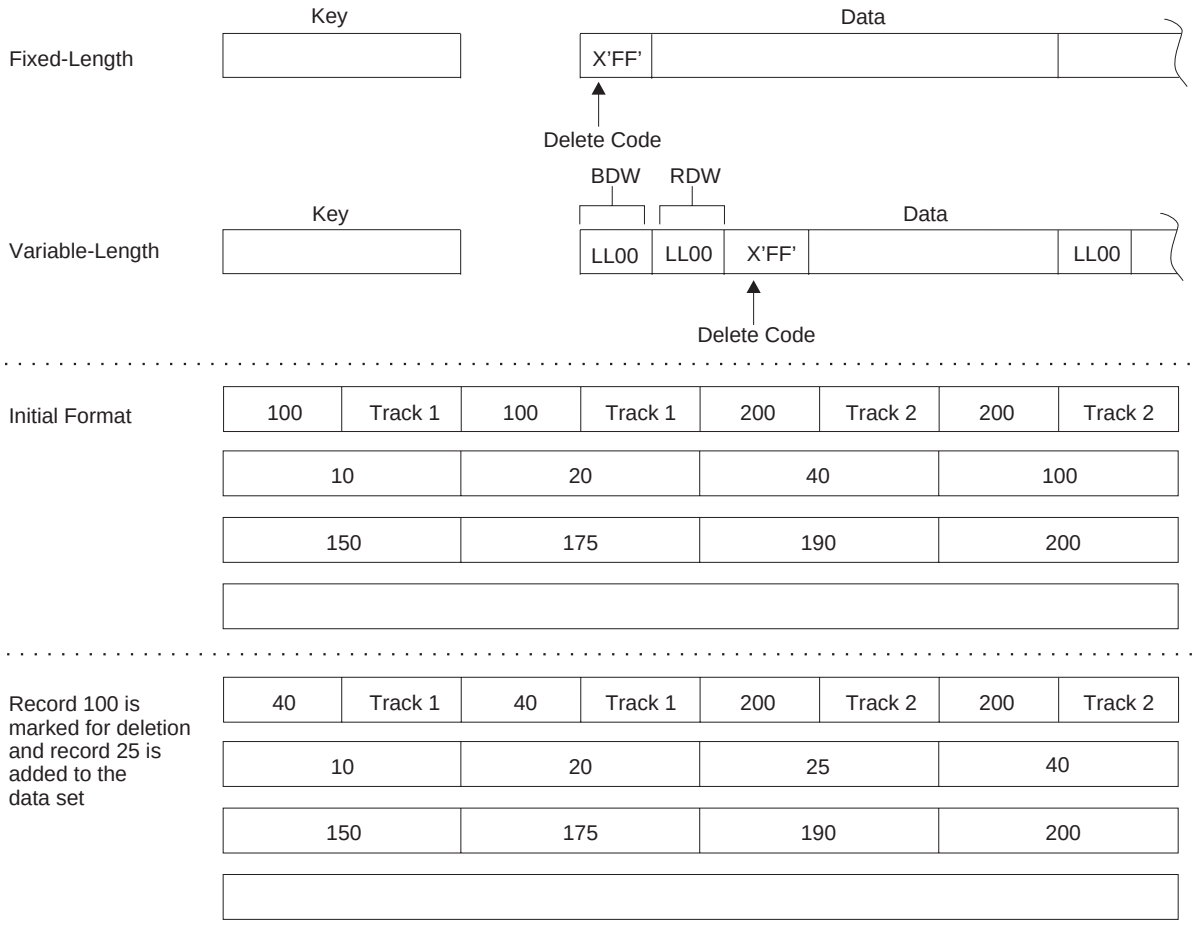


Figure 128. Deleting Records from an Indexed Sequential Data Set

Similarly, when you process sequentially, flagged records are not retrieved for processing. During direct processing, flagged records are retrieved the same as any other records, and you should check them for the delete code.

A WRITE KN instruction for a data set containing variable-length records removes all the deleted records from that prime data track. Also, to use the delete option, RKP must be greater than 0 for fixed-length records and greater than 4 for variable-length records.

Buffer Requirements

The only case in which you will ever have to compute the buffer length (BUFL) requirements for your program occurs when you use the BUILD or GETPOOL macro to construct the buffer area. If you are creating an indexed sequential data set (using the PUT macro), each buffer must be 8 bytes longer than the block size to allow for the hardware count field. That is:

```
Buffer length = 8 + Block size

 +-------+-------------------------+
 |  (8)  |          Data           |
 |       |       (BLKSIZE)         |
 +-------+-------------------------+
 <------------------Buffer------------------>
```

One exception to this formula arises when you are dealing with an unblocked format-F record whose key field precedes the data field; its relative key position is 0 (RKP=0). In that case, the key length must also be added:

```
Buffer length = 8 + Key length + Record length

 +-------+-----------+-------------------+
 |  (8)  |    Key    |       Data        |
 |       |  (KEYLEN) |      (LRECL)      |
 +-------+-----------+-------------------+
 <------------------Buffer------------------>
```

The buffer requirements for using the queued access method to read or update (using the GET or PUTX macro) an indexed sequential data set are discussed below.

For fixed-length unblocked records when both the key and data are to be read, and for variable-length unblocked records, padding is added so that the data will be on a doubleword boundary, that is:

```
Buffer length = Key length + Padding + 10 + Block size

 +-----------+---------+---------------+-----------+
 |    Key    |         | ISAM Link Field |   Data    |
 |  (KEYLEN) | Padding |     (10)      | (BLKSIZE) |
 +-----------+---------+---------------+-----------+
 <------------------Buffer------------------>
```

For fixed-length unblocked records when only data is to be read:

```
Buffer length = 16 + LRECL

 +-----------+---------------+-------------------+
 |  Padding  | ISAM Link Field |       Data        |
 |    (6)    |     (10)      |      (LRECL)      |
 +-----------+---------------+-------------------+
 <------------------Buffer------------------>
```

The buffer area for fixed-length blocked records must be:

```
Buffer length = 16 + Block size

| Padding | ISAM Link Field | Data |
|   (6)   |     (10)        | (BLKSIZE) |

<------------------Buffer------------------>
```

Tip: When you use the basic access method to update records in an indexed sequential data set, the key length field need not be considered in determining your buffer requirements.

For variable-length blocked records, padding is 2 if the buffer starts on a fullword boundary that is not also a doubleword boundary, or 6 if the buffer starts on a doubleword boundary. The buffer area must be:

```
Buffer length = 12 or 16 + Block size

| Padding | ISAM Link Field | Data |
|   (6)   |     (10)        | (BLKSIZE) |

<------------------Buffer------------------>
```

Work Area Requirements

If you are using the input data set with fixed-length, unblocked records as a basis for creating a new data set, a work area is required. The size of the work area is given by:

```
Work area = Key length + Record length

| Key | Data |
|     | (LRECL) |

<------------------Work Area------------------>
```

If you are reading only the data portion of fixed-length unblocked records or variable-length records, the work area is the same size as the record.

```
Work area = Record length

| Data |
| (LRECL) |

<------------------Work Area------------------>
```

You can save processing time by adding fixed-length or variable-length records to a data set by using the MSWA (main storage work area) parameter of the DCB macro to provide a special work area for the operating system. The size of the work area (SMSW parameter in the DCB) must be large enough to contain a full track of data, the count fields of each block, and the work space for inserting the new record.

The size of the work area needed varies according to the record format and the device type. You can calculate it during execution using device-dependent information obtained with the TRKCALC macro, DEVTYPE macro, and data set

information from the DSCB obtained with the OBTAIN macro. The TRKCALC, DEVTYPE and OBTAIN macros are discussed in *z/OS DFSMSdfp Advanced Services*.

Restriction: You can use the TRKCALC or DEVTYPE macro only if the index and prime areas are on devices of the same type or if the index area is on a device with a larger track capacity than the device containing the prime area.

If you do not need to maintain device independence, you can precalculate the size of the work area needed and specify it in the SMSW field of the DCB macro. The maximum value for SMSW is 65 535.

Calculating the Size of the Work Area

For calculating the size of the work area, see the IBM storage device publication specific to your device.

For fixed-length blocked records, the size of the main storage work area (SMSW) is calculated as follows:

```
SMSW = (DS2HIRPR) (BLKSIZE + 8) + LRECL + KEYLEN
```

The formula for fixed-length unblocked records is

```
SMSW = (DS2HIRPR) (KEYLEN + LRECL + 8) + 2
```

The value for DS2HIRPR is in the index (format-2) DSCB. If you do not use the MSWA and SMSW parameters, the control program supplies a work area using the formula BLKSIZE + LRECL + KEYLEN.

For variable-length records, SMSW can be calculated by one of two methods. The first method can lead to faster processing, although it might require more storage than the second method.

The first method is:

```
SMSW = DS2HIRPR (BLKSIZE + 8) + LRECL + KEYLEN + 10
```

The second method is as follows:

```
SMSW = ( (Trk Cap - Bn + 1) / Block length) (BLKSIZE)
       + 8 (DS2HIRPR) + LRECL + KEYLEN + 10 + (REM - N - KEYLEN)
```

The second method yields a minimum value for SMSW. Therefore, the first method is valid only if its application results in a value higher than the value that would be derived from the second method. If neither MSWA nor SMSW is specified, the control program supplies the work area for variable-length records, using the second method to calculate the size.

In all the above formulas, the terms BLKSIZE, LRECL, KEYLEN, and SMSW are the same as the parameters in the DCB macro (Trk Cap=track capacity). REM is the remainder of the division operation in the formula and N is the first constant in the block length formulas. (REM-N-KEYLEN) is added only if its value is positive.

Space for the Highest-Level Index

Another technique to increase the speed of processing is to provide space in virtual storage for the highest-level index. To specify the address of this area, use the MSHI (main storage highest-level index) parameter of the DCB. When the address of that area is specified, you must also specify its size, which you can do by using the SMSI (size of main storage index) parameter of the DCB. The maximum value

for SMSI is 65 535. If you do not use this technique, the index on the volume must be searched. If the high-level index is greater than 65 535 bytes in length, your request for the high-level index in storage is ignored.

The size of the storage area (SMSI parameter) varies. To allocate that space during execution, you can find the size of the high-level index in the DCBNCRHI field of the DCB during your DCB user exit routine or after the data set is open. Use the DCBD macro to gain access to the DCBNCRHI field (see “Chapter 22. Specifying and Initializing Data Control Blocks” on page 313). You can also find the size of the high-level index in the DS2NOBYT field of the index (format 2) DSCB, but you must use the utility program IEHLIST to print the information in the DSCB. You can calculate the size of the storage area required for the high-level index by using the formula

```
SMSI = (Number of Tracks in High-Level Index)
       (Number of Entries per Track)
       (Key Length + 10)
```

The formula for calculating the number of tracks in the high-level index is in “Calculating Space Requirements” on page 556. When a data set is shared and has the DCB integrity feature (DISP=SHR), the high-level index in storage is not updated when DCB fields are changed.

Device Control

An indexed sequential data set is processed sequentially or directly. Direct processing is accomplished by the basic access method. Because you provide the key for the record you want read or written, all device control is handled automatically by the system. If you are processing the data set sequentially, using the queued access method, the device is automatically positioned at the beginning of the data set.

In some cases, you might want to process only a section or several separate sections of the data set. You do that by using the SETL macro, which directs the system to begin sequential retrieval at the record having a specific key. The processing of succeeding records is the same as for normal sequential processing, except that you must recognize when the last desired record has been processed. At this point, issue the ESETL macro to ends sequential processing. You can then begin processing at another point in the data set. If you do not specify a SETL macro before retrieving the data, the system assumes default SETL values. See the GET and SETL macros in *z/OS DFSMS Macro Instructions for Data Sets*.

SETL—Specifying Start of Sequential Retrieval

The SETL macro lets you retrieve records starting at the beginning of an indexed sequential data set or at any point in the data set. Processing that is to start at a point other than the beginning can be requested in the form of a record key, a key class (key prefix), or an actual address of a prime data record.

The key class is useful because you do not have to know the entire key of the first record to be processed. A key class consists of all the keys that begin with identical characters. The key class is defined by specifying the desired characters of the key class at the address specified in the lower-limit address of the SETL macro and setting the remaining characters to the right of the key class to binary zeros.

To use actual addresses, you must keep a record of where the records were written when the data set was allocated. The device address of the block containing the record just processed by a PUT-move macro is available in the 8-byte data control

block field DCBLPDA. For blocked records, the address is the same for each record in the block.

Retrieval of Deleted Records

Normally, when a data set is allocated with the delete option specified, deleted records cannot be retrieved using the QISAM retrieval mode. When the delete option is not specified in the DCB, the SETL macro options function as follows.

SETL B—Start at the first record in the data set.

SETL K—Start with the record having the specified key.

SETL KH—Start with the record whose key is equal to or higher than the specified key.

SETL KC—Start with the first record having a key that falls into the specified key class.

SETL I—Start with the record found at the specified direct access address in the prime area of the data set.

Because the DCBOPTCD field in the DCB can be changed after the data set is allocated (by respecifying the OPTCD in the DCB or DD statement), it is possible to retrieve deleted records. Then, SETL functions as noted above.

When the delete option is specified in the DCB, the SETL macro options function as follows.

SETL B—Start retrieval at the first undeleted record in the data set.

SETL K—Start retrieval at the record matching the specified key, if that record is not deleted. If the record is deleted, an NRF (no record found) indication is set in the DCBEXCD field of the DCB, and SYNAD is given control.

SETL KH—Start with the first undeleted record whose key is equal to or higher than the specified key.

SETL KC—Start with the first undeleted record having a key that falls into the specified key class or follows the specified key class.

SETL I—Start with the first undeleted record following the specified direct access address.

Without the delete option specified, QISAM retrieves and handles records marked for deletion as nondeleted records.

Regardless of the SETL or delete option specified, the NRF condition will be posted in the DCBEXCD field of the DCB, and SYNAD is given control if the key or key class:

- Is higher than any key or key class in the data set
- Does not have a matching key or key class in the data set

ESETL—Ending Sequential Retrieval

The ESETL macro directs the system to stop retrieving records from an indexed sequential data set. A new scan limit can then be set, or processing ends. An end-of-data-set indication automatically ends retrieval. An ESETL macro must be run before another SETL macro (described above) using the same DCB is run.

Note: If the previous SETL macro completed with an error, an ESETL macro should be run before another SETL macro.

Appendix E. Using ISAM Programs with VSAM Data Sets

This appendix covers the following topics.

Topic	Page
How an ISAM Program Can Process a VSAM Data Set	580
Converting an Indexed Sequential Data Set	585
JCL for Processing with the ISAM Interface	586
Restrictions on the Use of the ISAM Interface	588

This appendix is intended to help you use ISAM programs with VSAM data sets. Use of ISAM is not recommended. The information in this appendix is shown to facilitate conversion to VSAM.

Although the ISAM interface is an efficient way of processing your existing ISAM programs, all new programs that you write should be VSAM programs. Indexed sequential data sets should be migrated to VSAM key-sequenced data sets. Existing programs can use the ISAM/VSAM interface to access those data sets and need not be deleted. You can use the REPRO command with the ENVIRONMENT keyword to handle the ISAM “dummy” records.

SMS does not support indexed sequential data sets.

VSAM, through its ISAM interface program, allows a debugged program that processes an indexed sequential data set to process a key-sequenced data set. The key-sequenced data set can have been converted from an indexed-sequential or a sequential data set (or another VSAM data set) or can be loaded by one of your own programs. The loading program can be coded with VSAM macros, ISAM macros, PL/I statements, or COBOL statements. That is, you can load records into a newly defined key-sequenced data set with a program that was coded to load records into an indexed sequential data set.

Figure 129 on page 580 shows the relationship between ISAM programs processing VSAM data with the ISAM interface and VSAM programs processing the data.

© Copyright IBM Corp. 1987, 2002

Using ISAM Programs with VSAM Data Sets

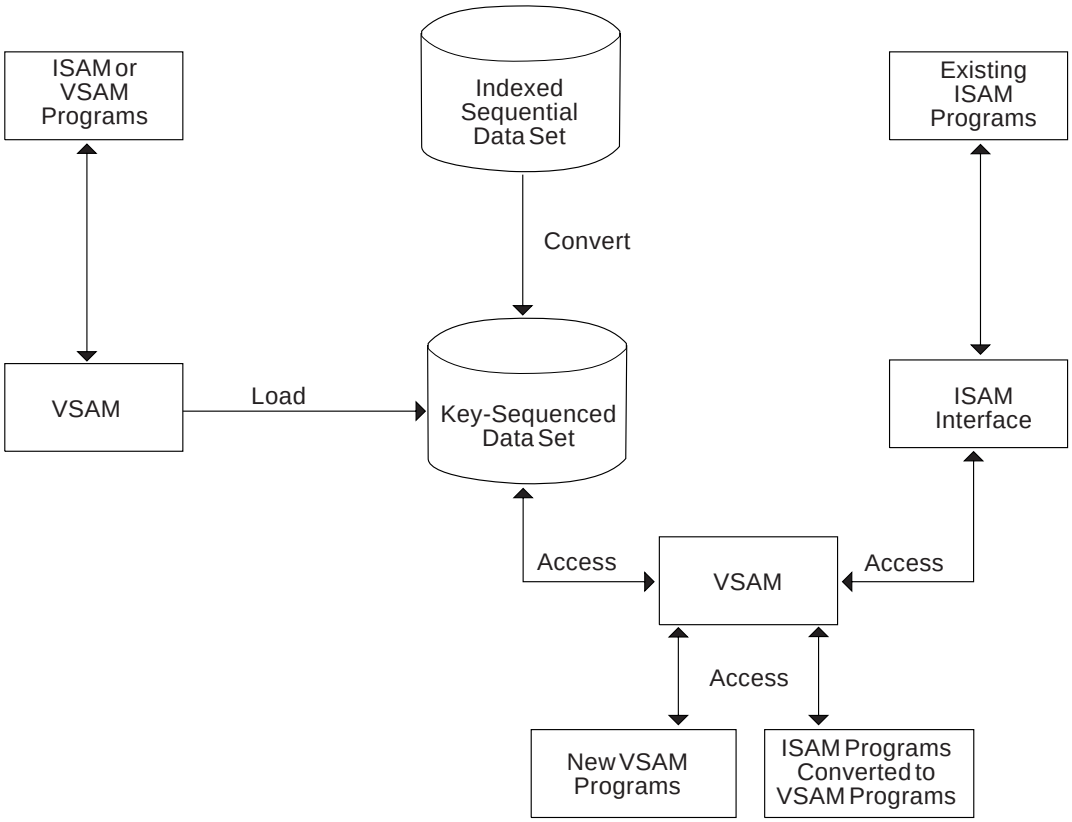


Figure 129. Use of ISAM Processing Programs

There are some minor restrictions on the types of processing an ISAM program can do if it is to be able to process a key-sequenced data set. These restrictions are described in “Restrictions on the Use of the ISAM Interface” on page 588.

Significant performance improvement can be gained by modifying an ISAM program that issues multiple OPEN and CLOSE macros to switch between a QISAM and BISAM DCB. The ISAM program can be modified to open the QISAM and BISAM DCBs at the beginning of the program and to close them when all processing is complete. The performance improvement is proportional to the frequency of OPEN and CLOSE macros in the ISAM program.

How an ISAM Program Can Process a VSAM Data Set

When a processing program that uses ISAM (assembler-language macros, PL/I, or COBOL) issues an OPEN to open a key-sequenced data set, the ISAM interface is given control to:

- Construct control blocks that are required by VSAM
- Load the appropriate ISAM interface routines into virtual storage
- Initialize the ISAM DCB (data control block) to enable the interface to intercept ISAM requests
- Take the DCB user exit requested by the processing program

The ISAM interface intercepts each subsequent ISAM request, analyzes it to determine the equivalent keyed VSAM request, defines the keyed VSAM request in a request parameter list, and initiates the request.

The ISAM interface receives return codes and exception codes for logical and physical errors from VSAM, translates them to ISAM codes, and routes them to the processing program or error-analysis (SYNAD) routine through the ISAM DCB or DECB. Table 48 shows QISAM error conditions and the meaning they have when the ISAM interface is being used.

Table 48. QISAM Error Conditions

Byte and Offset	QISAM Meaning	Error Detected By	Request Parameter List Error Code	Interface/VSAM Meaning
DCBEXCD1				
Bit 0	Record not found	Interface		Record not found (SETL K for a deleted record)
		VSAM	16	Record not found
		VSAM	24	Record on nonmountable volume
Bit 1	Invalid device address	–	–	Always 0
Bit 2	Space not found	VSAM	28	Data set cannot be extended
		VSAM	40	Virtual storage not available
Bit 3	Invalid request	Interface		Two consecutive SETL requests
		Interface		Invalid SETL (I or ID)
		Interface		Invalid generic key (KEY=0)
		VSAM	4	Request after end-of-data
		VSAM	20	Exclusive use conflict
		VSAM	36	No key range defined for insertion
		VSAM	64	Placeholder not available for concurrent data-set positioning
		VSAM	96	Key change attempted
Bit 4	Uncorrectable input error	VSAM	4	Physical read error (register 15 contains a value of 12) in the data component
		VSAM	8	Physical read error (register 15 contains a value of 12) in the index component
		VSAM	12	Physical read error (register 15 contains a value of 12) in the sequence set of the index
Bit 5	Uncorrectable output error	VSAM	16	Physical write error (register 15 contains a value of 12) in the data component
		VSAM	20	Physical write error (register 15 contains a value of 12) in the index component
		VSAM	24	Physical write error (register 15 contains a value of 12) in the sequence set of the index
Bit 6	Unreachable block input	VSAM		Logical error not covered by other exception codes
Bit 7	Unreachable block (output)	VSAM		Logical error not covered by other exception codes
DEBEXCD2				
Bit 0	Sequence check	VSAM	12	Sequence check
		Interface		Sequence check (occurs only during resume load)
Bit 1	Duplicate record	VSAM	8	Duplicate record

Table 48. QISAM Error Conditions (continued)

Byte and Offset	QISAM Meaning	Error Detected By	Request Parameter List Error Code	Interface/VSAM Meaning
Bit 2	DCB closed when error routine entered	VSAM		Error in close error routine entered
Bit 3	Overflow record	Interface	–	Always 1
Bit 4	Length of logical record is greater than DCBLRECL (VLR only)	Interface	–	Length of logical record is greater than DCBLRECL (VLR only)
		VSAM	108	Invalid record length
Bits 5-7	Reserved		–	Always 0

Table 49 shows BISAM error conditions and the meaning they have when the ISAM interface is being used.

If invalid requests occur in BISAM that did not occur previously and the request parameter list indicates that VSAM is unable to handle concurrent data-set positioning, the value specified for the STRNO AMP parameter should be increased. If the request parameter list indicates an exclusive-use conflict, reevaluate the share options associated with the data.

Table 49. BISAM Error Conditions

Byte and Offset	BISAM Meaning	Error Detected By	Request Parameter List Error Code	Interface/VSAM Meaning
DCBEXC1				
Bit 0	Record not found	VSAM	16	Record not found
		VSAM	24	Record on nonmountable volume
Bit 1	Record length check	VSAM	108	Record length check
Bit 2	Space not found	VSAM	28	Data set cannot be extended
Bit 3	Invalid request	Interface	–	No request parameter list available
		VSAM	20	Exclusive-use conflict
		VSAM	36	No key range defined for insertion
		VSAM	64	Placeholder not available for concurrent data-set positioning
		VSAM	96	Key change attempted
Bit 4	Uncorrectable I/O	VSAM	–	Physical error (register 15 will contain a value of 12)
Bit 5	Unreachable block	VSAM	–	Logical error not covered by any other exception code
Bit 6	Overflow record	Interface	–	Always 1 for a successful READ request
Bit 7	Duplicate record	VSAM	8	Duplicate record
DECBEXC2				
Bits 0-5	Reserved		–	Always 0

Table 49. BISAM Error Conditions (continued)

Byte and Offset	BISAM Meaning	Error Detected By	Request Parameter List Error Code	Interface/VSAM Meaning
Bit 6	Channel program initiated by an asynchronous routine		–	Always 0
Bit 7	Previous macro was READ KU	Interface	–	Previous macro was READ KU

Table 50 gives the contents of registers 0 and 1 when a SYNAD routine specified in a DCB gets control.

Table 50. Register Contents for DCB-Specified ISAM SYNAD Routine

Register	BISAM	QISAM
0	Address of the DECB	0, or, for a sequence check, the address of a field containing the higher key involved in the check
1	Address of the DECB	0

You can also specify a SYNAD routine through the DD AMP parameter (see “JCL for Processing with the ISAM Interface” on page 586). Table 51 gives the contents of registers 0 and 1 when a SYNAD routine specified through AMP gets control.

Table 51. Register Contents for AMP-Specified ISAM SYNAD Routine

Register	BISAM	QISAM
0	Address of the DECB	0, or, for a sequence check, the address of a field containing the higher key involved in the check
1	Address of the DECB	Address of the DCB

If your SYNAD routine issues the SYNADAF macro, registers 0 and 1 are used to communicate. When you issue SYNADAF, register 0 must have the same contents it had when the SYNAD routine got control and register 1 must contain the address of the DCB.

When you get control back from SYNADAF, the registers have the same contents they would have if your program were processing an indexed sequential data set: register 0 contains a completion code, and register 1 contains the address of the SYNADAF message.

The completion codes and the format of a SYNADAF message are given in *z/OS DFSMS Macro Instructions for Data Sets*.

Table 52 shows abend codes issued by the ISAM interface when there is no other method of communicating the error to the user.

Table 52. ABEND Codes Issued by the ISAM Interface

ABEND Code	Error Detected By	DCB/DECB Set By Module/Routine	ABEND Issued By	Error Condition
03B	OPEN	OPEN/OPEN ACB and VALID CHECK	OPEN	Validity check; either (1) access method services and DCB values for LRECL, KEYLE, and RKP do not correspond, (2) DISP=OLD, the DCB was opened for output, and the number of logical records is greater than zero (RELOAD is implied), or (3) OPEN ACB error code 116 was returned for a request to open a VSAM structure.
031	VSAM	SYNAD	SYNAD	SYNAD (ISAM) was not specified and a VSAM physical and logical error occurred.
	VSAM	SCAN/GET and SETL	SYNAD	SYNAD (ISAM) was not specified and an invalid request was found.
	LOAD	LOAD/RESUME	LOAD	SYNAD (ISAM) was not specified and a sequence check occurred.
	LOAD	LOAD	LOAD	SYNAD (ISAM) was not specified and the RDW (record descriptor word) was greater than LRECL.
039	VSAM	SCAN/EODAD	SCAN	End-of-data was found, but there was no EODAD exit.
001	VSAM	SYNAD		I/O error detected.
	BISAM	SYNAD	BISAM	I/O error detected during check.
	BISAM	BISAM	BISAM	Invalid request.

If a SYNAD routine specified through AMP issues the SYNADAF macro, the parameter ACSMETH can specify either QISAM or BISAM, regardless of which of the two is used by your processing program.

A dummy DEB is built by the ISAM interface to support:
- References by the ISAM processing program
- Checkpoint/restart
- ABEND

Table 53 shows the DEB fields that are supported by the ISAM interface. Except as noted, field meanings are the same as in ISAM.

Table 53. DEB Fields Supported by ISAM Interface

DEB Section	Bytes	Fields Supported
PREFIX	16	LNGTH
BASIC	32	TCBAD, OPATB, DEBAD, OFLGS (DISP ONLY), FLGS1 (ISAM-interface bit), AMLNG (104), NMEXT(2), PRIOR, PROTG, DEBID, DCBAD, EXSCL (0-DUMMY DEB), APPAD
ISAM Device	16	EXPTR, FPEAD
Direct Access	16	UCBAD (VSAM UCB)
Access Method	24	WKPT5 (ISAM-interface control block pointer), FREED (pointer to IDAIIFBF)

Converting an Indexed Sequential Data Set

Access method services is used to convert an indexed-sequential data set to a key-sequenced data set. If a master and/or user catalog has been defined, define a key-sequenced data set with the attributes and performance options you want. Then use the access method services REPRO command to convert the indexed-sequential records and load them into the key-sequenced data set. VSAM builds the index for the key-sequenced data set as it loads the data set.

Each volume of a multivolume component must be on the same type of device; the data component and the index component, however, can be on volumes of devices of different types.

When you define the key-sequenced data set into which the indexed sequential data set is to be copied, you must specify the attributes of the VSAM data set for variable and fixed-length records.

For variable-length records:
- VSAM record length equals ISAM DCBLRECL-4.
- VSAM key length equals ISAM DCBKEYLE.
- VSAM key position equals ISAM DCBRKP-4.

For fixed-length records:
- VSAM record length (average and maximum must be the same) equals ISAM DCBLRECL (+ DCBKEYLE, if ISAM DCBRKP is not equal to 0 and records are unblocked).
- VSAM key length equals ISAM DCBKEYLE.
- VSAM key position equals ISAM DCBRKP.

To learn the attributes of the ISAM data set you can use a program such as ISPF/PDF, ISMF, or the IEHLIST utility. IEHLIST can be used to get the dump format of the format 1 DSCB. The layout is in *z/OS DFSMSdfp Advanced Services*. The RKP is at X'5B'.

The level of sharing permitted when the key-sequenced data set is defined should be considered. If the ISAM program opens multiple DCBs pointing to different DD statements for the same data set, a share-options value of 1, which is the default, permits only the first DD statement to be opened. See “Cross-Region Share Options” on page 201 for a description of the cross-region share-options values.

JCL is used to identify data sets and volumes for allocation. Data sets can also be allocated dynamically.

If JCL is used to describe an indexed sequential data set to be converted to VSAM using the access method services REPRO command, include DCB=DSORG=IS. Use a STEPCAT or JOBCAT DD statement as described in “Retrieving an Existing VSAM Data Set” on page 272 to make user catalogs available; you can also use dynamic allocation.

Restriction: Do not use JOBCAT or STEPCAT DD statements for system-managed data sets. The JOBCAT or STEPCAT DD statement will fail if it references a system-managed catalog, or if the data set being searched is system managed. Also, you must connect all referenced catalogs to the system master catalog.

With ISAM, deleted records are flagged as deleted, but are not actually removed from the data set. To avoid reading VSAM records that are flagged as deleted

(X'FF'), code DCB=OPTCD=L. If your program depends on a record's only being flagged and not actually removed, you might want to keep these flagged records when you convert and continue to have your programs process these records. The access method services REPRO command has a parameter (ENVIRONMENT) that causes VSAM to keep the flagged records when you convert.

JCL for Processing with the ISAM Interface

To execute your ISAM processing program to process a key-sequenced data set, replace the ISAM DD card with a VSAM DD card using the ddname that was used for ISAM. The VSAM DD card names the key-sequenced data set and gives any necessary VSAM parameters (through AMP). Specify DISP=MOD for resume loading and DISP=OLD or SHR for all other processing. You do not have to specify anything about the ISAM interface itself. The interface is automatically brought into action when your processing program opens a DCB whose associated DD statement describes a key-sequenced data set (instead of an indexed sequential data set). If you have defined your VSAM data set in a user catalog, specify the user catalog in a JOBCAT or STEPCAT DD statement.

The DCB parameter in the DD statement that identifies a VSAM data set is invalid and must be removed. If the DCB parameter is not removed, unpredictable results can occur. Certain DCB-type information can be specified in the AMP parameter, which is described later in this chapter.

Table 54 shows the DCB fields supported by the ISAM interface.

Table 54. DCB Fields Supported by ISAM Interface

Field Name	Meaning
BFALN	Same as in ISAM; defaults to a doubleword
BLKSI	Set equal to LRECL if not specified
BUFCB	Same as in ISAM
BUFL	The greater value of AMDLRECL or DCBLRECL if not specified
BUFNO	For QISAM, one; for BISAM, the value of STRNO if not specified
DDNAM	Same as in ISAM
DEBAD	During the DCB exit, contains internal system information; after the DCB exit, contains the address of the dummy DEB built by the ISAM interface
DEVT	Set from the VSAM UCB device type code
DSORG	Same as in ISAM
EODAD	Same as in ISAM
ESETL	Address of the ISAM interface ESETL routine (see Table 38 on page 495)
EXCD1	See the QISAM exception codes
EXCD2	See the QISAM exception codes
EXLST	Same as in ISAM (except that VSAM does not support the JFCBE exit)
FREED	Address of the ISAM-interface dynamic buffering routine
GET/PUT	For QISAM LOAD, the address of the ISAM-interface PUT routine; for QISAM SCAN, 0, the address of the ISAM-interface GET routine; 4, the address of the ISAM-interface PUTX routine; and 8, the address of the ISAM-interface RELSE routine
KEYLE	Same as in ISAM
LRAN	Address of the ISAM-interface READ K/WRITE K routine

Table 54. DCB Fields Supported by ISAM Interface (continued)

Field Name	Meaning
LRECL	Set to the maximum record size specified in the access method services DEFINE command if not specified (adjusted for variable-length, fixed, unblocked, and RKP=0 records)
LWKN	Address of the ISAM-interface WRITE KN routine
MACRF	Same as in ISAM
NCP	For BISAM, defaults to one
NCRHI	Set to a value of 8 before DCB exit
OFLGS	Same as in ISAM
OPTCD	Bit 0 (W), same as in ISAM; bit 3 (I), dummy records are not to be written in the VSAM data set; bit 6 (L), VSAM-deleted records (X'FF') are not read; dummy records are to be treated as in ISAM; all other options ignored
RECFM	Same as in ISAM; default to unblocked, variable-length records
RKP	Same as in ISAM
RORG1	Set to a value of 0 after DCB exit
RORG2	Set to a value of X'7FFFF' after DCB exit
RORG3	Set to a value of 0 after DCB exit
SETL	For BISAM, address of the ISAM-interface CHECK routine; for QISAM, address of the ISAM-interface SETL routine
ST	Bit 1 (key-sequence check), same as in ISAM; bit 2 (loading has completed), same as in ISAM
SYNAD	Same as in ISAM
TIOT	Same as in ISAM
WKPT1	For QISAM SCAN, WKPT1 +112=address of the W1CBF field pointing to the current buffer
WKPT5	Address of an internal system control block
WKPT6	For QISAM LOAD, address of the dummy DCB work area vector pointers; the only field supported is ISLVPTRS+4=pointer to KEYSAVE

When an ISAM processing program is run with the ISAM interface, the AMP parameter enables you to specify:

- That a VSAM data set is to be processed (AMORG)
- The need for additional data buffers to improve sequential performance (BUFND)
- The need for extra index buffers for simulating the residency of the highest level(s) of an index in virtual storage (BUFNI)
- Whether to remove records flagged (OPTCD)
- What record format (RECFM) is used by the processing program
- The number of concurrent BISAM and QISAM (basic and queued indexed-sequential access methods) requests that the processing program can issue (STRNO)
- The name of an ISAM user exit routine to analyze physical and logical errors (SYNAD).

For a complete description of the AMP parameter and its syntax, see *z/OS MVS JCL Reference*.

Restrictions on the Use of the ISAM Interface

Some restrictions indicated earlier in this chapter can require you to modify an ISAM processing program to process a key-sequenced data set. All operating system and VSAM restrictions apply to the use of the ISAM interface, including the following restrictions:

- VSAM does not allow the OPEN TYPE=J macro. If your program issues it, remove TYPE=J and the RDJFCB macro.
- VSAM does not allow an empty data set to be opened for input.
- If a GET macro is issued for an empty data set, the resulting messages indicate “no record found (NRF)” rather than “end of data (EOD)”, as it would appear in the noninterface QISAM environment.
- The DUMMY DD statement is not supported for the ISAM interface. An attempt to use the DUMMY DD statement with the ISAM interface will result in a system 03B ABEND. If your program uses a DUMMY DD statement, you might be able to change your JCL to use a temporary or permanent VSAM data set. See “Examples Using JCL to Allocate VSAM Data Sets” on page 269 for examples.

Sharing Restrictions:

- You can share data among subtasks that specify the same DD statement in their DCB(s), and VSAM ensures data integrity. But, if you share data among subtasks that specify different DD statements for the data, you are responsible for data integrity. The ISAM interface does not ensure DCB integrity when two or more DCBs are opened for a data set. All of the fields in a DCB cannot be depended on to contain valid information.
- Processing programs that issue concurrent requests requiring exclusive control can encounter exclusive-use conflicts if the requests are for the same control interval. For more information, see “Chapter 13. Sharing VSAM Data Sets” on page 195.
- When a data set is shared by several jobs (DISP=SHR), you must use the ENQ and DEQ macros to ensure exclusive control of the data set. Exclusive control is necessary to ensure data integrity when your program adds or updates records in the data set. You can share the data set with other users (that is, relinquish exclusive control) when reading records.

Additional restrictions:

- A program must run successfully under ISAM using standard ISAM interfaces; the interface does not check for parameters that are invalid for ISAM.
- VSAM path processing is not supported by the ISAM interface.
- Your ISAM program (on TSO/E) cannot dynamically allocate a VSAM data set (use LOGON PROC).
- CATALOG/DADSM macros in the ISAM processing program must be replaced with access method services commands.
- ISAM programs will run, with sequential processing, if the key length is defined as smaller than it actually is. This is not permitted with the ISAM interface.
- If your ISAM program creates dummy records with a maximum key to avoid overflow, remove that code for VSAM.
- If your program counts overflow records to determine reorganization needs, its results will be meaningless with VSAM data sets.
- For processing programs that use locate processing, the ISAM interface constructs buffers to simulate locate processing.

- For blocked-record processing, the ISAM interface simulates unblocked-record processing by setting the overflow-record indicator for each record. (In ISAM, an overflow record is never blocked with other records.) Programs that examine ISAM internal data areas (for example, block descriptor words (BDW) or the MBBCCHHR address of the next overflow record) must be modified to use only standard ISAM interfaces.The ISAM RELSE instruction causes no action to take place.
- If your DCB exit list contains an entry for a JFCBE exit routine, remove it. The interface does not support the use of a JFCBE exit routine. If the DCB exit list contains an entry for a DCB open exit routine, that exit is taken.
- The work area into which data records are read must not be shorter than a record. If your processing program is designed to read a portion of a record into a work area, you must change the design. The interface takes the record length indicated in the DCB to be the actual length of the data record. The record length in a BISAM DECB is ignored, except when you are replacing a variable-length record with the WRITE macro.
- If your processing program issues the SETL I or SETL ID instruction, you must modify the instruction to some other form of the SETL or remove it. The ISAM interface cannot translate a request that depends on a specific block or device address.
- Although asynchronous processing can be specified in an ISAM processing program, all ISAM requests are handled synchronously by the ISAM interface; WAIT and CHECK requests are always satisfied immediately. The ISAM CHECK macro does not result in a VSAM CHECK macro's being issued but merely causes exception codes in the DECB (data event control block) to be tested.
- If your ISAM SYNAD routine examines information that cannot be supported by the ISAM interface (for example, the IOB), specify a replacement ISAM SYNAD routine in the AMP parameter of the VSAM DD statement.
- The ISAM interface uses the same RPL over and over, thus, for BISAM, a READ for update uses up an RPL until a WRITE or FREEDBUF is issued (when the interface issues an ENDREQ for the RPL). (When using ISAM you can merely issue another READ if you do not want to update a record after issuing a BISAM READ for update.)
- The ISAM interface does not support RELOAD processing. RELOAD processing is implied when an attempt is made to open a VSAM data set for output, specifying DISP=OLD, and, also, the number of logical records in the data set is greater than zero.

Example: Converting a Data Set

In this example, the indexed sequential data set to be converted (ISAMDATA) is cataloged. A key-sequenced data set, VSAMDATA, has previously been defined in user catalog USERCTLG. Because both the indexed-sequential and key-sequenced data set are cataloged, unit and volume information need not be specified.

ISAMDATA contains records flagged for deletion; these records are to be kept in the VSAM data set. The ENVIRONMENT(DUMMY) parameter in the REPRO command tells the system to copy the records flagged for deletion.

```
//CONVERT  JOB  ...
//JOBCAT   DD   DISP=SHR,DSNAME=USERCTLG
//STEP     EXEC PGM=IDCAMS
//SYSPRINT DD   SYSOUT=A
//ISAM     DD   DISP=OLD,DSNAME=ISAMDATA,DCB=DSORG=IS
//VSAM     DD   DISP=OLD,DSNAME=VSAMDATA
//SYSIN    DD   *

       REPRO -
            INFILE(ISAM  ENVIRONMENT(DUMMY)) -
            OUTFILE(VSAM)
/*
```

To drop records flagged for deletion in the indexed-sequential data set, omit ENVIRONMENT(DUMMY).

The use of the JOBCAT DD statement prevents this job from accessing any system-managed data sets.

Example: Issuing a SYNADAF Macro

The following example shows how a SYNAD routine specified through AMP can issue a SYNADAF macro without preliminaries. Registers 0 and 1 already contain what SYNADAF expects to find.

```
AMPSYN   CSECT
         USING     *,15              Register 15 contains the entry
                                     address to AMPSYN.
         SYNADAF   ACSMETH=QISAM     Either QISAM or BISAM can be
                                     specified.
         STM       14,12,12(13)
         BALR      7,0               Load address of next instruction
         USING     *,7

         L         15,132(1)         The address of the DCB is stored
                                     132 bytes into the SYNADAF message.
         L         14,128(1)         The address of the DECB is stored
                                     128 bytes into the SYNADAF message.

         TM        42(15),X'40'      Does the DCB indicate QISAM scan?
         BO        QISAM             Yes.
         TM        43(15),X'40'      Does the DCB indicate QISAM load?
         BO        QISAM             Yes.
BISAM    TM        24(14),X'10'      Does the DECB indicate an invalid
                                     BISAM request?
         BO        INVBISAM          Yes.
```

The routine might print the SYNADAF message or issue ABEND.

```
QISAM    TM        80(15),X'10'      Does the DCB indicate an invalid
                                     QISAM request?
         BO        INVQISAM          Yes.
```

The routine might print the SYNADAF message or issue ABEND.

```
INVBISAM EQU       *
INVQISAM EQU       *
         LM        14,12,12(13)
         DROP      7
         USING     AMPSYN,15

         SYNADRLS
         BR        14
         END       AMPSYN
```

When the processing program closes the data set, the interface issues VSAM PUT macros for ISAM PUT locate requests (in load mode), deletes the interface routines from virtual storage, frees virtual-storage space that was obtained for the interface, and gives control to VSAM.

Appendix F. Converting Character Sets

This appendix covers the following topics.

Topic	Page
Coded Character Sets Sorted by CCSID	593
Coded Character Sets Sorted by Default LOCALNAME	596
CCSID Conversion Groups	603
CCSID Decision Tables	605
Tables for Default Conversion Codes	610

Coded Character Sets Sorted by CCSID

A complete list of coded character sets follows, sorted by the decimal value of the coded character set identifier (CCSID), in the character data representation architecture (CDRA) repository. You can specify and define data sets that use specific coded character sets with ISMF panels. The same information is sorted by the default LOCALNAME element in “Coded Character Sets Sorted by Default LOCALNAME” on page 596.

In the table below, the CCSID Conversion Group column identifies the group, if any, containing the CCSIDs which can be supplied to BSAM or QSAM for ISO/ANSI V4 tapes to convert from or to the CCSID shown in the CCSID column. For a description of CCSID conversion and how you can request it see “Character Data Conversion” on page 299.

See “CCSID Conversion Groups” on page 603 for the conversion groups. A blank in the CCSID Conversion Group column indicates that the CCSID is not supported by BSAM or QSAM for CCSID conversion with ISO/ANSI V4 tapes.

For more information about CCSIDs and LOCALNAMEs, see *Character Data Representation Architecture Reference and Registry*.

CCSID	CCSID Conversion Group	DEFAULT LOCALNAME	CCSID	CCSID Conversion Group	DEFAULT LOCALNAME
37	1	COM EUROPE EBCDIC	4993		JAPAN SB PC-DATA
256		NETHERLAND EBCDIC	5014		URDU EBCDIC
259		SYMBOLS SET 7	5026		JAPAN MIX EBCDIC
273	1	AUS/GERM EBCDIC	5028		JAPAN MIX PC-DATA
277	1	DEN/NORWAY EBCDIC	5029		KOREA MIX EBCDIC
278	1	FIN/SWEDEN EBCDIC	5031		S-CH MIXED EBCDIC
280	1	ITALIAN EBCDIC	5033	1	T-CHINESE EBCDIC
282	1	PORTUGAL EBCDIC	5035		JAPAN MIX EBCDIC
284	1	SPANISH EBCDIC	5045		KOREA KS PC-DATA
285	1	UK EBCDIC	5047		KOREA KS PC-DATA
286		AUS/GER 3270 EBCD	5143	5	LATIN OPEN SYS
290	2	JAPANESE EBCDIC	8229	1	INTL EBCDIC
297	1	FRENCH EBCDIC	8448		INTL EBCDIC
300		JAPAN LATIN HOST	8476		SPAIN EBCDIC
301		JAPAN DB PC-DATA	8489		FRANCE EBCDIC
367	4	US ANSI X3.4 ASCII	8612		ARABIC EBCDIC

© Copyright IBM Corp. 1987, 2002

Converting Character Sets

CCSID	CCSID Conversion Group	DEFAULT LOCALNAME	CCSID	CCSID Conversion Group	DEFAULT LOCALNAME
420		ARABIC EBCDIC	8629		AUS/GERM PC-DATA
421		MAGHR/FREN EBCDIC	8692	4	AUS/GERMAN EBCDIC
423		GREEK EBCDIC	8692	4	AUS/GERMAN EBCDIC
424		HEBREW EBCDIC	9025		KOREA SB EBCDIC
437		USA PC-DATA	9026		KOREA DB EBCDIC
500	4	INTL EBCDIC	9047		CYRILLIC PC-DATA
803		HEBREW EBCDIC	9056		ARABIC PC-DATA
813		GREEK/LATIN ASCII	9060		URDU PC-DATA
819		ISO 8859-1 ASCII	9089		JAPAN PC-DATA SB
833		KOREAN EBCDIC	9122		JAPAN MIX EBCDIC
834		KOREAN DB EBCDIC	9124		JAPAN MIX EBCDIC
835		T-CHINESE DB EBCD	9125		KOREA MIX PC-DATA
836		S-CHINESE EBCDIC	12325		CANADA EBCDIC
837		S-CHINESE EBCDIC	12544		FRANCE EBCDIC
838		THAILAND EBCDIC	12725		FRANCE PC-DATA
839		THAI DB EBCDIC	12788	4	ITALY EBCDIC
850		LATIN-1 PC-DATA	13152		ARABIC PC-DATA
851		GREEK PC-DATA	13218		JAPAN MIX EBCDIC
852		ROECE PC-DATA	13219		JAPAN MIX EBCDIC
853		TURKISH PC-DATA	13221		KOREA MIX EBCDIC
855		CYRILLIC PC-DATA	16421	1	CANADA EBCDIC
856		HEBREW PC-DATA	16821		ITALY PC-DATA
857		TURKISH PC-DATA	16884	4	FIN/SWEDEN EBCDIC
860		PORTUGESE PC-DATA	20517		PORTUGAL EBCDIC
861		ICELAND PC-DATA	20917		UK PC-DATA
862		HEBREW PC-DATA	20980	4	DEN/NORWAY EBCDIC
863		CANADA PC-DATA	24613	1	INTL EBCDIC
864		ARABIC PC-DATA	24877		JAPAN DB PC-DISPL
865		DEN/NORWAY PC-DAT	25013		USA PC-DISPLAY
866		CYRILLIC PC-DATA	25076	4	DEN/NORWAY EBCDIC
868		URDU PC-DATA	25426		LATIN-1 PC-DISP
869		GREEK PC-DATA	25427		GREECE PC-DISPLAY
870		ROECE EBCDIC	25428		LATIN-2 PC-DISP
871	1	ICELAND EBCDIC	25429		TURKEY PC-DISPLAY
874		THAI PC-DISPLAY	25431		CYRILLIC PC-DISP
875	3	GREEK EBCDIC	25432		HEBREW PC-DISPLAY
880		CYRILLIC EBCDIC	25433		TURKEY PC-DISPLAY
891		KOREA SB PC-DATA	25436		PORTUGAL PC-DISP
895		JAPAN 7-BIT LATIN	25437		ICELAND PC-DISP
896		JAPAN 7-BIT KATAK	25438		HEBREW PC-DISPLAY
897		JAPAN SB PC-DATA	25439		CANADA PC-DISPLAY
899		SYMBOLS - PC	25440		ARABIC PC-DISPLAY
903		S-CHINESE PC-DATA	25441		DEN/NOR PC-DISP
904		T-CHINESE PC-DATA	25442		CYRILLIC PC-DISP
905		TURKEY EBCDIC	25444		URDU PC-DISPLAY
912		ISO 8859-2 ASCII	25445		GREECE PC-DISPLAY
915		ISO 8859-5 ASCII	25450		THAILAND PC-DISP
916		ISO 8859-8 ASCII	25467		KOREA SB PC-DISP
918		URDU EBCDIC	25473		JAPAN SB PC-DISP
920		ISO 8859-9 ASCII	25479		S-CHIN SB PC-DISP
926		KOREA DB PC-DATA	25480		T-CHINESE PC-DISP
927		T-CHINESE PC-DATA	25502		KOREA DB PC-DISP

CCSID	CCSID Conversion Group	DEFAULT LOCALNAME	CCSID	CCSID Conversion Group	DEFAULT LOCALNAME
928		S-CHINESE PC-DATA	25503		T-CHINESE PC-DISP
929		THAI DB PC-DATA	25504		S-CHINESE PC-DISP
930		JAPAN MIX EBCDIC	25505		THAILAND PC-DISP
931		JAPAN MIX EBCDIC	25508		JAPAN PC-DISPLAY
932		JAPAN MIX PC-DATA	25510		KOREA PC-DISPLAY
933		KOREA MIX EBCDIC	25512		S-CHINESE PC-DISP
934		KOREA MIX PC-DATA	25514		T-CHINESE PC-DISP
935		S-CHINESE MIX EBC	25518		JAPAN PC-DISPLAY
936		S-CHINESE PC-DATA	25520		KOREA PC-DISPLAY
937	1	T-CHINESE MIX EBC	25522		S-CHINESE PC-DISP
938		T-CHINESE MIX PC	25524		T-CHINESE PC-DISP
939		JAPAN MIX EBCDIC	25525		KOREA KS PC-DISP
942		JAPAN MIX PC-DATA	25527		KOREA KS PC-DISP
944		KOREA MIX PC-DATA	25616		KOREA SB PC-DISP
946		S-CHINESE PC-DATA	25617		JAPAN PC-DISPLAY
948		T-CHINESE PC-DATA	25618		S-CHINESE PC-DISP
949		KOREA KS PC-DATA	25619		T-CHINESE PC-DISP
951		IBM KS PC-DATA	25664		KOREA KS PC-DISP
1008		ARABIC ISO/ASCII	28709	1	T-CHINESE EBCDIC
1010		FRENCH ISO-7 ASCI	29109		USA PC-DISPLAY
1011		GERM ISO-7 ASCII	29172	4	BRAZIL EBCDIC
1012		ITALY ISO-7 ASCII	29522		LATIN-1 PC-DISP
1013		UK ISO-7 ASCII	29523		GREECE PC-DISPLAY
1014		SPAIN ISO-7 ASCII	29524		ROECE PC-DISPLAY
1015		PORTUGAL ISO7 ASC	29525		TURKEY PC-DISPLAY
1016		NOR ISO-7 ASCII	29527		CYRILLIC PC-DISP
1017		DENMK ISO-7 ASCII	29528		HEBREW PC-DISPLAY
1018		FIN/SWE ISO-7 ASC	29529		TURKEY PC-DISPLAY
1019		BELG/NETH ASCII	29532		PORTUGAL PC-DISP
1020		CANADA ISO-7	29533		ICELAND PC-DISP
1021		SWISS ISO-7	29534		HEBREW PC-DISPLAY
1023		SPAIN ISO-7	29535		CANADA PC-DISPLAY
1025		CYRILLIC EBCDIC	29536		ARABIC PC-DISPLAY
1026	3	TURKEY LATIN-5 EB	29537		DEN/NOR PC-DISP
1027	2	JAPAN LATIN EBCD	29540		URDU PC-DISPLAY
1040		KOREA PC-DATA	29541		GREECE PC-DISPLAY
1041		JAPAN PC-DATA	29546		THAILAND PC-DISP
1042		S-CHINESE PC-DATA	29614		JAPAN PC-DISPLAY
1043		T-CHINESE PC-DATA	29616		KOREA PC-DISPLAY
1046		ARABIC - PC	29618		S-CHINESE PC-DISP
1047	5	LATIN OPEN SYS EB	29620		T-CHINESE PC-DISP
1051		HP EMULATION	29621		KOREA KS MIX PC
1088		KOREA KS PC-DATA	29623		KOREA KS PC-DISP
1089		ARABIC ISO 8859-6	29712		KOREA PC-DISPLAY
1097		FARSI EBCDIC	29713		JAPAN PC-DISPLAY
1098		FARSI - PC	29714		S-CHINESE PC-DISP
1100		MULTI EMULATION	29715		T-CHINESE PC-DISP
1101		BRITISH ISO-7 NRC	29760		KOREA KS PC-DISP
1102		DUTCH ISO-7 NRC	32805	1	JAPAN LATIN EBCDIC
1103		FINNISH ISO-7 NRC	33058	2	JAPAN EBCDIC
1104		FRENCH ISO-7 NRC	33205		SWISS PC-DISPLAY
1105		NOR/DAN ISO-7 NRC	33268	4	UK/PORTUGAL EBCDIC

Converting Character Sets

CCSID	CCSID Conversion Group	DEFAULT LOCALNAME	CCSID	CCSID Conversion Group	DEFAULT LOCALNAME
1106		SWEDISH ISO-7 NRC	33618		LATIN-1 PC-DISP
1107		NOR/DAN ISO-7 NRC	33619		GREECE PC-DISPLAY
4133	1	USA EBCDIC	33620		ROECE PC-DISPLAY
4369	1	AUS/GERMAN EBCDIC	33621		TURKEY PC-DISPLAY
4370	1	BELGIUM EBCDIC	33623		CYRILLIC PC-DISP
4371	1	BRAZIL EBCDIC	33624		HEBREW PC-DISPLAY
4372		CANADA EBCDIC	33632		ARABIC PC-DISPLAY
4373	1	DEN/NORWAY EBCDIC	33636		URDU PC-DISPLAY
4374	1	FIN/SWEDEN EBCDIC	33637		GREECE PC-DISPLAY
4376	1	ITALY EBCDIC	33665		JAPAN PC-DISPLAY
4378	1	PORTUGAL EBCDIC	33698		JAPAN KAT/KAN EBC
4380	1	LATIN EBCDIC	33699		JAPAN LAT/KAN EBC
4381	1	UK EBCDIC	33700		JAPAN PC-DISPLAY
4386	2	JAPAN EBCDIC SB	33717		KOREA KS PC-DISP
4393	1	FRANCE EBCDIC	37301		AUS/GERM PC-DISP
4396		JAPAN EBCDIC DB	37364		BELGIUM EBCDIC
4516		ARABIC EBCDIC	37719		CYRILLIC PC-DISP
4519		GREEK EBCDIC 3174	37728		ARABIC PC-DISPLAY
4520		HEBREW EBCDIC	37732		URDU PC-DISPLAY
4533		SWISS PC-DATA	37761		JAPAN SB PC-DISP
4596	4	LATIN AMER EBCDIC	37796		JAPAN PC-DISPLAY
4929		KOREA SB EBCDIC	37813		KOREA KS PC-DISP
4932		S-CHIN SB EBCDIC	41397		FRANCE PC-DISPLAY
4934		THAI SB EBCDIC	41460	4	SWISS EBCDIC
4946		LATIN-1 PC-DATA	41824		ARABIC PC-DISPLAY
4947		GREEK PC-DATA	41828		URDU PC-DISPLAY
4948		LATIN-2 PC-DATA	45493		ITALY PC-DISPLAY
4949		TURKEY PC-DATA	45556	4	SWISS EBCDIC
4951		CYRILLIC PC-DATA	45920		ARABIC PC-DISPLAY
4952		HEBREW PC-DATA	49589		UK PC-DISPLAY
4953		TURKEY PC-DATA	49652	4	BELGIUM EBCDIC
4960		ARABIC PC-DATA	59748	4	INTL EBCDIC
4964		URDU PC-DATA	61696	4	GLOBAL SB EBCDIC
4965		GREEK PC-DATA	61697		GLOBAL SB PC-DATA
4966		ROECE LATIN-2 EBC	61698		GLOBAL PC-DISPLAY
4967	1	ICELAND EBCDIC	61699		GLBL ISO-8 ASCII
4970		THAI SB PC-DATA	61700		GLBL ISO-7 ASCII
4976		CYRILLIC EBCDIC	61710		GLOBAL USE ASCII
			61711	4	GLOBAL USE EBCDIC
			61712	4	GLOBAL USE EBCDIC

Coded Character Sets Sorted by Default LOCALNAME

DEFAULT LOCALNAME	CCSID	CCSID Conversion Group
ARABIC - PC	1046	
ARABIC EBCDIC	420	
ARABIC EBCDIC	4516	
ARABIC EBCDIC	8612	
ARABIC ISO 8859-6	1089	
ARABIC ISO/ASCII	1008	

DEFAULT LOCALNAME	CCSID	CCSID Conversion Group
ARABIC PC-DATA	864	
ARABIC PC-DATA	4960	
ARABIC PC-DATA	9056	
ARABIC PC-DATA	13152	
ARABIC PC-DISPLAY	25440	
ARABIC PC-DISPLAY	29536	
ARABIC PC-DISPLAY	33632	
ARABIC PC-DISPLAY	37728	
ARABIC PC-DISPLAY	41824	
ARABIC PC-DISPLAY	45920	
AUS/GER 3270 EBCD	286	
AUS/GERM EBCDIC	273	1
AUS/GERM PC-DATA	8629	
AUS/GERM PC-DISP	37301	
AUS/GERMAN EBCDIC	4369	1
AUS/GERMAN EBCDIC	8692	4
BELG/NETH ASCII	1019	
BELGIUM EBCDIC	4370	1
BELGIUM EBCDIC	37364	
BELGIUM EBCDIC	49652	4
BRAZIL EBCDIC	4371	1
BRAZIL EBCDIC	29172	4
BRITISH ISO-7 NRC	1101	
CANADA EBCDIC	4372	
CANADA EBCDIC	12325	
CANADA EBCDIC	16421	1
CANADA ISO-7	1020	
CANADA PC-DATA	863	
CANADA PC-DISPLAY	25439	
CANADA PC-DISPLAY	29535	
COM EUROPE EBCDIC	37	1
CYRILLIC EBCDIC	880	
CYRILLIC EBCDIC	1025	
CYRILLIC EBCDIC	4976	
CYRILLIC PC-DATA	855	
CYRILLIC PC-DATA	866	
CYRILLIC PC-DATA	4951	
CYRILLIC PC-DATA	9047	
CYRILLIC PC-DISP	25431	
CYRILLIC PC-DISP	25442	
CYRILLIC PC-DISP	29527	
CYRILLIC PC-DISP	33623	
CYRILLIC PC-DISP	37719	
DEN/NOR PC-DISP	25441	
DEN/NOR PC-DISP	29537	
DEN/NORWAY EBCDIC	277	1
DEN/NORWAY EBCDIC	4373	1
DEN/NORWAY EBCDIC	20980	4
DEN/NORWAY EBCDIC	25076	4
DEN/NORWAY PC-DAT	865	
DENMK ISO-7 ASCII	1017	

DEFAULT LOCALNAME	CCSID	CCSID Conversion Group
DUTCH ISO-7 NRC	1102	
FARSI - PC	1098	
FARSI EBCDIC	1097	
FIN/SWE ISO-7 ASC	1018	
FIN/SWEDEN EBCDIC	278	1
FIN/SWEDEN EBCDIC	4374	1
FIN/SWEDEN EBCDIC	16884	4
FINNISH ISO-7 NRC	1103	
FRANCE EBCDIC	4393	1
FRANCE EBCDIC	8489	
FRANCE EBCDIC	12544	
FRANCE PC-DATA	12725	
FRANCE PC-DISPLAY	41397	
FRENCH EBCDIC	297	1
FRENCH ISO-7 ASCI	1010	
FRENCH ISO-7 NRC	1104	
GERM ISO-7 ASCII	1011	
GLBL ISO-7 ASCII	61700	
GLBL ISO-8 ASCII	61699	
GLOBAL PC-DISPLAY	61698	
GLOBAL SB EBCDIC	61696	4
GLOBAL SB PC-DATA	61697	
GLOBAL USE ASCII	61710	
GLOBAL USE EBCDIC	61711	4
GLOBAL USE EBCDIC	61712	4
GREECE PC-DISPLAY	25427	
GREECE PC-DISPLAY	25445	
GREECE PC-DISPLAY	29523	
GREECE PC-DISPLAY	29541	
GREECE PC-DISPLAY	33619	
GREECE PC-DISPLAY	33637	
GREEK EBCDIC	423	
GREEK EBCDIC	875	3
GREEK EBCDIC 3174	4519	
GREEK PC-DATA	851	
GREEK PC-DATA	869	
GREEK PC-DATA	4947	
GREEK PC-DATA	4965	
GREEK/LATIN ASCII	813	
HEBREW EBCDIC	424	
HEBREW EBCDIC	803	
HEBREW EBCDIC	4520	
HEBREW PC-DATA	856	
HEBREW PC-DATA	862	
HEBREW PC-DATA	4952	
HEBREW PC-DISPLAY	25432	
HEBREW PC-DISPLAY	25438	
HEBREW PC-DISPLAY	29528	
HEBREW PC-DISPLAY	29534	
HEBREW PC-DISPLAY	33624	
HP EMULATION	1051	

DEFAULT LOCALNAME	CCSID	CCSID Conversion Group
IBM KS PC-DATA	951	
ICELAND EBCDIC	871	1
ICELAND EBCDIC	4967	1
ICELAND PC-DATA	861	
ICELAND PC-DISP	25437	
ICELAND PC-DISP	29533	
INTL EBCDIC	500	4
INTL EBCDIC	8229	1
INTL EBCDIC	8448	
INTL EBCDIC	24613	1
INTL EBCDIC	53748	4
ISO 8859-1 ASCII	819	
ISO 8859-2 ASCII	912	
ISO 8859-5 ASCII	915	
ISO 8859-8 ASCII	916	
ISO 8859-9 ASCII	920	
ITALIAN EBCDIC	280	1
ITALY EBCDIC	4376	1
ITALY EBCDIC	12788	4
ITALY ISO-7 ASCII	1012	
ITALY PC-DATA	16821	
ITALY PC-DISPLAY	45493	
JAPAN DB PC-DATA	300	
JAPAN DB PC-DATA	301	
JAPAN DB PC-DISPL	24877	
JAPAN EBCDIC	33058	2
JAPAN EBCDIC DB	4396	
JAPAN EBCDIC SB	4386	2
JAPAN KAT/KAN EBC	33698	
JAPAN LAT/KAN EBC	33699	
JAPAN LATIN EBCD	1027	2
JAPAN LATIN EBCDIC	32805	1
JAPAN MIX EBCDIC	930	
JAPAN MIX EBCDIC	931	
JAPAN MIX EBCDIC	939	
JAPAN MIX EBCDIC	5026	
JAPAN MIX EBCDIC	5035	
JAPAN MIX EBCDIC	9122	
JAPAN MIX EBCDIC	13218	
JAPAN MIX EBCDIC	13219	
JAPAN MIX PC-DATA	932	
JAPAN MIX PC-DATA	942	
JAPAN MIX PC-DATA	5028	
JAPAN MIX PC-DATA	9124	
JAPAN PC-DATA	1041	
JAPAN PC-DATA SB	9089	
JAPAN PC-DISPLAY	25508	
JAPAN PC-DISPLAY	25518	
JAPAN PC-DISPLAY	25617	
JAPAN PC-DISPLAY	29614	
JAPAN PC-DISPLAY	29713	

DEFAULT LOCALNAME	CCSID	CCSID Conversion Group
JAPAN PC-DISPLAY	33665	
JAPAN PC-DISPLAY	33700	
JAPAN PC-DISPLAY	37796	
JAPAN SB PC-DATA	897	
JAPAN SB PC-DATA	4993	
JAPAN SB PC-DISP	25473	
JAPAN SB PC-DISP	37761	
JAPAN 7-BIT KATAK	896	
JAPAN 7-BIT LATIN	895	
JAPANESE EBCDIC	290	2
KOREA DB EBCDIC	9026	
KOREA DB PC-DATA	926	
KOREA DB PC-DISP	25502	
KOREA KS MIX PC	29621	
KOREA KS PC DATA	5047	
KOREA KS PC-DATA	949	
KOREA KS PC-DATA	1088	
KOREA KS PC-DATA	5045	
KOREA KS PC-DISP	25525	
KOREA KS PC-DISP	25527	
KOREA KS PC-DISP	25664	
KOREA KS PC-DISP	29623	
KOREA KS PC-DISP	29760	
KOREA KS PC-DISP	33717	
KOREA KS PC-DISP	37813	
KOREA MIX EBCDIC	933	
KOREA MIX EBCDIC	5029	
KOREA MIX EBCDIC	9125	
KOREA MIX EBCDIC	13221	
KOREA MIX PC-DATA	934	
KOREA MIX PC-DATA	944	
KOREA PC-DATA	1040	
KOREA PC-DISPLAY	25510	
KOREA PC-DISPLAY	25520	
KOREA PC-DISPLAY	29616	
KOREA PC-DISPLAY	29712	
KOREA SB EBCDIC	4929	
KOREA SB EBCDIC	9025	
KOREA SB PC-DATA	891	
KOREA SB PC-DISP	25467	
KOREA SB PC-DISP	25616	
KOREAN DB EBCDIC	834	
KOREAN EBCDIC	833	
LATIN AMER EBCDIC	4596	4
LATIN EBCDIC	4380	1
LATIN OPEN SYS	5143	5
LATIN OPEN SYS EB	1047	5
LATIN-1 PC-DATA	850	
LATIN-1 PC-DATA	4946	
LATIN-1 PC-DISP	25426	
LATIN-1 PC-DISP	29522	

DEFAULT LOCALNAME	CCSID	CCSID Conversion Group
LATIN-1 PC-DISP	33618	
LATIN-2 PC-DATA	4948	
LATIN-2 PC-DISP	25428	
MAGHR/FREN EBCDIC	421	
MULTI EMULATION	1100	
NETHERLAND EBCDIC	256	
NOR ISO-7 ASCII	1016	
NOR/DAN ISO-7 NRC	1105	
NOR/DAN ISO-7 NRC	1107	
PORTUGAL EBCDIC	282	1
PORTUGAL EBCDIC	4378	1
PORTUGAL EBCDIC	20517	1
PORTUGAL ISO7 ASC	1015	
PORTUGAL PC-DISP	25436	
PORTUGAL PC-DISP	29532	
PORTUGESE PC-DATA	860	
ROECE EBCDIC	870	
ROECE LATIN-2 EBC	4966	
ROECE PC-DATA	852	
ROECE PC-DISPLAY	29524	
ROECE PC-DISPLAY	33620	
S-CH MIXED EBCDIC	5031	
S-CHIN SB EBCDIC	4932	
S-CHIN SB PC-DISP	25479	
S-CHINESE EBCDIC	836	
S-CHINESE EBCDIC	837	
S-CHINESE MIX EBC	935	
S-CHINESE PC-DATA	903	
S-CHINESE PC-DATA	928	
S-CHINESE PC-DATA	936	
S-CHINESE PC-DATA	946	
S-CHINESE PC-DATA	1042	
S-CHINESE PC-DISP	25504	
S-CHINESE PC-DISP	25512	
S-CHINESE PC-DISP	25522	
S-CHINESE PC-DISP	25618	
S-CHINESE PC-DISP	29618	
S-CHINESE PC-DISP	29714	
SPAIN EBCDIC	8476	
SPAIN ISO-7	1023	
SPAIN ISO-7 ASCII	1014	
SPANISH EBCDIC	284	1
SWEDISH ISO-7 NRC	1106	
SWISS EBCDIC	41460	4
SWISS EBCDIC	45556	4
SWISS ISO-7	1021	
SWISS PC-DATA	4533	
SWISS PC-DISPLAY	33205	
SYMBOLS - PC	899	
SYMBOLS SET 7	259	
T-CHINESE DB EBCD	835	

DEFAULT LOCALNAME	CCSID	CCSID Conversion Group
T-CHINESE EBCDIC	5033	1
T-CHINESE EBCDIC	28709	1
T-CHINESE MIX EBC	937	1
T-CHINESE MIX PC	938	
T-CHINESE PC-DATA	904	
T-CHINESE PC-DATA	927	
T-CHINESE PC-DATA	948	
T-CHINESE PC-DATA	1043	
T-CHINESE PC-DISP	25480	
T-CHINESE PC-DISP	25503	
T-CHINESE PC-DISP	25514	
T-CHINESE PC-DISP	25524	
T-CHINESE PC-DISP	25619	
T-CHINESE PC-DISP	29620	
T-CHINESE PC-DISP	29715	
THAI DB EBCDIC	839	
THAI DB PC-DATA	929	
THAI PC-DISPLAY	874	
THAI SB EBCDIC	4934	
THAI SB PC-DATA	4970	
THAILAND EBCDIC	838	
THAILAND PC-DISP	25450	
THAILAND PC-DISP	25505	
THAILAND PC-DISP	29546	
TURKEY EBCDIC	905	
TURKEY LATIN-5 EB	1026	3
TURKEY PC-DATA	4949	
TURKEY PC-DATA	4953	
TURKEY PC-DISPLAY	25429	
TURKEY PC-DISPLAY	25433	
TURKEY PC-DISPLAY	29525	
TURKEY PC-DISPLAY	29529	
TURKEY PC-DISPLAY	33621	
TURKISH PC-DATA	853	
TURKISH PC-DATA	857	
UK EBCDIC	285	1
UK EBCDIC	4381	1
UK ISO-7 ASCII	1013	
UK PC-DATA	20917	
UK PC-DISPLAY	49589	
UK/PORTUGAL EBCDC	33268	4
URDU EBCDIC	918	
URDU EBCDIC	5014	
URDU PC-DATA	868	
URDU PC-DATA	4964	
URDU PC-DATA	9060	
URDU PC-DISPLAY	25444	
URDU PC-DISPLAY	29540	
URDU PC-DISPLAY	33636	
URDU PC-DISPLAY	37732	
URDU PC-DISPLAY	41828	

DEFAULT LOCALNAME	CCSID	CCSID Conversion Group
US ANSI X3.4 ASCI	367	4
USA EBCDIC	4133	1
USA PC-DATA	437	
USA PC-DISPLAY	25013	
USA PC-DISPLAY	29109	

CCSID Conversion Groups

The following figures list the CCSID conversion groups. Figure 130 lists conversion group 1, Figure 131 lists conversion group 2, Figure 132 lists conversion group 3, Figure 133 on page 604 lists conversion group 4, and Figure 134 on page 604 lists conversion group 5.

```
  500              45556
 4596              49652
 8692              53748
12788              61696
16884              61711
20980              61712
25076               1047
29172               5143
33268                367
41460
```

Figure 130. CCSID Conversion Group 1

```
  500              53748
 4596              61696
 8692              61711
12788              61712
16884               1047
20980               5143
25076                367
29172               1027
33268                290
41460               4386
45556              33058
49652
```

Figure 131. CCSID Conversion Group 2

```
 1047
 5143
```

Figure 132. CCSID Conversion Group 3

37	280	5143
4133	4376	367
8229	282	500
16421	4378	4596
20517	284	8692
24613	4380	12788
28709	285	16884
32805	4381	20980
937	290	25076
5033	4386	29172
273	33058	33268
4369	297	41460
4370	4393	45556
275	871	49652
4371	4967	53748
277	875	61696
4373	1026	61711
278	1027	61712
4374	1047	

Figure 133. CCSID Conversion Group 4

37	280	4596
4133	4376	8692
8229	282	12788
16421	4378	16884
20517	284	20980
24613	4380	25076
28709	285	29172
32805	4381	33268
937	290	41460
5033	4386	45556
273	33058	49652
4369	297	53748
4370	4393	61696
275	871	61711
4371	4967	61712
277	875	875
4373	1026	1026
278	1027	367
4374	500	

Figure 134. CCSID Conversion Group 5

CCSID Decision Tables

The following three tables are used by data management to determine the type of data conversion to perform for ISO/ANSI tapes. See “Character Data Conversion” on page 299 for a description of data conversion and how it is requested.

For purposes of these tables:

USER

Refers to the CCSID which describes the data used by the application program. It is derived from (in order of precedence):

1. CCSID parameter on the EXEC statement, or
2. CCSID parameter on the JOB statement.

If the table entry contains a 0, it means that the CCSID parameter was not supplied in the JCL. In this case, if data management performs CCSID conversion, a system default CCSID of 500 is used.

TAPE (DD)

Refers to the CCSID which is specified by the CCSID parameter on the DD statement, dynamic allocation, or TSO ALLOCATE. If the table entry contains a 0, it means that the CCSID parameter was not supplied. In this case, if data management performs CCSID conversion, a system default CCSID of 367 is used when open for output and not DISP=MOD.

Label

Refers to the CCSID which will be stored in the tape label during output processing (not DISP=MOD), or which is found in an existing label during DISP=MOD or input processing. Unless otherwise indicated in the tables, a CCSID found in the tape label overrides a CCSID specified on the DD statement on input.

Conversion

Refers to the type of data conversion data management will perform based on the combination of CCSIDs supplied from the previous three columns.

- **Default** denotes that data management performs conversion using Default Character conversion as described in “Character Data Conversion” on page 299. This conversion is used when CCSIDs are not supplied by any source.

  An existing data set created using Default Character conversion cannot be read or written (unless DISP=OLD) using CCSIDs.
- **Convert a->b** denotes that data management performs CCSID conversion with *a* and *b* representing the CCSIDs used.
- **No conversion** denotes that data management performs no conversion on the data.
- **Fail** denotes that the combination of CCSIDs is invalid. This results in an ABEND513-14 during open.

A CCSID of X, Y, or Z is used to represent any of the supported CCSIDs (other than 65535) where X ≠ Y ≠ Z. Unless otherwise specified in the tables, a CCSID of 65535 indicates that data management will perform no conversion.

Converting Character Sets

Table 55 describes processing used when the data set is opened for output (not EXTEND) with DISP=NEW or DISP=OLD. The label column indicates what will be stored in the label.

Table 55. Output DISP=NEW,OLD

USER	TAPE(DD)	Label	Conversion	Comments
0	0	BLANK	Default	No CCSIDs specified. Use Default Character Conversion.
0	Y	Y	Convert 500->Y	USER default is 500.
0	65535	65535	No conversion	No convert specified on DD. CCSID of data is unknown.
X	0	367	Convert X->367	Default tape is 367.
X	Y	Y	Convert X->Y	USER is X. DD is Y.
X	65535	X	No conversion	User data assumed to be X.
65535	0	65535	No conversion	No convert specified on JOB/EXEC. CCSID of data unknown.
65535	Y	Y	No conversion	User data assumed to be Y.
65535	65535	65535	No conversion	No convert specified.

Table 56 describes processing used when the data set is opened for output with DISP=MOD or when the OPEN option is EXTEND. This is only allowed for IBM created Version 4 tapes. Attempting to open a non-IBM created Version 4 tape with DISP=MOD will result in ABEND513-10.

Table 56. Output DISP=MOD (IBM V4 tapes only)

USER	TAPE(DD)	Label	Conversion	Comments
0	0	BLANK	Default	No CCSIDs specified. Use Default Character Conversion.
0	0	Z	Convert 500->Z	USER default is 500.
0	0	65535	Fail	CCSID of tape data is unknown. Prevent mixed user data.
0	Y	BLANK	Fail	Blank in label means Default Character Conversion but Y specified. CCSID mismatch.
0	Y	Z	Fail	CCSID mismatch. Label says Z but DD says Y.
0	Y	Y	Convert 500->Y	USER default is 500. Label says Y and DD says Y.
0	Y	65535	Fail	DD says Y but CCSID of data is unknown. CCSID mismatch.
0	65535	BLANK	Fail	Blank in label means Default Character Conversion but unknown CCSID on tape.
0	65535	Z	Fail	DD says no convert. Label says Z and USER CCSID not specified.
0	65535	65535	No conversion	No convert specified. User must ensure data is in correct CCSID.
X	0	BLANK	Fail	Blank in label means Default Character Conversion but USER CCSID is X. No interface to convert X to 7-bit ASCII.
X	0	Z	Convert X->Z	USER is X. Label is Z.
X	0	65535	Fail	Label CCSID is unknown, but USER is X with no convert specified. Potential mismatch.
X	Y	BLANK	Fail	Blank in label means Default Character Conversion but DD says Y. CCSID mismatch.

Table 56. Output DISP=MOD (IBM V4 tapes only) (continued)

USER	TAPE(DD)	Label	Conversion	Comments
X	Y	Z	Fail	DD says Y but label says Z. CCSID mismatch.
X	Y	Y	Convert X->Y	DD says Y and label says Y.
X	Y	65535	Fail	DD says Y but tape CCSID is unknown. Possible mismatch.
X	65535	BLANK	Fail	Blank in label means Default Character Conversion but USER says X with no convert. CCSID mismatch.
X	65535	Z	Fail	USER and label CCSID mismatch with no convert specified.
X	65535	X	No conversion	USER and label CCSID agree with no convert specified.
X	65535	65535	No conversion	DD and label agree but data on tape must be X as well or it will cause problems later.
65535	0	BLANK	Fail	Blank in label means Default Character Conversion but USER CCSID specified. CCSID mismatch.
65535	0	Z	No conversion	No convert specified. Must assume tape data is Z or it will cause problems later.
65535	0	65535	No conversion	No convert specified and label says no convert.
65535	Y	BLANK	Fail	Blank in label means Default Character Conversion but DD says Y. CCSID mismatch.
65535	Y	Z	Fail	DD say Y but label says Z. CCSID mismatch even though no convert is specified.
65535	Y	Y	No conversion	TAPE and label agree and no conversion specified.
65535	Y	65535	No conversion	Label is unknown and USER specified no convert. Assume data is correct. Otherwise, problems later on.
65535	65535	BLANK	Fail	Blank in label means Default Character Conversion. Even though no convert specified, still possible mismatch.
65535	65535	Z	No conversion	No convert specified but data must be in Z or it will cause problems later.
65535	65535	65535	No conversion	No convert specified.

Converting Character Sets

Table 57 describes processing used when the data set is opened for INPUT or RDBACK.

Table 57. Input

USER	TAPE(DD)	Label	Conversion	Comments
0	0	BLANK	Default	No CCSIDs specified. Assume Default Character Conversion.
0	0	Z	Convert Z->500	USER default is 500. Label says Z.
0	0	65535	No conversion	Label says no convert and no CCSIDs specified.
0	Y	BLANK	Fail	Fail if IBM V4 tape because blank in label means Default Character Conversion but DD says Y.
0	Y	BLANK	Convert Y->500	Allow if not IBM V4 tape because user is indicating data on tape is Y via the DD.
0	Y	Z	Fail	Label say Z but DD says Y. CCSID mismatch.
0	Y	Y	Convert Y->500	USER default is 500. DD says Y and label says Y.
0	Y	65535	Convert Y->500	DD is saying tape data is Y. USER default is 500.
0	65535	BLANK	No conversion	DD specified no conversion.
0	65535	Z	No conversion	DD specified no conversion.
0	65535	65535	No conversion	DD specified no conversion.
X	0	BLANK	Fail	Blank in label means Default Character Conversion but USER specified CCSID. CCSID mismatch.
X	0	Z	Convert Z->X	USER is X. Label is Z.
X	0	65535	No conversion	Label says no conversion and no CCSID specified on DD, therefore, no conversion.
X	Y	BLANK	Fail	Fail if IBM V4 tape because blank in label means Default Character Conversion but DD says Y. CCSID mismatch.
X	Y	BLANK	Convert Y->X	Allow if not IBM V4 tape because DD is indicating data is Y. USER is X.
X	Y	Z	Fail	Label says Z but DD says Y. CCSID mismatch.
X	Y	Y	Convert Y->X	Label and DD both specify Y. USER is X.
X	Y	65535	Convert Y->X	Label CCSID is unknown but DD says Y. USER is X. Assume data is Y.
X	65535	BLANK	Fail	Fail if IBM V4 tape because blank in label means Default Character Conversion but USER says X. CCSID mismatch.
X	65535	BLANK	No conversion	Allow if not IBM V4 tape because DD specified no convert.
X	65535	Z	Fail	Label says Z, USER says X but DD says no convert. CCSID mismatch between USER and label.
X	65535	X	No conversion	Label says X and USER says X, therefore, allow no conversion.
X	65535	65535	No conversion	No conversion specified, but tape data must be X.
65535	0	BLANK	No conversion	USER specified no conversion indicating that application can accept any data including 7-bit ASCII.
65535	0	Z	No conversion	USER specified no conversion indicating that application can accept any data including Z.

Table 57. Input (continued)

USER	TAPE(DD)	Label	Conversion	Comments
65535	0	65535	No conversion	USER specified no conversion indicating that application can accept any data including unknown data on tape.
65535	Y	BLANK	Fail	Fail if IBM V4 tape because blank means Default Character Conversion but DD says Y. CCSID mismatch.
65535	Y	BLANK	No conversion	Allow if not IBM V4 tape because DD is indicating tape is Y with no conversion specified.
65535	Y	Z	Fail	Label says Z but DD says Y. CCSID mismatch.
65535	Y	Y	No conversion	Label and DD agree, therefore, no convert.
65535	Y	65535	No conversion	Label data is unknown and no convert specified.
65535	65535	BLANK	No conversion	USER specified no convert so application can accept tape data in any format including 7-bit ASCII.
65535	65535	Z	No conversion	USER specified no convert so application can accept tape data in any format including Z.
65535	65535	65535	No conversion	No convert specified.

Tables for Default Conversion Codes

The following tables are used by data management when performing default character conversion as described in “Character Data Conversion” on page 299. They are also used by the system when you issue the XLATE macro instruction.

When converting EBCDIC code to ASCII code, all EBCDIC code not having an ASCII equivalent is converted to X'1A'. When converting ASCII code to EBCDIC code, all ASCII code not having an EBCDIC equivalent is converted to X'3F'. Because Version 3 ASCII uses only 7 bits in each byte, bit 0 is always set to 0 during EBCDIC to ASCII conversion and is expected to be 0 during ASCII to EBCDIC conversion.

Converting from EBCDIC to ASCII

The next line shows that the first four EBCDIC values (00, 01, 02, 03) are not changed during conversion to ASCII.

```
         0 1 2 3 4 5 6 7  8 9 A B C D E F
00-0F  000102031A091A7F 1A1A1A0B0C0D0E0F
10-1F  101112131A1A081A 18191A1A1C1D1E1F
20-2F  1A1A1A1A1A0A171B 1A1A1A1A1A050607
30-3F  1A1A161A1A1A1A04 1A1A1A1A14151A1A
40-4F  201A1A1A1A1A1A1A 1A1A5B2E3C282B21
50-5F  261A1A1A1A1A1A1A 1A1A5D242A293B5E
60-6F  2D2F1A1A1A1A1A1A 1A1A7C2C255F3E3F
70-7F  1A1A1A1A1A1A1A1A 1A603A2340273D22
80-8F  1A61626364656667 68691A1A1A1A1A1A
90-9F  1A6A6B6C6D6E6F70 71721A1A1A1A1A1A
A0-AF  1A7E737475767778 797A1A1A1A1A1A1A
B0-BF  1A1A1A1A1A1A1A1A 1A1A1A1A1A1A1A1A
C0-CF  7B41424344454647 48491A1A1A1A1A1A
D0-DF  7D4A4B4C4D4E4F50 51521A1A1A1A1A1A
E0-EF  5C1A535455565758 595A1A1A1A1A1A1A
F0-FF  3031323334353637 38391A1A1A1A1A1A
```

For example, EBCDIC “A” is X'C1' and is converted to X'41'.

Converting from ASCII to EBCDIC

```
         0 1 2 3 4 5 6 7  8 9 A B C D E F
00-0F  00010203372D2E2F 1605250B0C0D0E0F
10-1F  101112133C3D3226 18193F271C1D1E1F
20-2F  404F7F7B5B6C507D 4D5D5C4E6B604B61
30-3F  F0F1F2F3F4F5F6F7 F8F97A5E4C7E6E6F
40-4F  7CC1C2C3C4C5C6C7 C8C9D1D2D3D4D5D6
50-5F  D7D8D9E2E3E4E5E6 E7E8E94AE05A5F6D
60-6F  7981828384858687 8889919293949596
70-7F  979899A2A3A4A5A6 A7A8A9C06AD0A107
80-8F  3F3F3F3F3F3F3F3F 3F3F3F3F3F3F3F3F
90-9F  3F3F3F3F3F3F3F3F 3F3F3F3F3F3F3F3F
A0-AF  3F3F3F3F3F3F3F3F 3F3F3F3F3F3F3F3F
B0-BF  3F3F3F3F3F3F3F3F 3F3F3F3F3F3F3F3F
C0-CF  3F3F3F3F3F3F3F3F 3F3F3F3F3F3F3F3F
D0-DF  3F3F3F3F3F3F3F3F 3F3F3F3F3F3F3F3F
E0-EF  3F3F3F3F3F3F3F3F 3F3F3F3F3F3F3F3F
F0-FF  3F3F3F3F3F3F3F3F 3F3F3F3F3F3F3F3F
```

Accessibility

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 it 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 Volume 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.

© Copyright IBM Corp. 1987, 2002

Notices

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

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

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

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, 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.

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:

© Copyright IBM Corp. 1987, 2002

IBM Corporation
Information Enabling Requests
Dept. DZWA
5600 Cottle Road
San Jose, CA 95193 U.S.A.

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.

Programming Interface Information

The purpose of this publication is to help you use access methods to process data sets.

This publication documents an intended programming interface that enables the customer to write programs for obtaining DFSMS services.

Trademarks

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

CICS
CICS/ESA
CICS/MVS
DB2
DFSMS/MVS
DFSMSdfp
DFSMSdss
DFSMShsm
DFSMSrmm
DFSORT
Hiperbatch
Hiperspace
IBM
IBMLink
MVS/DFP
MVS/ESA
OS/390
Parallel Sysplex
RACF
RAMAC
RETAIN
SAA
SP
System/390
Systems Application Architecture
S/390
VTAM
z/OS

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

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

Glossary

This glossary defines technical terms and abbreviations that DFSMS documentation uses. If you do not find the term you are looking for, refer to the index of the appropriate DFSMS manual, or view the *IBM Dictionary of Computing Terms* at the following web site:

http://www.ibm.com/networking/nsg/nsgmain.htm

This glossary includes terms and definitions from the *IBM Dictionary of Computing*, New York: McGraw-Hill, 1994.

This glossary uses the following kinds of cross-references:

See Refers to (a) a related term, (b) a term that is the expanded form of an abbreviation or acronym, or (c) a synonym or more preferred term.

See also
Refers to a related term.

Contrast with
Refers to a contrasting term.

A

ABSTR. Absolute track (value of SPACE).

ACB. Access method control block.

access method services. A multifunction service program that manages VSAM and non-VSAM data sets, as well as catalogs. Access method services provides the following functions:

- Defines and allocates space for data sets and catalogs
- Converts indexed-sequential data sets to key-sequenced data sets
- Modifies data set attributes in the catalog
- Reorganizes data sets
- Facilitates data portability among operating systems
- Creates backup copies of data sets
- Assists in making inaccessible data sets accessible
- Lists the records of data sets and catalogs
- Defines and builds alternate indexes
- Converts CVOLS to catalogs

ACS. See *automatic class selection (ACS) routine*.

ADSP. See *automatic data set protection*.

AL. American National Standard Labels.

alias. An alternate name for a member of a partitioned data set.

allocation. (1) The entire process of obtaining a volume and unit of external storage. (2) Setting aside space on that storage for a data set.

alternate index. In systems with VSAM, a collection of index entries related to a given base cluster and organized by an alternate key, that is, a key other than the primary key of the associated base cluster data records. An alternate index gives an alternate directory for finding records in the data component of a base cluster.

AMODE. Addressing mode (24, 31, ANY).

ANSI. American National Standards Institute.

AOR. Application owning region.

APF. Authorized program facility.

application. The use to which an access method is put or the end result that it serves, contrasted to the internal operation of the access method.

ASCII. American National Standard Code for Information Interchange.

ATL. See *automated tape library*.

AUL. American National Standard user labels (value of LABEL).

automated tape library data server. A device that consists of robotic components, cartridge storage areas, tape subsystems, and controlling hardware and software, together with the set of tape volumes that reside in the library and can be mounted on the library tape drives. Contrast with *manual tape library*. See also *tape library*.

automatic class selection (ACS) routine. A procedural set of ACS language statements. Based on a set of input variables, the ACS language statements generate the name of a predefined SMS class, or a list of names of predefined storage groups, for a data set.

automatic data set protection (ADSP). In z/OS, a user attribute that causes all permanent data sets created by the user to be automatically defined to RACF with a discrete RACF profile.

AVGREC. Average record scale (JCL keyword).

B

backup. The process of creating a copy of a data set or object to be used in case of accidental loss.

© Copyright IBM Corp. 1987, 2002

base configuration. The part of an SMS configuration that contains general storage management attributes, such as the default management class, default unit, and default device geometry. It also identifies the systems or system groups that an SMS configuration manages.

BCDIC. Binary coded decimal interchange code.

BCS. Basic catalog structure.

BDAM. Basic direct access method.

BDW. Block descriptor word.

BFALN. Buffer alignment (parameter of DCB and DD).

BFTEK. Buffer technique (parameter of DCB and DD).

BISAM. Basic indexed sequential access method.

BLKSIZE. Block size (parameter of DCB, DCBE and DD).

blocking. (1) The process of combining two or more records in one block. (2) Suspending a program process (UNIX).

block size. A measure of the size of a block, usually specified in units such as records, words, computer words, or characters.

BLP. Bypass label processing.

BLT. Block locator token.

BPAM. Basic partitioned access method.

BPI. Bytes per inch.

BSAM. Basic sequential access method.

BSM. Backspace past tape mark and forward space over tape mark (parameter of CNTRL).

BSP. Backspace one block (macro).

BSR. Backspace over a specified number of blocks (parameter of CNTRL).

BUFC. Buffer control block.

BUFCB. Buffer pool control block (parameter of DCB).

BUFL. Buffer length (parameter of DCB and DD).

BUFNO. Buffer number (parameter of DCB and DD).

BUFOFF. Buffer offset (length of ASCII block prefix by which the buffer is offset; parameter of DCB and DD).

C

CA. Control area.

catalog. A data set that contains extensive information required to locate other data sets, to allocate and deallocate storage space, to verify the access authority of a program or operator, and to accumulate data set usage statistics. A catalog has a basic catalog structure (BCS) and its related volume tables of contents (VTOCs) and VSAM volume data sets (VVDSs). See *master catalog* and *user catalog*. See also *VSAM volume data set*.

CBIC. See *control blocks in common*.

CBUF. Control block update facility.

CCHHR. Cylinder/head/record address.

CCSID. Coded Character Set Identifier.

CCW. Channel command word.

CDRA. See *Character Data Representation Architecture (CDRA) API*.

CF. Coupling facility.

Character Data Representation Architecture (CDRA) API. A set of identifiers, services, supporting resources, and conventions for consistent representation, processing, and interchange of character data.

CI. Control interval.

CICS. Customer Information Control System.

CIDF. See *control interval definition field*.

CKDS. Cryptographic key data set.

class. See *SMS class*.

cluster. A named structure consisting of a group of related components. For example, when the data set is key sequenced, the cluster contains both the data and index components; when the data set is entry sequenced, the cluster contains only a data component.

collating sequence. An ordering assigned to a set of items, such that any two sets in that assigned order can be collated.

component. In systems with VSAM, a named, cataloged collection of stored records, such as the data component or index component of a key-sequenced file or alternate index.

compress. (1) To reduce the amount of storage required for a given data set by having the system replace identical words or phrases with a shorter token associated with the word or phrase. (2) To reclaim the unused and unavailable space in a partitioned data set that results from deleting or modifying members by moving all unused space to the end of the data set.

compressed format. A particular type of extended-format data set specified with the (COMPACTION) parameter of data class. VSAM can compress individual records in a compressed-format data set. SAM can compress individual blocks in a compressed-format data set. See *compress*.

concurrent copy. A function to increase the accessibility of data by enabling you to make a consistent backup or copy of data concurrent with the usual application program processing.

configuration. The arrangement of a computer system as defined by the characteristics of its functional units. See *SMS configuration*.

CONTIG. Contiguous space allocation (value of SPACE).

control blocks in common (CBIC). A facility that allows a user to open a VSAM data set so the VSAM control blocks are placed in the common service area (CSA) of the MVS operating system. This provides the capability for multiple memory accesses to a single VSAM control structure for the same VSAM data set.

control interval definition field (CIDF). In VSAM, the four bytes at the end of a control interval that contain the displacement from the beginning of the control interval to the start of the free space and the length of the free space. If the length is 0, the displacement is to the beginning of the control information.

control unit. A hardware device that controls the reading, writing, or displaying of data at one or more input/output devices. A control unit acts as the interface between channels and devices.

cross memory. A synchronous method of communication between address spaces.

CSA. Common service area.

CSW. Channel status word.

CYLOFL. Number of tracks for cylinder overflow records (parameter of DCB).

D

DA. Direct access (value of DEVD or DSORG).

DADSM. See *direct access device space management*.

DASD volume. Direct access storage device volume.

DATACLAS. Data class (JCL keyword).

data class. A collection of allocation and space attributes, defined by the storage administrator, that are used to create a data set.

data control block (DCB). A control block used by access method routines in storing and retrieving data.

data definition (DD) statement. A job control statement that describes a data set associated with a particular job step.

Data Facility Storage Management Subsystem (DFSMS). An operating environment that helps automate and centralize the management of storage. To manage storage, SMS provides the storage administrator with control over data class, storage class, management class, storage group, and automatic class selection routine definitions.

Data Facility Storage Management Subsystem data facility product (DFSMSdfp). An IBM licensed program used to manage programs, devices, and data in a z/OS environment. It is a component of the DFSMS system.

data integrity. Preservation of data or programs for their intended purpose. As used in this publication, data integrity is the safety of data from inadvertent destruction or alteration.

data management. The task of systematically identifying, organizing, storing, and cataloging data in an operating system.

data record. A collection of items of information from the standpoint of its use in an application, as a user supplies it to VSAM for storage. Contrast with *index record*.

data security. Prevention of access to or use of data or programs without authorization. As used in this publication, data security is the safety of data from unauthorized use, theft, or purposeful destruction.

data set. In DFSMS, the major unit of data storage and retrieval, consisting of a collection of data in one of several prescribed arrangements and described by control information to which the system has access. In z/OS non-UNIX environments, the terms *data set* and *file* are generally equivalent and sometimes are used interchangeably. In z/OS UNIX environments, the terms *data set* and *file* have quite distinct meanings. See also *hierarchical file system (HFS) data set*.

data synchronization. The process by which the system ensures that data previously given to the system through WRITE, CHECK, PUT, and PUTX macros is written to some form of nonvolatile storage.

DAU. Direct access unmovable data set (value of DSORG).

DBB. Dictionary building block.

DBCS. See *double-byte character set*.

DCB. Data control block name, macro, or parameter of DD statement. See also *data control block*.

DCBD. Data-control-block dummy section.

DCBE. Data control block extension.

DD. Data definition. See also *data definition (DD) statement*.

DDM. Distributed data management (DDM).

DEB. Data extent block.

DECB. Data event control block.

DEN. Magnetic tape density (parameter of DCB and DD).

DES. Data Encryption Standard.

DEVD. Device dependent (parameter of DCB and DCBD).

DFSMSdss. A DFSMS functional component or base element of z/OS, used to copy, move, dump, and restore data sets and volumes.

DFSMShsm. A DFSMS functional component or base element of z/OS, used for backing up and recovering data, and managing space on volumes in the storage hierarchy.

DFSMSrmm. A DFSMS functional component or base element of z/OS, that manages removable media.

dictionary. A table that associates words, phrases, or data patterns to shorter tokens. The tokens replace the associated words, phrases, or data patterns when a data set is compressed.

direct access. The retrieval or storage of data by a reference to its location in a data set rather than relative to the previously retrieved or stored data.

direct access device space management (DADSM). A collection of subroutines that manages space on disk volumes. The subroutines are Create, Scratch, Extend, and Partial Release.

direct data set. A data set whose records are in random order on a direct access volume. Each record is stored or retrieved according to its actual address or its address according to the beginning of the data set. Contrast with *sequential data set.*

directory entry services (DE Services). Directory Entry (DE) Services provides directory entry services for PDS and PDSE data sets. Not all of the functions will operate on a PDS however. DE Services is usable by authorized as well as unauthorized programs through the executable macro, DESERV.

discrete profile. An RACF profile that contains security information about a single data set, user, or resource. Contrast with *generic profile*.

DISP. Disposition (JCL DD parameter).

DIV. Data-in-virtual.

DLF. Data lookaside facility.

double-byte character set (DBCS). A 2-byte value that can represent a single character for languages that contain too many characters or symbols for each to be assigned a 1-byte value.

DSCB. Data set control block.

DSORG. Data set organization (parameter of DCB and DD and in a data class definition).

dummy storage group. A type of storage group that contains the serial numbers of volumes no longer connected to a system. Dummy storage groups allow existing JCL to function without having to be changed. See also *storage group*.

dynamic allocation. The allocation of a data set or volume using the data set name or volume serial number rather than using information contained in a JCL statement.

dynamic buffering. A user-specified option that requests that the system handle acquisition, assignment, and release of buffers.

E

EBCDIC. Extended binary coded decimal interchange code.

ECB. Event control block.

ECSA. Extended common service area.

entry-sequenced data set (ESDS). A data set whose records are loaded without respect to their contents and whose RBAs cannot change. Records are retrieved and stored by addressed access, and new records are added at the end of the data set.

EOB. End-of-block.

EOD. End-of-data.

EODAD. End-of-data-set exit routine address (parameter of DCB, DCBE, and EXLST).

EOV. End-of-volume.

ESDS. See *entry-sequenced data set*.

ESETL. End-of-sequential retrieval (QISAM macro).

exception. An abnormal condition such as an I/O error encountered in processing a data set or a file.

EXCEPTIONEXIT. An exit routine invoked by an exception.

EXCP. Execute channel program.

EXLST. Exit list (parameter of DCB and VSAM macros).

EXPDT. Expiration date for a data set (JCL keyword).

export. To create a backup or portable copy of a VSAM cluster, alternate index, or user catalog.

extended format. The format of a data set that has a data set name type (DSNTYPE) of EXTENDED. The data set is structured logically the same as a data set that is not in extended format but the physical format is different. Data sets in extended format can be striped or compressed. Data in an extended format VSAM KSDS can be compressed. See also *striped data set* and *compressed format*.

extent. A continuous space on a DASD volume occupied by a data set or portion of a data set.

F

FCB. Forms control buffer.

FEOV. Force end-of-volume (macro).

field. In a record or control block, a specified area used for a particular category of data or control information.

FIFO. See *first-in-first-out*.

file system. In the z/OS UNIX HFS environment, the collection of files and file management structures on a physical or logical mass storage device, such as a diskette or minidisk. See also *hierarchical file system (HFS) data set* and *z/OS UNIX System Services (USS)*.

first-in-first-out (FIFO). A queuing technique in which the next item to be retrieved is the item that has been in the queue for the longest time.

FOR. File owning region.

format-D. ASCII or ISO/ANSI variable-length records.

format-DB. ASCII variable-length, blocked records.

format-DBS. ASCII variable-length, blocked spanned records.

format-DS. ASCII variable-length, spanned records.

format-F. Fixed-length records.

format-FB. Fixed-length, blocked records.

format-FBS. Fixed-length, blocked, standard records.

format-FBT. Fixed-length, blocked records with track overflow option.

format-FS. Fixed-length, standard records.

format-U. Undefined-length records.

format-V. Variable-length records.

format-VB. Variable-length, blocked records.

format-VBS. Variable-length, blocked, spanned records.

format-VS. Variable-length, spanned records.

free control interval pointer list. In a sequence-set index record, a vertical pointer that gives the location of a free control interval in the control area governed by the record.

free space. Space reserved within the control intervals of a key-sequenced data set for inserting new records into the data set in key sequence or for lengthening records already there; also, whole control intervals reserved in a control area for the same purpose.

FSM. Forward space past tape mark and backspace over tape mark (parameter of CNTRL).

FSR. Forward space over a specified number of blocks (parameter of CNTRL).

G

GCR. Group coded recording (tape recording).

GDG. See *generation data group*.

GDS. See *generation data set*.

generation data group (GDG). A collection of historically related non-VSAM data sets that are arranged in chronological order; each data set is called a generation data set.

generation data group base entry. An entry that permits a non-VSAM data set to be associated with other non-VSAM that sets as generation data sets.

generation data set (GDS). One of the data sets in a generation data group; it is historically related to the others in the group.

generic profile. An RACF profile that contains security information about multiple data sets, users, or resources that may have similar characteristics and require a similar level of protection. Contrast with *discrete profile*.

gigabyte. 2^{30} bytes, 1 073 741 824 bytes. This is approximately a billion bytes in American English.

GL. GET macro, locate mode (value of MACRF).

GM. GET macro, move mode (value of MACRF).

GRS. Global resource serialization.

GSR. Global shared resources.

GTF. Generalized trace facility.

H

header entry. In a parameter list of GENCB, MODCB, SHOWCB, or TESTCB, the entry that identifies the type of request and control block and gives other general information about the request.

header, index record. In an index record, the 24-byte field at the beginning of the record that contains control information about the record.

header label. (1) An internal label, immediately preceding the first record of a file, that identifies the file and contains data used in file control. (2) The label or data set label that precedes the data records on a unit of recording media.

HFS. Hierarchical file system.

hierarchical file system (HFS) data set. A data set that contains a POSIX-compliant 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. See also *file system.*

Hiperbatch. An extension to both QSAM and VSAM designed to improve performance. Hiperbatch uses the data lookaside facility to provide an alternate fast path method of making data available to many batch jobs.

Hiperspace. A high performance virtual storage space of up to 2 GB. Unlike an address space, a Hiperspace contains only user data and does not contain system control blocks or common areas; code does not execute in a Hiperspace. Unlike a data space, data in Hiperspace cannot be referenced directly; data must be moved to an address space in blocks of 4 KB before they can be processed. Hiperspace pages can be backed by expanded storage or auxiliary storage, but never by main storage. The Hiperspace used by VSAM is only backed by expanded storage. See also *Hiperspace buffer*.

Hiperspace buffer. A 4K-multiple buffer that facilitates the moving of data between a Hiperspace and an address space. VSAM Hiperspace buffers are backed only by expanded storage.

I

IBG. Interblock gap.

ICI. Improved control interval access.

import. To restore a VSAM cluster, alternate index, or catalog from a portable data set created by the EXPORT command.

index record. A collection of index entries that are retrieved and stored as a group. Contrast with *data record*.

INOUT. Input and then output (parameter of OPEN).

I/O. Input/output.

I/O device. An addressable input/output unit, such as a direct access storage device, magnetic tape device, or printer.

IOB. Input/output block.

IRG. Interrecord gap.

IS. Indexed sequential (value of DSORG).

ISAM interface. A set of routines that allow a processing program coded to use ISAM (indexed sequential access method) to gain access to a VSAM key-sequenced data set.

ISMF. Interactive storage management facility.

ISO. International Organization for Standardization.

ISU. Indexed sequential unmovable (value of DSORG).

J

JES. Job entry subsystem.

JFCB. Job file control block.

JFCBE. Job file control block extension.

K

KEYLEN. Key length (JCL and DCB keyword).

key-sequenced data set (KSDS). A VSAM data set whose records are loaded in ascending key sequence and controlled by an index. Records are retrieved and stored by keyed access or by addressed access, and new records can be inserted in key sequence because of free space allocated in the data set. Relative byte addresses can change, because of control interval or control area splits.

kilobyte. 2^{10} bytes, 1 024 bytes.

KSDS. See *key-sequenced data set*.

L

large block interface (LBI). The set of BSAM, BPAM, and QSAM interfaces that deal with block sizes in 4 byte fields instead of 2 byte fields.

LBI. See *large block interface*.

LDS. See *linear data set*.

library. A partitioned data set (PDS) that contains a related collection of named members. See *partitioned data set*.

linear data set (LDS). A VSAM data set that contains data but no control information. A linear data set can be accessed as a byte-addressable string in virtual storage.

load module. The output of the linkage editor; a program in a format ready to load into virtual storage for execution.

locate mode. A way of providing data by pointing to its location instead of moving it.

LRECL. Logical record length (JCL and DCB keyword).

LRI. Logical record interface.

LSR. Local shared resources.

M

MACRF. Macro instruction form (parameter of DCB and ACB).

management class. (1) A named collection of management attributes describing the retention and backup characteristics for a group of data sets, or for a group of objects in an object storage hierarchy. For objects, the described characteristics also include class transition. (2) In DFSMSrmm, if assigned by ACS routine to system-managed tape volumes, management class can be used to identify a DFSMSrmm vital record specification.

manual tape library. Installation-defined set of tape drives defined as a logical unit together with the set of system-managed volumes that can be mounted on the drives.

master catalog. A catalog that contains extensive data set and volume information that VSAM requires to locate data sets, to allocate and deallocate storage space, to verify the authorization of a program or operator to gain access to a data set, and to accumulate usage statistics for data sets.

MBBCCHHR. Module number, bin number, cylinder number, head number, record number.

media. The disk surface on which data is stored.

MEDIA2. Enhanced Capacity Cartridge System Tape.

MEDIA3. High Performance Cartridge Tape.

MEDIA4. Extended High Performance Cartridge Tape.

megabyte. 2^{20} bytes, 1 048 576 bytes.

member. A partition of a PDS or PDSE.

MGMTCLAS. Management class (JCL keyword).

migration. The process of moving unused data to lower cost storage in order to make space for high-availability data. If you wish to use the data set, it must be recalled. See also *migration level 1* and *migration level 2*.

migration level 1. DFSMShsm-owned DASD volumes that contain data sets migrated from primary storage volumes. The data can be compressed. See also *storage hierarchy*. Contrast with *migration level 2* and *primary storage*.

migration level 2. DFSMShsm-owned tape or DASD volumes that contain data sets migrated from primary storage volumes or from migration level 1 volumes. The data can be compressed. See also *storage hierarchy*. Contrast with *migration level 1* and *primary storage*.

MLA. See *multilevel alias (MLA) facility*.

MOD. Modify data set (value of DISP).

move mode. A transmittal mode in which the record to be processed is moved into a user work area.

MSHI. Main storage for highest-level index (parameter of DCB).

MSWA. Main storage for work area (parameter of DCB).

multilevel alias (MLA) facility. A function in catalog address space that allows catalog selection based on one to four data set name qualifiers.

MVS/DFP. An IBM licensed program that is the base for the storage management subsystem.

MVS/ESA™. Multiple Virtual Storage/Enterprise Systems Architecture. A z/OS operating system environment that supports ESA/390.

MVS/ESA SP™. An IBM licensed program used to control the MVS operating system. MVS/ESA SP together with DFSMS compose the base MVS/ESA operating environment.

N

national. In z/OS, the three characters that in U.S. EBCDIC are represented as X'7C', X'7B' and X'5B', which are @ (“at”), # (“pound” sign or “number”) and $

("dollar" sign). On many keyboards and display screens in other countries, these byte values display differently.

NCI. Normal control interval.

NCP. Number of channel programs (parameter of DCB and DD).

NFS. Network File System.

NIST. National Institute of Standards and Technology.

non-VSAM data set. A data set allocated and accessed using one of the following methods: BDAM, BISAM, BPAM, BSAM, QSAM, QISAM.

NOPWREAD. No password required to read a data set (value of LABEL).

NRZI. Nonreturn-to-zero-inverted.

NSL. Nonstandard label (value of LABEL).

NSR. Nonshared resources.

NTM. Number of tracks in cylinder index for each entry in lowest level of master index (parameter of DCB).

NUB. No user buffering.

NUP. No update.

O

object. A named byte stream having no specific format or record orientation.

object backup storage group. A type of storage group that contains optical or tape volumes used for backup copies of objects. See also *storage group*.

object storage group. A type of storage group that contains objects on DASD, tape, or optical volumes. See also *storage group*.

operand. Information entered with a command name to define the data on which a command operates and to control the execution of the command.

operating system. Software that controls the execution of programs; an operating system input/output control, and data management.

OPTCD. Optional services code (parameter of DCB).

optical volume. Storage space on an optical disk, identified by a volume label. See also *volume*.

optimum block size. For non-VSAM data sets, optimum block size represents the block size that would result in the greatest space utilization on a device, taking into consideration record length and device characteristics.

OUTIN. Output and then input (parameter of OPEN).

OUTINX. Output at end of data set (to extend) and then input (parameter of OPEN).

P

page. (1) A fixed-length block of instructions, data, or both, that can be transferred between real storage and external page storage. (2) To transfer instructions, data, or both between real storage and external page storage.

page space. A system data set that contains pages of virtual storage. The pages are stored in and retrieved from the page space by the auxiliary storage manager.

paging. A technique in which blocks of data, or pages, are moved back and forth between main storage and auxiliary storage. Paging is the implementation of the virtual storage concept.

Parallel Sysplex. A collection of systems in a multisystem environment supported by Cross System Coupling Facility (XCF).

partitioned data set (PDS). A data set on direct access storage that is divided into partitions, called members, each of which can contain a program, part of a program, or data.

partitioned data set extended (PDSE). A system-managed data set that contains an indexed directory and members that are similar to the directory and members of partitioned data sets. A PDSE can be used instead of a partitioned data set.

password. A unique string of characters that a program, a computer operator, or a terminal user must supply to meet security requirements before a program gains access to a data set.

PDAB. Parallel data access block.

PDS. See *partitioned data set*.

PDS directory. A set of records in a partitioned data set (PDS) used to relate member names to their locations on a DASD volume.

PDSE. See *partitioned data set extended*.

PE. Phase encoding (tape recording mode).

petabyte. 2^{50} bytes, 1 125 899 906 842 624 bytes. This is approximately a quadrillion bytes in American English.

PL. PUT macro, locate mode (value of MACRF).

PM. PUT macro, move mode (value of MACRF).

PO. Partitioned organization (value of DSORG).

pointer. An address or other indication of location. For example, an RBA is a pointer that gives the relative location of a data record or a control interval in the data set to which it belongs.

pool storage group. A type of storage group that contains system-managed DASD volumes. Pool storage groups allow groups of volumes to be managed as a single entity. See also *storage group*.

portability. The ability to use VSAM data sets with different operating systems. Volumes whose data sets are cataloged in a user catalog can be demounted from storage devices of one system, moved to another system, and mounted on storage devices of that system. Individual data sets can be transported between operating systems using access method services.

POSIX. Portable operating system interface for computer environments.

POU. Partitioned organization unmovable (value of DSORG).

primary space allocation. Amount of space requested by a user for a data set when it is created. Contrast with *secondary space allocation*.

primary key. One or more characters within a data record used to identify the data record or control its use. A primary key must be unique.

primary storage. A DASD volume available to users for data allocation. The volumes in primary storage are called primary volumes. See also *storage hierarchy*. Contrast with *migration level 1* and *migration level 2*.

PRTSP. Printer line spacing (parameter of DCB).

PS. Physical sequential (value of DSORG).

PSU. Physical sequential unmovable (value of DSORG).

PSW. Program status word.

Q

QISAM. Queued indexed sequential access method.

QSAM. Queued sequential access method.

R

R0. Record zero.

RACF. See *Resource Access Control Facility*.

RACF authorization. (1) The facility for checking a user's level of access to a resource against the user's desired access. (2) The result of that check.

random access. See *direct access*.

RBA. Relative byte address.

RDBACK. Read backward (parameter of OPEN).

RDF. See *record definition field*.

RDW. Record descriptor word.

RECFM. Record format (JCL keyword and DCB macro parameter).

record definition field (RDF). A field stored as part of a stored record segment; it contains the control information required to manage stored record segments within a control interval.

record-level sharing. See *VSAM Record-Level Sharing (VSAM RLS)*.

REFDD. Refer to previous DD statement (JCL keyword).

register. An internal computer component capable of storing a specified amount of data and accepting or transferring this data rapidly.

relative record data set (RRDS). A VSAM data set whose records have fixed or variable lengths, and are accessed by relative record number.

Resource Access Control Facility (RACF). An IBM licensed program that is included in z/OS Security Server and is also available as a separate program for the z/OS and VM environments. RACF provides access control by identifying and verifying the users to the system, authorizing access to protected resources, logging detected unauthorized attempts to enter the system, and logging detected accesses to protected resources.

RETPD. Retention period (JCL keyword).

reusable data set. A VSAM data set that can be reused as a work file, regardless of its old contents. It must not be a base cluster of an alternate index.

RKP. Relative key position (parameter of DCB).

RLS. Record-level sharing. See *VSAM Record-Level Sharing (VSAM RLS)*.

RLSE. Release unused space (DD statement).

RMODE. Residence mode.

RPL. Request parameter list.

S

SAA. Systems Application Architecture.

SBCS. Single-byte character set.

scheduling. The ability to request that a task set should be started at a particular interval or on occurrence of a specified program interrupt.

SDW. Segment descriptor word.

secondary space allocation. Amount of additional space requested by the user for a data set when primary space is full. Contrast with *primary space allocation*.

security. See *data security*.

SEOF. Software end-of-file.

sequence checking. The process of verifying the order of a set of records relative to some field's collating sequence.

sequential data set. A data set whose records are organized on the basis of their successive physical positions, such as on magnetic tape. Contrast with *direct data set.*

sequential data striping. A software implementation of a disk array that distributes data sets across multiple volumes to improve performance.

SER. Volume serial number (value of VOLUME).

serialization. In MVS, the prevention of a program from using a resource that is already being used by an interrupted program until the interrupted program is finished using the resource.

service request block (SRB). A system control block used for dispatching tasks.

SETL. Set lower limit of sequential retrieval (QISAM macro).

SF. Sequential forward (parameter of READ or WRITE).

shared resources. A set of functions that permit the sharing of a pool of I/O related control blocks, channel programs, and buffers among several VSAM data sets open at the same time. See also *LSR* and *GSR*.

SI. Shift in.

SK. Skip to a printer channel (parameter of CNTRL).

SL. IBM standard labels (value of LABEL).

slot. For a relative record data set, the data area addressed by a relative record number which may contain a record or be empty.

SMB. See *system-managed buffering*.

SMS. See *storage management subsystem* and *system-managed storage*.

SMS class. A list of attributes that SMS applies to data sets having similar allocation (data class), performance (storage class), or backup and retention (management class) needs.

SMS configuration. A configuration base, Storage Management Subsystem class, group, library, and drive definitions, and ACS routines that the Storage Management Subsystem uses to manage storage. See also *configuration, base configuration,* and *source control data set*.

SMSI. Size of main-storage area for highest-level index (parameter of DCB).

SMS-managed data set. A data set that has been assigned a storage class.

SMSVSAM. The name of the VSAM server that provides VSAM record-level sharing (RLS). See also *VSAM record-level sharing (VSAM RLS)*.

SMSW. Size of main-storage work area (parameter of DCB).

SO. Shift out.

source control data set (SCDS). A VSAM linear data set containing an SMS configuration. The SMS configuration in an SCDS can be changed and validated using ISMF.

SP. Space lines on a printer (parameter of CNTRL).

spanned record. A logical record whose length exceeds control interval length, and as a result, crosses, or spans, one or more control interval boundaries within a single control area.

SRB. See *service request block*.

SS. Select stacker on card reader (parameter of CNTRL).

storage administrator. A person in the data processing center who is responsible for defining, implementing, and maintaining storage management policies.

storage class. A collection of storage attributes that identify performance goals and availability requirements, defined by the storage administrator, used to select a device that can meet those goals and requirements.

storage group. A collection of storage volumes and attributes, defined by the storage administrator. The collections can be a group of DASD volumes or tape volumes, or a group of DASD, optical, or tape volumes treated as a single object storage hierarchy.

storage hierarchy. An arrangement of storage devices with different speeds and capacities. The levels of the storage hierarchy include main storage (memory, DASD cache), primary storage (DASD containing

uncompressed data), migration level 1 (DASD containing data in a space-saving format), and migration level 2 (tape cartridges containing data in a space-saving format). See also *primary storage, migration level 1* and *migration level 2*.

Storage Management Subsystem (SMS). A DFSMS facility used to automate and centralize the management of storage. Using SMS, a storage administrator describes data allocation characteristics, performance and availability goals, backup and retention requirements, and storage requirements to the system through data class, storage class, management class, storage group, and ACS routine definitions.

STORCLAS. Storage class (JCL keyword).

stripe. In DFSMS, the portion of a striped data set, such as an extended format data set, that resides on one volume. The records in that portion are not always logically consecutive. The system distributes records among the stripes such that the volumes can be read from or written to simultaneously to gain better performance. Whether it is striped is not apparent to the application program.

striped data set. An extended format data set that occupies multiple volumes. A software implementation of sequential data striping.

striping. A software implementation of a disk array that distributes a data set across multiple volumes to improve performance.

SUL. IBM standard and user labels (value of LABEL).

SYNAD. The physical error user exit routine. Synchronous error routine address (parameter of DCB, DCBE, and EXLST).

synchronize. See *data synchronization*.

system. A functional unit, consisting of one or more computers and associated software, that uses common storage for all or part of a program and also for all or part of the data necessary for the execution of the program.

Note: A computer system can be a stand-alone unit, or it can consist of multiple connected units.

system-managed data set. A data set that has been assigned a storage class.

system-managed buffering (SMB). A facility available for system-managed extended-format VSAM data sets in which DFSMSdfp determines the type of buffer management technique along with the number of buffers to use, based on data set and application specifications.

system-managed directory entry (SMDE). A directory that contains all the information contained in the PDS directory entry (as produced by the BLDL macro) as well as information specific to program objects, in the extensible format.

system-managed storage. Storage managed by the Storage Management Subsystem. SMS attempts to deliver required services for availability, performance, and space to applications.

system-managed tape library. A collection of tape volumes and tape devices, defined in the tape configuration database. A system-managed tape library can be automated or manual. See also *tape library.*

T

T. Track overflow option (value of RECFM); user-totaling (value of OPTCD).

tape library. A set of equipment and facilities that support an installation's tape environment. This can include tape storage racks, a set of tape drives, and a set of related tape volumes mounted on those drives. See also *automated tape library data server* and *system-managed tape library*.

tape storage group. A type of storage group that contains system-managed private tape volumes. The tape storage group definition specifies the system-managed tape libraries that can contain tape volumes. See also *storage group*.

tape volume. A tape volume is the recording space on a single tape cartridge or reel. See also *volume*.

task control block (TCB). Holds control information related to a task.

TCB. See *task control block*.

terabyte. 2^{40} bytes, 1 099 511 627 776 bytes. This is approximately a trillion bytes in American English.

TIOT. Task input/output table.

TMP. Terminal monitor program.

trailer label. A file or data set label that follows the data records on a unit of recording media.

transaction ID (TRANSID). A number associated with each of several request parameter lists that define requests belonging to the same data transaction.

TRC. Table reference character.

TRTCH. Track recording technique (parameter of DCB and of DD statement).

TTR. Track record address. A representation of a relative track address.

U

UCB. Unit control block.

UCS. See *universal character set*.

UHL. User header label.

universal character set (UCS). A printer feature that permits the use of a variety of character arrays. Character sets used for these printers are called UCS images.

UPAD. User processing exit routine.

UPD. Update.

update number. For a spanned record, a binary number in the second RDF of a record segment that indicates how many times the segments of a spanned record should be equal. An inequality indicates a possible error.

USAR. User security authorization record.

user buffering. The use of a work area in the processing program's address space for an I/O buffer; VSAM transmits the contents of a control interval between the work area and direct access storage without intermediary buffering.

user catalog. An optional catalog used in the same way as the master catalog and pointed to by the master catalog. It also lessens the contention for the master catalog and facilitates volume portability.

USVR. User security verification routine.

UTL. User trailer label.

USS. See *z/OS UNIX System Services*.

V

VBS. Variable blocked spanned.

VIO. Virtual input/output.

volume. The storage space on DASD, tape, or optical devices, which is identified by a volume label. See also *DASD volume, optical volume,* and *tape volume*.

volume positioning. Rotating the reel or cartridge so that the read-write head is at a particular point on the tape.

VSAM. Virtual storage access method.

VSAM record-level sharing (VSAM RLS). An extension to VSAM that provides direct record-level sharing of VSAM data sets from multiple address spaces across multiple systems. Record-level sharing uses the z/OS Coupling Facility to provide cross-system locking, local buffer invalidation, and cross-system data caching.

VSAM RLS. See *VSAM record-level sharing*.

VSAM shared information (VSI). Blocks that are used for cross-system sharing.

VSAM sphere. The base cluster of a VSAM data set and its associated alternate indexes.

VSAM volume data set (VVDS). A data set that describes the characteristics of VSAM and system-managed data sets that reside on a given DASD volume; part of a catalog. See also *catalog*.

VSI. See *VSAM shared information*.

VTOC. Volume table of contents.

VVDS. See *VSAM volume data set*.

XDAP. Execute direct access program.

z/OS UNIX System Services (USS). The set of functions provided by the SHELL and UTILITIES, kernel, debugger, file system, C/C++ Run-Time Library, Language Environment, and other elements of the z/OS operating system that allow users to write and run application programs that conform to UNIX standards.

Index

Numerics

16 MB line
  above 342, 400
  above, below 335
  below 342
2 GB bar
  DCB central storage address 353
  DCBE central storage address 353
  real buffer address 342, 343
24-bit addressing mode 400
2540 Card Read Punch 312
31-bit addressing mode 343, 351, 400
  buffers above 16 MB 170
  keywords for vsam 263
  multiple LSR pools 212
  OPEN, CLOSE (non-VSAM) 335
3211 printer 480
3262 Model 5 printer 480
3525 Card Punch
  opening associated data sets 369
  record format 312
3800 Model 3 printer, table reference character 290, 293, 310
4245 printer 480
4248 printer 480
7-track tapes, VSE (Virtual Storage Extended) 479

A

ABEND
  001 316, 584
  002 293
  002-68 397
  013 341, 342, 374
  013-4C 341
  013-60 380
  013-DC 306
  013-FD 324
  013-FE 324
  013-FF 324
  031 584
  039 584
  03B 584, 588
  0C4 263
  117-3C 510
  213 336
  213-88 21
  237-0C 510
  513-10 606
  513-14 605
  913 61
  913-34 521
  937-44 521
  D37 438
  EC6-FF0D 20
ABEND macro 506, 552
abnormal termination 154
ABS value 482
absolute generation name 469
absolute track allocation 29
ABSTR (absolute track) value for SPACE parameter
  absolute track allocation 29
  ISAM 548, 554
  SMS restriction 39
ACB (access control block) 335
ACB macro
  access method control block 137
  buffer space 139, 169
  improved control interval access 191
  MACRF parameter 198
  PASSWD parameter 58
  RMODE31 parameter 170
  storage for control block 142
  STRNO parameter 175
ACCBIAS subparameter 170, 172
access method services
  allocation examples 32, 34
  ALTER LIMIT command 476
  ALTER ROLLIN command 476
  DEFINE command (non-VSAM) 469
access methods 585
  above 2 GB 15
  basic 347, 353
  BDAM (basic direct access method) 535
  create control block 138
  data management 12
  EXAMINE command 235
  indexed sequential data set 545
  KSDS cluster analysis 235
  processing signals 20
  processing USS files 18
  queued 359, 362
  queued, buffer control 346
  recommended xxiii
  selecting, defining 4, 7
  VSAM 75, 104
  VSAM (virtual storage access method) 16
  VSAM, non-VSAM 13
accessibility 611
ACS (automatic class selection) 29
  assigning classes 329
  data class 29
  distributed file manager (DFM) 28
  installation data class 319
  management class 29
  SMS configuration 27
  storage class 29
ACS routines 377
actual track address
  BDAM (basic direct access method) 535
  DASD volumes 10
  direct data sets 539
  ISAM 574
  using feedback option 540
add PDS members 411

© Copyright IBM Corp. 1987, 2002

address
  accessing a KSDS's index 275
  relative
    direct data sets 539
    directories 407, 408, 431
addressed access 98
addressed direct retrieval 148
addressed sequential retrieval 147
ADDVOL command 43
ADSP processing, cluster profiles 56
AL (ISO/ANSI standard label) 57
alias name
  PDS
    creating 418
    deleting 418
    directory format 407
  PDSE
    creating 457
    deleting 457, 465
    differences from PDS 431
    directory format 431
    length 429
    program object 450
    renaming 465
    restrictions 434
    storage requirements 440
ALLOCATE command
  building alternate indexes 120
  creating data sets 377
  data set allocation 14, 30
  defining data sets 106
  examples 31, 32, 34, 35, 132
  releasing space 334
  temporary data set names 108
  USS files 401
allocation
  data set
    definition 30
    examples 30, 35, 330
    generation 473, 475
    partitioned 410, 413
    sequential 377, 378
    system-managed 330
    using access method services 32, 34
    VSAM 269
  retrieval list 504
ALTER command 14, 58, 59, 62, 114, 125, 167, 467
  GDG limits 476
  rolling in generation data sets 476
ALTER LIMIT command, access method services 477
alternate index 103, 173
  backing up 124
  maximum definition 123
  name, define 121
  nonunique keys 122
  nonunique pointers, maximum 100
  verification 53
alternate key 152
ALTERNATEINDEX parameter 121
ALX command 23
AMASPZAP service aid 464
AMP parameter 139, 159, 272, 582, 583, 587
ANSI (American National Standards Institute) 11
ANSI control characters, chained scheduling 391
AOR (application owning region) 225
APF (authorized program facility) 55, 127, 192, 521
  access method services 64
  improved control interval access 191
ARG parameter 219
ASCII (American National Standard Code for Information Interchange)
  block prefix, format-D records 302
  buffer alignment 342
  data conversion 11, 15
  format, converting data 356, 359
  ISO/ANSI tapes 299
  label character coding 11
  tape records 342
associated data sets, opening 369
asynchronous mode 153
ATL (Automatic Tape Library) 12
ATTACH macro 197, 221, 367
ATTEMPTS parameter 60
AUTHORIZATION parameter 62, 110, 260
authorized program facility 64
automatic blocking/deblocking, queued access methods 359
automatic error options 499
AVGREC keyword
  allocating space 37, 410
  scale, modify 38
AVGREC parameter 132, 409, 410

B

backspacing
  BSP macro 481
  CNTRL macro 479
backup
  EXPORT/IMPORT 50
  program (write) 51
backup procedures 47
base
  cluster 100, 200
  RBA index entry 280
  sphere 200
basic access method
  buffer control 341
  overlapped I/O 353
  reading and writing data 353
BCS (basic catalog structure) 126
BDAM (basic direct access method)
  creating direct data sets 537
  data set sharing 365, 368
  data sets 544
  dynamic buffering 536
  exclusive control (block) 540
  extended search option 539
  feedback option 540
  I/O requests 544
  I/O status information 486
  organization 536

BDAM (basic direct access method) *(continued)*
READ macro 355, 356
record addressing 539
records (adding, updating) 540
spanned variable-length records 294, 297
user labels 543
using 535
VIO data sets 538
WRITE macro 356
BDW (block descriptor word) 323
blocked records 302
extended 294
location in buffer 355
nonextended 292, 294
variable-length block 293
BFRFND field 214
BFTEK parameter 355
binder 442, 452
BISAM (basic indexed sequential access method)
description 545
ECB (event control block)
conditions 488, 489
exception code bits 488
error conditions 582
I/O status information 486
indexed sequential data set
retrieving 561, 566
updating 561, 567
sequential data sets 377
sharing a data set 365, 367, 368
sharing a DCB 367
SYNAD routine 497
BISAM (queued indexed sequential access method)
SYNAD routine 497
BLDINDEX command 105, 123, 135
alternate index, build 135
BLDL macro 406, 411, 413, 423, 431, 434, 445, 451, 453, 459
build list format 413
coding example 421
description
PDS (partitioned data set) 413, 414
BLDL NOCONNECT option 445
BLDVRP macro 137, 221, 263
access method 212
resource pool 211
BLKSIZE parameter
BDAM 536
block size
maximum 322
minimum 323
system determined 326
card reader and punch 311
determining block length 393, 394
device independence 388
extended format data sets 397
LBI (large block interface) 322
PDS space 409
performance 388, 390
reading PDSE directory 435, 466
recommendation 315
BLKSIZE parameter *(continued)*
sequential concatenation 380, 381
space allocation 37
BLKSZLIM parameter
block size limit 326
keyword 315, 324
block
boundaries 401
chaining 66
control
real storage 192
single structure, share 197
count
EOV exit 510
exit routine 510, 511
event control 487, 489
grouping records 289
length
BSAM, BPAM, or BDAM READ 393
change 394
descriptor word 293
determining 393
variable 293
level
compression 397
null segment 296
output buffer 495
prefix
access method requirements 302
ASCII magnetic tape 342
block length 302
blocked records 302
buffer alignment 342
creating 302
data types 301, 302
format-D records 302
reading 302
prefix, reading 301
processing
READ macro 353
WRITE macro 353
READ macro 355
record processing 401
size
BSAM, BPAM, or BDAM READ 393
card reader and punch 311
compressed format data set 398
ISO/ANSI spanned records 303
ISO/ANSI Version 3 or Version 4 tapes 303
JFCLRECL field 374
large 390
like concatenation 382
limit 322, 324
maximum 320
minimum 309
new DASD data set 325
non-VSAM data sets 323
PDSE (partitioned data set extended) 432, 435
physical 439
printer 359
recommendation 315

block *(continued)*
size *(continued)*
SYSOUT DD statement 375
tape data set 326
VSAM data sets 159, 160
spanned records 294
blocking 8
automatic 359
fixed-length records 290, 302
records
QISAM 545
variable-length records 292, 293
blocking factor 373
BLP (bypass label processing) 57
BLT (block locator token) 10, 398
boundary
alignment
buffer 342
boundary alignment
data control block 331
boundary extent
cylinder, track 334
BPAM (basic partitioned access method)
data set
DCB ABEND exit routine 505
sharing 365, 368
data set (EODAD (end-of-data-set)) 493
description 16
I/O status information 486
PDSE (partitioned data set extended) 440
processing
PDS (partitioned data set) 405, 427
PDSE 429, 468
retrieving members
PDS (partitioned data set) 419, 423
retrieving members (PDSE) 459
BSAM (basic sequential access method)
BLKSIZE parameter 393
block size
like concatenation 382
BUFOFF parameter
chained scheduling 391
CHECK macro 398
compatible record format 383
creating
PDS (partitioned data set) 410, 413
PDSE 444
creating (PDSE) 441
data set
user labels 514
data set (EODAD) 493
data sets
user totaling 523, 524
description 17
extended format data sets
sequential data striping 399
extending a sequential data set 387
I/O status information 486
incompatible
record format 380
larger NCP, set 400
BSAM (basic sequential access method) *(continued)*
like concatenation 384
NCP parameter 391
OPEN processing
JFCB 383
overlap I/O 390
overlap of I/O 353
performance chaining 400
READ 391
read (PDSE directory) 431, 466
READ macro 355, 356
reading
PDS directory 427
record length 393
retrieving
PDS member 419
retrieving (PDSE member) 458
sequential data sets 380
sharing a data set 365, 368
update PDSE directory 457
updating
PDS member 424
PDSE member 464
updating (PDS directory) 418
USS files 401
WRITE 391
WRITE macro 356
write, short block 394
BSAM DCB macro 535
BSP macro 389, 446, 481, 493
BSAM 315
BSTRNO parameter 179
BUFCB parameter 343
buffer
acquisition 352
alignment 342
control 352
index allocation 176
length
calculating 571
managing 183
non-VSAM
acquisition 341
control 346
pool 342, 351
segment 346
nonshared resource 169
pool 341
constructing 345
constructing automatically 344
record area 343
real storage, VSAM 192
releasing 351
retain, release 150
segment 341
sequential access 178
simple
parallel input 360
SMBHWT Hiperspace 171
space 169
VSAM 159

buffer *(continued)*
truncating 351
VSAM
acquisition 179
allocation 170, 179
concurrent data set positioning 170
Hiperspace 212
invalidation 206
marking for output 219
parameters 179
pool 211, 214, 216
UBF 190
user storage area 190
VSAM, direct access 175
VSAM, path 179
writing (deferred/forced) 149
buffer pool
size
maximum 342
buffered data invalidation
VARY OFFLINE 463
buffering
simple 378
buffering macros
queued access method 351
BUFFERS parameter 214
BUFFERSPACE parameter 109, 122, 169, 179
BUFL (buffer length) 315
BUFL parameter 311, 312, 324, 344
BUFND parameter 169, 179
BUFNI parameter 169, 179
BUFNO
buffer pool
construct automatically 344
BUFNO (number of buffers) 315
BUFNO parameter 391, 400
BUFOFF parameter 391
BUFRDS field 214
BUFSP parameter 169, 179
BUILD macro 324, 341, 342, 345, 351, 571
buffer pool 342
description 343
BUILDRCD macro 295, 324, 341, 343, 346
usage 296
BWD (backward) 98
bypass label processing (BLP) 57
BYPASSLLA option 413, 445

C

CA (control area) 83, 232
caching VSAM RLS data 161, 224
CANCEL command 396
capacity record 537
card
reader (CNTRL macro) 479
catalog 55, 469, 477
BCS component 235
control interval size 159
EXAMINE command 235
protection 58
catalog *(continued)*
structural analysis 235
user, examining 236
catalog damage recovery 52
catalog management 16
CATALOG parameter 61, 109, 122, 134, 135
catalog verification 53
cataloging
data sets
GDG 469
cataloging data sets 469, 471
CATLIST line operator 126
CBIC (control blocks in common) 192, 200
CBUF (control block update facility) 205
CCSID (coded character set identifier) 11, 391, 593, 596, 603
CCSID parameter 299
decision tables 605
QSAM (queued sequential access method) 299, 359
CDRA 593
central storage address
DCB, DCBE 353
CF (coupling facility) 223
CF cache for VSAM RLS data 161, 224
chained
scheduling
channel programs 390
description 390
ignored request conditions 390
non-DASD data sets 390
chaining
RPL (request parameter list) 141
channel
programs
chained segments 390
channel programs
number of (NCP) 355
channel status word 492
chapter reference
control intervals, processing 183
character
control
chained scheduling 391
character codes
DBCS 533
CHARS parameter 310
CHECK macro 154
BDAM 535
before CLOSE 332
BPAM 353
BSAM 353
compressed format data set 398
DECB 354
description 356
determining block length 393
end-of-data-set routine 493
I/O operations 17
MULTACC 392
PDSE synchronization 435
performance 392

CHECK macro *(continued)*
read 393
sharing data set 366
SYNAD routine 500
TRUNC macro 356
unlike data sets 385
update 386, 423
VSAM 153
writing PDS 411
checkpoint
shared data sets 207
shared resource restrictions 220
checkpoint data set
security 369
checkpoint/restart 220
CHKPT macro 220, 511
CI (control interval) 232
CICS
CICS Transaction Server xxiii
CICS (Customer Information Control System)
with VSAM RLS 225
CICS transactional recovery
VSAM recoverable files 227
CIDF (control interval definition field) 185
control information 76
CIMODE parameter 50
ciphertext 66
CIPOPS utility 512
class specifications 33
classes
examples 330
JCL keyword for 329
clear, reset to empty (PDSE directory)
STOW INITIALIZE 458
CLOSE macro 314, 335, 499
description
non-VSAM 332, 338
VSAM 154
multiple data sets 332
parallel input processing 360, 362
PDS (partitioned data set) 418, 419
SYNAD 332
temporary close option 332, 338
TYPE=T 332, 338
volume positioning 332, 339
closing a data set
non-VSAM 332, 338
VSAM 154
CLUSTER parameter 108, 131
cluster verification 53
clusters 106
define, naming 106
CNTRL macro 315, 389, 390, 479
CO (Create Optimized) 172, 174
COBOL applications 85
COBOL programming language 326
CODE parameter 60
coded character sets
sorted by CCSID 593
sorted by default LOCALNAME 596
codes
exception 492
common service area 192
COMPACTION option 398
completion check
asynchronous requests 153
COMPRESS parameter 398
compressed control
information field 78
compressed format data sets 38, 397, 399
access method 309
CHECK macro 398
fixed-length blocked records 328
MULTACC option 391
synchronizing data 483
compression
DBB, tailored 399
type, tailored 397
CON_INTENT=HOLD parameter 447
CONCAT parameter 416, 449, 454
concatenation
data sets
extended format 399
related 379
tape and DASD 381
unlike attributes 384
data sets (QSAM, BSAM) 380
defined 426
like, sequential 466
partitioned 426, 427, 466
reading a PDS or PDSE directory 427
sequential 426, 466
sequential, partitioned 466
concurrent
copy (backup, recovery) 51
copy (DASD) 47
copy (DFSMShsm) 51
data set positioning 175
positioning (STRNO) 176
requests (maximum (255)) 152
requests (parallel) 141
requests (positioning) 151
condition, exception 486
control
areas 78
control access
shared data 230
control area
free control intervals 166
size 164, 166
control block
data event 487
event 489
generate 142
grouping 353
in common 192
macros 15
manipulation macros 142, 144
structure 205
update facility 205
control buffer 480

control characters
ANSI 302
fixed-length records 300
format-D records 302
format-F records 300
format-V records 293
ISO/ANSI 300
optional 308
SYSOUT data set 374
variable-length records 293
control information 280
structure 185
control interval 76
access 183, 191
improved 191
index 275
update contents 185
access, password 140
definition field 185
device selection 191
free space 166
improved access 191, 192
index RBA 280
maximum record size 160
size 160, 161
adjustments 164
KSDS 163
split
JRNAD routine 247
storage 179
control interval splits 91
control section, dummy 331
CONTROLINTERVALSIZE parameter 110
conversion
ASCII to/from EBCDIC 356, 359
indirect addressing 537
ISAM to VSAM 585
PDS to PDSE 411
PDSE to PDS 411
conversion codes, tables 610
copy DBCS characters 533
count area
ISAM index entry format 549
count-data format 9
count-key-data format 9
coupling facility CF cache for VSAM RLS data 161, 224
CPOOL macro 341
CR (consistent read) 232
CR (Create Recovery Optimized) 172, 174
CRE (consistent read explicit) 232
creating
PDSE member
BSAM (basic sequential access method) 441
cross
region sharing 203
system sharing 204
cross reference table
direct data sets 536
cross-memory mode 155
cross-region sharing 201, 208
cryptographic option 65
Cryptographic Unit Support 66, 68, 69, 70
CSA (common service area) 192
Customer Information Control System (CICS) 225
cylinder
index
calculating space requirements 553
definition 547, 549
overflow
calculating space 553, 557
defined 550
specifying size 553
tracks 553
CYLINDERS parameter 109, 131, 134
CYLOFL parameter 551, 553

D

DASD (direct access storage device)
architecture 527
characteristics 329
control interval size 160
data set
erasing 63
indexed sequential 547
data set input 371
device selection 191
Hiperspace buffer 212
record format 309
shared 369
track capacity 160
data
compressed format 397
control block 509
control interval 276
DASD
erase-on-scratch 62
encryption standard 65
event control block 487
integrity
passwords 57, 62
protection 55
RACF (Resource Access Control Facility) 55
lookaside facility 395
data area
prime 548
data buffers
nonshared resources 178
data class
attributes
Dynamic Volume Count 42
Reduce Space Up To 42
Space Constraint Relief 42
definition 27
examples 330
multivolume VSAM 40, 111
data component
processing 149
Data Control Block Closed When Error Routine Entered condition 496
data control interval 161

data control interval *(continued)*
VSAM RLS CF caching 161, 224
data conversion 11
data management
quick start 314
data mode 346
DATA parameter 108, 121, 131
data set
closing
non-VSAM 332, 338
compatible characteristics 382
concatenation
like attributes 427
partitioned 426
concatenation, partitioned 466
control interval access 191
conversion 467, 468
description 310
name sharing 196, 199
RECFM 289, 307, 310
resource pool, connection 215
reusable 118
security 55
space allocation
indexed sequential data set 560
SYSIN 340
SYSOUT 340
temporary
allocation 268
names 268
unlike characteristics 382, 426
unopened 214
VIO maximum size, SMS managed 39
data sets
adding
records 167
allocation types 441
attributes, component 114
buffers, assigning 341
characteristics 3
checkpoint (PDSE) 435
checkpoint security 369
compress 103
compressed format 322
UPDAT option 386
concatenation
like attributes 379
unlike attributes 384
conversion 411, 585
copy 66
DASD, erasing 63
direct 535
discrete profile 56
DSORG parameter 328
duplicate names 107
exporting 50
extended, sequential 45
free space, altering 168
guaranteed space 40
improperly closed 53
ISMF (interactive storage management facility) 439
data sets *(continued)*
KSDS structural analysis 235
linear extended format 114
loading (VSAM) 115, 118
loading VSAM data sets 168
maximum size 39
maximum size (4 GB) 75
multiple cylinders 112
naming 23
non-system-managed 32, 34
nonspanned records 78
open for processing 139
options 313
organization
indexed sequential 546
organization, defined 4
password 58
read sharing (recoverable) 228
read/write sharing (nonrecoverable) 228
record loading
REPRO command 115, 116
recovery 51
recovery, backup 47
request access 143
restrictions (SMS) 29
routing 373, 375
RPL access 140
security 62
sequential
overlapping operations 386
sequential (extend) 387
sequential and PDS
quick start 314
sequential concatenation 379
shared
cross-system 209
shared (search direct) 371
sharing 195
sharing DCBs (data control block) 544
small 112
space allocation 437
indexed sequential data set 553
PDS (partitioned data set) 409, 410
specifying 37, 45
space allocation (DASD volume) 411
space allocation (direct) 536
spanned records 79
summary (VSAM) 94
SYSIN 373, 375
SYSOUT 373, 375
SYSOUT parameter 374
system-managed 31
tape 57
temporary (BDAM, VIO) 538
type
VSAM 84, 104
VIO (virtual I/O) 22
VSAM processing 137
data synchronization 483
DATACLAS keyword 473, 475
DATACLAS parameter 30, 377

DATAKEYFILE parameter 68
DATATEST parameter 235, 236, 238
DATATYPE option 454
DB2 striping 114
DBB-based (dictionary building blocks) 397
DBCS (double-byte character set)
  character codes 533
  printing and copying 533
  record length
    fixed-length records 533
    variable-length records 534
  SBCS strings 533
  SI (shift in) 533
  SO (shift out) 533
DCB (data control block) 485
  ABEND exit
    description 505
  ABEND exit, options 505
  ABEND installation exit 508
  address 331
  allocation retrieval list 504
  attributes of, determining 313, 331
  changing 331
  creation 313
  description 313, 320
  dummy control section 331
  exit list 501
  fields 313
  Installation OPEN exit 509
  modifying 314
  OPEN exit 509
  parameters 322
  sequence of completion 319
  sharing 544
  sharing a data set 365
DCB (DCBLIMCT) 539
DCB macro 389, 411, 457, 537
DCBBLKSI (without LBI) 394
DCBD macro 331, 423
DCBE macro
  IHADCBE macro 332
  LBI (large block interface) 423
  MULTACC parameter 391
  MULTSDN parameter 391
  non-VSAM data set 313
  number of stripes 400
  parameters 322
  PASTEOD=YES 369
  PDSs and PDSEs 16
  performance with BSAM and BPAM 391
  sharing 365
DCBEBLKSI (with LBI) 394
DCBLPDA field 574
DCBLRECL field 293, 346
DCBNCRHI field 574
DCBOFOPN (test) 337
DCBOFPPC bit
  set 380
DCBPRECL field 293
DCBRELAD address
  DCB (data control block) 417
DCBSYNAD field 331
DD statement
  name sharing 196
DDM (distributed data management) 22, 51
DDSR option 63
DE services (directory entry services)
  DESERV 446
  DESERV DELETE 465
  DESERV GET 447
  DESERV GET_ALL 449
  DESERV GET_NAMES 450
  DESERV RELEASE 451
  DESERV RENAME 465
  DESERV UPDATE 452
  updating member (DESERV UPDATE) 453
DE Services (directory entry services) 414, 416
DEB (data extent block)
  ISAM interface 584
DECB (data event control block) 386
  contents 487
  exception code 486, 492
  SYNAD routine 353
  update restrictions
    PDS 424
    PDSE 464
DECIPHER parameter 67
  Cryptographic Unit Support Facility 70
  Programmed Cryptographic Facility 70
deciphering data 65
DEFER parameter 518
deferred
  requests by transaction ID 217
  write requests 216
  writing buffers 217
deferred delete 465
DEFINE ALTERNATEINDEX command 105, 133
DEFINE CLUSTER command 29, 67, 105, 106, 120, 130
DEFINE CLUSTER|ALTERNATEINDEX command 118
DEFINE command 30, 58, 59, 62, 76, 109, 120, 121, 123, 131, 134, 159, 260
  creating a generation data group 478
  free space 166
  GENERATIONDATAGROUP 476
DEFINE commands 117
DEFINE NONVSAM command 14
DEFINE PAGESPACE command 124, 125
DEFINE PATH command 103, 105, 124, 133
DEFINE USERCATALOG command 130
DELAY option 337
DELETE command 14, 63, 125, 127, 267
deleting a member
  STOW, DESERV DELETE 465
delimiter parameters 67
DEQ macro 204, 366, 367
  sharing BISAM DCB 563
DES (data encryption standard) 65
DESERV
  DELETE 465
  FUNC=UPDATE macro 434
  GET 447, 448, 452

DESERV *(continued)*
GET_ALL 449
GET_NAMES 450
macro 431, 446
RELEASE 451
UPDATE 452
determinate errors 336
DEVD parameter 375, 537
specifying 308
device
block size
maximum 342
control for sequential data sets 479, 482
dependent
macros 389
dependent macros 479, 483
direct access, storage 309
independence
DCB subparameters 389
PDSE 431
sequential data set 387
sequential data sets 389
unit-record devices 389
record format
summary 307
type
DEVD parameter (DCB macro) 308
DEVSUPxx member of SYS1.PARMLIB
block size limit 315, 324
compression 310
DEVTYPE macro 87, 384, 572
INFO=AMCAP 323
USS files 402
DFA (data facilites area) 324
DFM (distributed file manager) 4, 22, 28, 51
DFR option 200, 216
DFSMS Data Set Services 47
DFSMS Hierarchical Storage Manager 47
DFSMSdss
COPY
PDS 467
PDSE 467
COPY (high-level qualifier) 467
migrate, recall 71
PDSE back up 468
space, reclaim 468
DFSMSdss COPY
convert PDS to PDSE 411
convert PDSE to PDS 411
DFSMShsm
dump, restore 71
management class 27
PARTREL macro 334
space release 439
DFSMSrmm
management class 27
security classes 64
VTS (Virtual Tape Server) 27, 29
DFSORT work data sets 396
DIAGNOSE command 126
diagnostic trace 181
direct
processing control interval size 163
direct access
device selection 191
scheduling buffers 177
storage device architecture 527
volume
device characteristics 329
labels 527
RECFM 307, 310
record format 309
write validity check 329
direct access buffers 175
direct addressing 536
with keys 537
direct bias 172
direct data sets
adding records 542
processing 535
tape-to-disk update 543
direct insertion 145, 146
directory block (PDS) 406
directory entry (PDS) 407
directory level sharing 461
disability 611
discrete
profile 56
DISP parameter 20, 366, 418, 460, 517, 568
description 340
passing a generation 476
shared data sets
indexed sequential 563
DIV (data-in-virtual) 5
DIV macro 91, 98, 146
DLF (data lookaside facility) 181, 395
DLVRP macro
delete a resource pool 215
DO (Direct Optimized) 172, 173
random record access 171
DSAB chain 503
DSCB (data set control block)
data set label 527, 531
description 530
index (format-2) DS2HTRPR field 574
model 473, 474
security byte 61
DSECT statement
DCB 331
DCBE 332
DSNAME parameter 132, 417
DSNTYPE parameter 32, 34, 328, 434, 440
DSORG parameter 328, 333, 410, 417, 419, 537
indexed sequential data set 550
dummy
control section 331
records
direct data sets 540
DUMMY option 116
Duplicate Record condition 496
Duplicate Record Presented for Inclusion in the Data Set condition 489

DW (Direct Weighted) 171, 173, 174
DYNALLOC macro 30, 440
  SVC 99 parameter list 314
dynamic
  buffering
    ISAM data set 551, 562
dynamic allocation 35
dynamic buffering
  direct data set 536
Dynamic Volume Count attribute 42, 43

E

EBCDIC (extended binary coded decimal interchange code)
  conversion to/from ASCII 356, 359
  data conversion 11, 15
  record format dependencies 310
EBCDIC (extended binary coded decimal Interchange code)
  label character coding 11
ECB (event control block)
  description 487, 489
  exception code bits 489
empty sequential data set 336
ENCIPHER parameter 67
  Cryptographic Unit Support Facility 70
  Programmed Cryptographic Facility 70
enciphering data 65
encryption 68
end of
  sequential retrieval 494
end-of-file
  software 186
end-of-file mark 411
end-of-volume
  exit 381
  processing 384
ENDREQ macro 137, 143, 149, 150, 153, 154, 199
ENQ macro 210, 366, 367, 540
entry-sequenced 186
ENVIRONMENT parameter 116
EOD (end-of-data)
  restoring values (VERIFY) 53
EODAD (end-of-data-set) routine
  BSP macro 481
  changing address 331
  concatenated data sets 427
  description 493
  EODAD routine entered 493
  exit routine
    EXCEPTIONEXIT 244
    JRNAD, journalizing transactions 245
  indicated by CIDF 184
  processing 379
  programming considerations 244, 493
  receives control 459
  register contents 244, 245, 493
  specifications 493
  user exit 156
EODAD parameter 485
EOV (end-of-volume)
  defer nonstandard input trailer label exit 510
  EODAD routine entered 493
  forcing 340
  processing 207, 338, 340
EOV function 466
ERASE macro 137, 143, 149, 150
ERASE option 63
ERASE parameter 110, 127
erase-on-scratch
  DASD data sets 62, 63
  RAMAC Virtual Array 63
EROPT (automatic error options)
  DCB macro 499
error
  analysis
    logical 252
    physical 255
    register contents 497, 498
    status indicators 487
    uncorrectable 494
  conditions 581, 582
  determinate 336
  handling 362
  handling deferred writes 218
  indeterminate 336
  KSDS (key-sequenced data set) 235
  multiple regions 206
  structural 235
error analysis
  exception codes 490
  options, automatic 499
  status indicators 492
ERRORLIMIT parameter 237
ESDS (entry-sequenced data set) 6
  alternate index structure 101
  insert record 144
  processing 86
  record access 97
  sequential (non-VSAM) data sets 85
ESDS (entry-sequenced data sets) 84
ESETL (end-of-sequential retrieval) macro 494
  description 577
ESETL macro 367, 493
ESTAE exit 544
EVENTS macro 353, 356, 391, 392
EXAMINE command 126, 235, 237, 240
exception
  code 486, 492
  exit routine
    I/O errors 244
    register contents 245
exception code bits
  BDAM (basic direct access method) 490
EXCEPTIONEXIT parameter 110
exchange buffering 390
exclusive
  control
    deadlocks 199
    nonshared resources 197
    release 199

exclusive control
direct data sets 540
sharing a direct data set 367
EXCP macro 21
EXCPVR macro 21
existing, change
BSAM applications 400
QSAM applications 400
exit
MVS router (SAF) 371
exit list
programming conventions 504
restrictions 504
exit list, create 138
exit routine
block count 510, 511
DCB
abend 505, 508
defer nonstandard input trailer label 510
end-of-data-set 493
EODAD 243
example 256
exception exit 244
JRNAD 245
LERAD 252
returning to main program 243
RLSWAIT 253
standard user label 514, 517
SYNAD
analyzing physical errors 255
UPAD 257
user
written 241
exit routines
DCB (data control block) 485
EXLST macro 137, 138, 142, 241, 503
exception processing 241
parameter
VSAM exit locations 241
EXLST parameter 139, 485, 501
expiration date processing 521
EXPORT command 47, 50
exporting a data set 50
EXTEND option 45
EXTEND parameter 320, 321, 568
extended format data sets
allocating 32
closing 399
compressed format 397
defining 95
DS1LSTAR, DS1TRBAL 400
free space 400
Hiperbatch, cannot use with 395
ICI (improved control interval access) 191
maximum (123 extents) 396
MULTACC option 391
opening 399
read, short block 394
restrictions 95
sequential data striping
data class 399
extended format data sets *(continued)*
sequential data striping *(continued)*
storage class 399
storage format 5
striped data sets 395
types of 95
extended logical record interface 306
extended search option 539
extent
concatenation, limit 466
extent growth 439
extents
concatenation limit 426
defined 40
maximum 40

F

facility class
IHJ.CHKPT.volser 369
FCB (forms control buffer) 480, 511, 513
image (SETPRT) 480
overflow condition 480
feedback
BDAM READ macro 355
feedback option 540
FEOV macro
concatenation 380, 382
DCB OPEN exit routine 339
end-of-data-set routine 493
EOV (end-of-volume), forcing 340
extending to another DASD volume 43
nonspecific tape volume mount exit 517
QSAM locate mode 349
QSAM spanned records 295
reading a PDS directory 427
reading a PDSE directory 466
SYNAD routine 499
field
control values
nonspanned data sets 187
relative record data sets 190
spanned records 189
display contents 143
FIFO (first-in-first-out) 314
HFS data sets 31
file access exit 337
file system
create 72
FILEDATA parameter 19
FIND macro 16, 367, 406, 411, 413, 423, 431, 445, 451, 453, 455, 456, 459, 493
description
PDS (partitioned data set) 417
first-in-first-out (FIFO) 314
fixed-length
record (unblocked) 393
RRDS (relative-record data set) 92
fixed-length records 439
description 290, 302
members (PDSE/PDS) 291

fixed-length RRDS 186
FOR (file owning region) 225
format
control interval
definition field 186
index
entry 282
record 279
record entry 281
record header 279
format-D records
block prefix 302
tape data sets 302
variable-length spanned records 304
format-F records 536
card reader and punch 311, 312
description 290, 302
ISO/ANSI tapes 300, 302
format-S records
extended logical record interface 306
segment descriptor word 305
format-U records
card reader and punch 311
description 307
ISO/ANSI tapes 307
format-V records
BDW (block descriptor word) 293
card punch 311, 312
description 292, 297
record descriptor word 293
segment control codes 295
segment descriptor word 295
spanned 294, 297
forms control buffer 511
fragmentation 439
FREE command 439
free control interval 281
free space
altering 168
DEFINE command 166
determining 166
optimal control interval 167
performance 166
threshold 167
FREE=CLOSE parameter 338
FREEBUF macro 347, 351, 563, 564
buffer control 341
description 352
FREEDBUF macro 544, 564
FREEMAIN macro 344, 345, 504, 508
FREEPOOL macro 311, 312, 335, 342, 345
FREESPACE parameter 89, 110
full access, password 140
full page increments 438
full-track-index write option 551

G

gaps
interblock 289
GDG (generation data group)
absolute, relative name 469
allocating data sets 473, 476
building an index 478
creating a new 473, 477
deferred roll-in 476
entering in the catalog 469, 471
ISAM data set 476, 477
limits 476
naming conventions 472
retrieving 477, 478
ROLLOFF/expiration 477
GDS (generation data set)
absolute, relative name 469
activating 476
passing 476
roll-in 476
GENCB ACB macro 198
GENCB macro 137, 139, 140, 142, 143, 175, 263
general registers 519
generation
index, name 469
number
relative 469, 473
generic
profile 56
generic key 148
GET macro 211, 298, 340, 360, 380, 382, 385, 386, 393, 423, 425, 493, 496
description 359
parallel input 361
GET_ALL function 449
GET-locate 347, 350
pointer to current segment 378
GETBUF macro 347, 351
buffer control 341
description 351
GETIX macro
processing the index 275
GETMAIN macro 341, 508
GETPOOL macro 324, 341, 344, 345, 351, 571
buffer pool 342
global resource serialization 201
global shared resources 212
GRS (global resource serialization) 201, 365
GSR (global shared resources) 96, 103, 152
control block structure 200
subpool 212
GTF (generalized trace facility)
extended format 396
VSAM 181
guaranteed
SPACE attribute 41, 42
DASD volumes 40
synchronous write 483
guaranteed space attribute 83

H

header
index record 279

header *(continued)*
  label, user 514, 517, 530
HFS data sets
  requirements 71
  restrictions 71
HFS files
  access 7
  BSAM access 5
  DEVTYPE macro 402
  FIFO special files 31
  planning 71
  status 20
  type of USS file system 18, 21
  VSAM access 6
Hiperbatch
  DLF (data lookaside facility) 395
  not for extended format data set 95, 395
  performance 395
  QSAM 15
Hiperspace 15
  buffer
    LRU (least recently used) 173
    LSR 212
  SMBHWT 171
HOLD type connection 414
horizontal pointer index entry 280

I

I/O (input/output)
  buffers
    managing with shared resources 216
    sharing 211
    space management 169
  control block sharing 211
  error recovery 364
  journaling of errors 218
  overlap 353
  sequential data sets, overlapping operations 386
I/O data sets, spooling 373
I/O status indicators 486, 492
IBM 3380 Direct Access Storage
  drive 388
IBM standard label (SL) 57, 527
ICFCATALOG parameter 131
ICI (improved control interval access)
  ACB macro 191
  APF (authorized program facility) 191
  cross-memory mode 155
  extended format data sets 191
  MACRF option 200
  not for compressed data set 96
  not for extended format data set 95
  SHAREOPTIONS parameter 205
  UPAD routine 258
  user buffering (UBF) 190
  using 191
  VSAM 155, 192
ICKDSF (Device Support Facilities) 364, 527
IDCAMS print 87
IDRC (Improved Data Recording Capability) 321
IEBCOPY
  compress 429
  compressing PDSs 405, 406, 426
  convert PDS to PDSE 411
  convert PDSE to PDS 411
  fragmentation 439
  PDS to PDSE 467
  PDSE back up 468
  SELECT (member) 467
  space, reclaim 468
IEBIMAGE 480, 512
IEC034I message 383
IEC127D message 512
IEC129D message 512
IEC161I message 173
IEC501A message 518, 521
IEC501E message 518, 521
IEC502E message 521
IECOENTE macro 518
IECOEVSE macro 521
IEF630I message 441
IEFBR14 job step 114
IEHLIST program 574
IEHLIST utility 554, 585
IEHMOVE program 407, 408
IEHPROGM utility program
  generation data group
    building index 478
    ISAM data set 477
  ISAM data set, generation data group 477
  muiltivolume data set creation error 554
  PROTECT command 61
  SCRATCH control statement 63
IGWCISM macro 454
IGWDES macro 446
IGWLSHR parameter 463
IGWPMAR macro 450
IGWSMDE macro 414, 446, 447
IHAARL macro 504
IHADCB dummy section 331
IHADCBE macro 332, 400
IHAPDS macro 414, 448
IHJ000I message 95
imbedded index area 553, 554
IMPORT command 47, 50, 53, 54, 58
improved control interval (ICI) 258
Improved Data Recording Capability 310
Incorrect Record Length condition 496
INDATASET parameter 66, 135
independent overflow area
  description 550
  specifying 556
indeterminate errors 336
index
  access, with GETIX and PUTIX 275
  alternate 133
  alternate, buffers 179
  area
    calculating space 553, 554
    creation 546

index *(continued)*
component
control interval size 160
opening 275
control interval
data control area 276
size 163
split 284
control interval size 180
cylinder
calculating space 553
overflow area 550
data (separate volumes) 181
entry
data control interval relation 276
record format 281
spanned records 285
levels, VSAM 277
master
calculating space 553
creating 550
using 547
pointers 278
prime 276
processing 275
record
format 279
sequence set 278, 279
space allocation 546
structure 277
track, space 553
update 284
virtual storage 180
index area 547
index buffers 176
index options
replicating, imbedding 180
INDEX parameter 108, 121
index/overflow chain 496
indexed sequential access method 579
description 5
indexed sequential data set
adding records at the end 566, 568
allocating space 553, 560, 571, 574
areas
index 548, 550
overflow 550
prime 548
buffer
requirements 571
converting to VSAM
example 590
using access method services 585
using JCL 585, 589
creation 550, 553
deleting records 570
device control 574, 577
full-track-index write option 551
inserting new records 566, 567
ISAM 577
retrieving 561, 563
indexed sequential data set *(continued)*
updating 561, 563, 567
using BISAM 545
using QISAM 545
INDEXTEST parameter 236
description 235
EXAMINE output 238
testing 236
indirect addressing
randomizing, conversion 537
INFILE parameter 66, 131
INHIBITSOURCE parameter 50
INHIBITTARGET parameter 50
INOUT parameter 319, 482
INPUT parameter 319, 321
input/output devices 309
INSPECT command 364
installation exit
DCB OPEN 320
ISO/ANSI Version 3 or Version 4 tapes 337
interface program, ISAM 579
interval
control
split 284
INTOEMPTY parameter 51
invalid data set names 23
Invalid Request condition 488
invalidating data and index buffers 203
IOB (input/output block) 501
IOBBCT (load mode buffer control table) 501
IOBFLAGS field 501
IRG (interrecord gap) 289
ISAM (indexed sequential access method) 5, 501
converting from ISAM to VSAM 585, 590
interface
abend codes 583
DCB fields 586, 587
DEB fields 584
program 579
restrictions 588
processing a VSAM data set 580
SYNADAF macro example 591
ISAM data set
GDG (generation data group) 477
ISITMGD macro 454
ISMDTPGM constant 454
ISMDTREC constant 454
ISMDTUNK constant 454
ISMF (interactive storage management facility) 126, 439
ISO (International Organization for Standardization) 11
ISO/ANSI
control characters
format-D records 302
format-F tape records 300
tape
data conversion 11
fixed-length records 300
formats 299
undefined-length records 307
variable-length records 302

ISO/ANSI *(continued)*
Version 3 tapes 337
Version 4 tapes 337
ISO/ANSI standard label (AL) 57

J

JCL (job control language) 133
allocation examples
temporary VSAM data set 268
coding 265
converting from ISAM to VSAM 585
DD parameters 272, 329, 330
ISAM interface processing 586
non-VSAM data set 329, 330
VSAM data set 265
JCL keyword (DSNTYPE)
message IEF630I 441
JES (job entry subsystem) 373, 375
JFCB (job file control block) 319, 513
open function 318
JFCBE (job file control block extension) 513
JFCLRECL field 374
job control language (JCL) 133
journaling I/O errors 218
journalizing transactions
JRNAD exit 245
JRNAD exit routine
back up data 51
building parameter list 249
control interval splits 247
deferred writes 218
example 249
exit, register contents 246
journalizing transactions 139, 246
recording RBA changes 247
shared resources 218
transactions, journalizing 139
values 218

K

key
alternate 103
compression 284
front 282
rear 282
data encryption 68
field
indexed sequential data set 546
file, secondary 69
indexed sequential data set
adding records 566, 570
retrieving records 562, 567
RKP (relative key position) 551, 570, 571
track index 546, 549
key class
key prefix 574
key compression 91
key length
reading a PDSE directory 435
key-sequenced 186
keyboard 611
keyed direct retrieval 148
keyed sequential retrieval 146
keyed-direct access 98, 99
keyed-sequential access 97, 99
KEYLEN parameter 214, 309, 328, 435, 537
KEYRANGES parameter 110
KEYS parameter 109, 122, 134
KILOBYTES parameter 109, 131
KN (key, new) 564
KSDS (key-sequenced data set) 6, 84, 166
alternate index 100, 101
buffers 175
CI, CA splits 228
cluster analysis 235
control interval
access 183
data component 149
free space 163
index
accessing 275
processing 275
index options 180
insert record
description 91
sequential 144, 145
inserting records
description 88
logical records 88
record (retrieval, storage) 180
sequential access 97
structural errors 235
VSAM (virtual storage access method) 398
KU (key, update)
coding example 564
defined 355
read updated ISAM record 563

L

LABEL parameter 61, 321, 339, 515
label validation 337
label validation installation exit 322
labels
coding 11
direct access
DSCB 527, 531
format 527
user label groups 530
volume label group 528, 529
exits 514, 517
tapes, volumes 7
last-volume
extend 387
LBI (large block interface)
BLKSIZE parameter 322
block size merge 320
converting BSAM to LBI 423
DCB OPEN exit 315
determining BSAM block length 393

LBI (large block interface) *(continued)*
JCL example 384
like concatenation 384
performance 390
recommendation 315
requesting 323
system-determined block size 327, 392
using larger blocks
BPAM 323
BSAM 323
QSAM 323
writing short BSAM block 394
LDS (linear data set) 6
LEAVE option 333, 340
close processing 332
tape last read backward 339
tape last read forward 339
LERAD exit routine 156
error analysis 252
register contents 253
level sharing
directory, member 461
like
concatenation 381, 382
BSAM block size 382
data sets 379
DCB, DCBE 381
like concatenation 384
LIKE keyword 32, 96, 399, 440, 473, 475
linear data sets 84, 98, 113
processing 146
link field 571
link pack area (LPA) 445
linkage editor
note list 409
LISTCAT command 14, 105, 113, 130, 163, 213
LISTCAT parameter 132
load mode
BDAM (basic direct access method) 544
QISAM
description 546
sharing a DCB 367
loading
VSAM data sets
REPRO command 115, 116
local locking
non-RLS 229
local shared resources 212
locate
mode
parallel input 361
mode (QSAM) 464
locate mode 80, 346
buffers 350
QSAM (queued sequential access method) 378
records exceeding 32 760 bytes 296
lock manager (CF based) 223
locking unit 209
logical
error analysis routine 252
logical block size 439
logical end-of-file mark 434
logical record
control interval 90
length, SYSIN 374
lower-limit address 574
LPA (link pack area) 445
LRD (last record) 98
LRECL parameter 116, 132, 328, 359, 374, 551, 573
coding in K units 306
records exceeding 32 760 bytes 296
LRI (logical record interface)
QSAM 295
LRU (least recently used) 173
LSR (local shared resources) 15, 151, 172
buffering 171
Hiperspace buffers 212
pools
resource 212
specifying control block structure 200

M

machine control characters
described 308
MACRF parameter 211, 366, 425, 453, 464, 537, 564
ACB macro 198
control interval access 184
WAIT macro 357
MACRF=WL parameter 535
macros
buffering
basic access method 351
data management 12
device dependent 389
magnetic tape
direct access volumes
NOTE macro 389
magnetic tape volumes
labels
user 514, 517
positioning
close processing 332, 338
end-of-volume processing 339
management class
definition 27
examples 330
mark, end-of-file 411
mass sequential insertion 144
master index 550
MASTERPW parameter 134
maximum block size 439
maximum strings
255 152
MEGABYTES parameter 109
member
PDS (partitioned data set) 418
PDSE, add records 464
PDSE, address 453
PDSE, level sharing 461
member-level sharing 461
member, delete 465

members
adding, replacing
PDSE members serially 442
MGMTCLAS parameter 30
millisecond response 172
mixed processing 163
MKDIR command 72
MLA (multilevel alias) 107
MLT (Member Locator Tokens) 432, 433, 452
MMBBCCHHR
actual address 539
ISM 589
MODCB macro 98, 137, 142, 143, 263
mode
load 495
resume 568
locate, member (QSAM) 425
request execution
requirements 231
scan 496
model DSCB 474
MODEL parameter 109
modify
PDS (partitioned data set) 423
modifying members
PDSE 464
MOUNT command 72
move
mode
parallel input 360
move mode 346
PDSE restriction 322
MRKBFR macro
marking buffer for output 219
releasing exclusive control 199
MSWA parameter 573
MULTACC parameter 391
DCBE macro 315, 400
performance 391
TRUNC macro 356
WAIT macro 357
multiple
data sets
closing 335
opening 335
multiple step job 471
multiple string
processing 150
multiple system sharing (PDSEs) 462
multitasking mode
sharing data sets 367
multivolume
data sets
DASD (extending) 387
NOTE macro 481
RACF-protected 56
tape 56
unlike concatenation 385
MULTSDN parameter 315, 354, 391, 400
MXIG command 23

N

NAME parameter 131, 134
naming data sets 23
national 23
NCP (number of channel programs) 390
BSAM, BPAM 315
NCP parameter 391
BSAM 355
NEW parameter 132
NFS (Network File System) 3, 7, 18
NFS files
type of USS file system 18, 21
NL (no label) 57
NLW subparameter
MACRF parameter 198
no label (NL) 57
no post code 358
no user buffering (NUB) 184
NOCONNECT option 445
NOERASE option
AMS commands 63
non-CICS
VSAM RLS 228
non-RLS access 229
non-system-managed data set
allocating GDSs 475
non-VSAM
creating data set labels 485
error analysis 485
performing I/O operations (data sets) 485
requesting user totaling 485
system-managed data sets 33
non-VSAM data set
password 61
nonrecoverable file concept 227
NONSPANNED parameter 79
nonspecific tape volume mount exit
return codes
specifying 518
NONUNIQUEKEY parameter 134, 152
NOPWREAD parameter 61
NOREUSE parameter 109
NOSCRATCH parameter 127
NOTE macro 10, 17, 386, 389, 398, 408, 411, 423, 434, 455, 456, 481
NRI (no read integrity) 232
NSP (note string positioning) 98
NSR (nonshared resources)
Create Optimized (CO) 174
Create Recovery Optimized (CR) 174
Direct Weighted (DW) 174
job control language (JCL) 170
LSR (local shared resources) 171
NSR subparameter 152
Sequential Optimized (SO) 173
Sequential Weighted (SW) 172, 174
NSR subparameter 215
NTM parameter 550
NUB (no user buffering) 184
NUIW parameter 214
null data set 336

null record segments
PDSE (partitioned data set extended) 437

O

O/EOV (open/end-of-volume)
nonspecific tape volume mount exit 518
volume security/verification exit
described 521, 523
OAM (object access method) 5, 27
object
improved control interval access 191
OBJECT parameter 143
OBROWSE command 87, 403
OBTAIN macro 87, 572
offset reading 355
OPEN macro 386, 389, 411, 464
connecting program to data set 313
control interval processing 191
data sets 335
description 318, 322
EXTEND parameter 320
functions 318, 322
multiple 335
options 320, 322
parallel input processing 360
protect key of zero 345
resource pool, connection 215
OPEN UPDAT
positioning 462
OPTCD parameter 359, 389, 479, 523, 537, 540, 551
control interval access 184
master index 550
OPTCD=B
generate concatenation 385
OPTCD=C option 391
OPTCD=H
VSE checkpoint records 390
optimal block size 324
OUTDATASET parameter 66, 131, 135
OUTFILE parameter 66
OUTIN option 45
OUTIN parameter 482
OUTINX option 45
OUTINX parameter 482
output buffer, truncate 351
OUTPUT option 45
output stream 373, 375
overflow
area 547, 550
chain 568
PRTOV macro 480
records 550
overflow area 547
Overflow Record condition 489
overlap
input/output
performance 390
overlap I/O 424
overlapping operations 386

P

padded record
end-of-block condition 302
variable-length blocks 303
page
real storage
fixing 192
page size
physical block size 439
page space 124
PAGEDEF parameter 310
paging
excessive 180
paging operations
reduce 353
paper tape reader 312
parallel
input processing 360
parallel data access blocks (PDAB) 360
parallel sysplex-wide locking
DFSMS lock manager 229
partial release 112
partitioned table spaces (DB2) 114
PARTREL macro 63, 87, 334
PASS disposition 340
PASSWD parameter 58
password
access 58
authorization checking 62
authorize access 139
LABEL parameter 61
non-VSAM data sets 61
prompting 60
protection precautions 59
VSAM data sets 57
PATH parameter 19, 20, 31
path verification 53
PATHENTRY parameter 134
PATHOPTS parameter 20
PC (card punch) record format 311, 312
PDAB (parallel data access block) 523
PDAB (parallel data access blocks) 360
PDAB (parallel input processing) 360
PDAB macro 501
work area 360
PDF directory entry 407
PDS (partitioned data set)
adding members 411
concatenation 426, 427
converting to and from PDSE 411, 467
creating 410, 413
defined 5, 405, 407
directory 413
description 406, 409
processing macros 413
reading 427
directory (size limit) 406
directory, updating 418
locating members 413
macros 413, 418
processing 16

PDS (partitioned data set) *(continued)*
quick start 314
retrieving members 419, 423, 460
rewriting 425, 426
space allocation 409, 410
structure 405
updating member 423, 425
PDS and PDSE differences 431
PDSDE (BLDL Directory Entry) 448
PDSE (partitioned data set extended)
ABEND D37 438
allocating 32, 34, 440
block size 432
block size, physical 439
concatenation 466
connection 444
convert 467
convert (PDS, PDSE) 468
converting 411
creating 440, 444
data set compression 438
data set types 440
defined 5, 429
deleting 465
directory 446, 455
BLDL macro 444
description 431, 432
indexed search 430, 432
size limit 432
directory (FIND macro) 453
directory structure 433
directory, read 466
directory, update 457
DYNALLOC macro 440
extended sharing protocol 463
extents 426, 439
fixed-length blocked records 435
fragmentation 439
free space 439
full block allocation 438
integrated directory 438
logical block size 439
macros 444, 458
member
add record 464
retrieving 458
members
adding, replacing 442
members (multiple) 443
multiple system sharing 462
NOTE macro (TTRz) 455
null segments 296
OPEN macro options 322
performance 468
advantages 429
positioning 434
processing 16
reblocking 435
reblocking records 430
record
numbers 433
PDSE (partitioned data set extended) *(continued)*
record processing 434, 437
records (unblocked, blocked) 433
relative track addresses 432
rename 467
restrictions 434
rewrite 464
sharing 460
size
dynamic variation 431
space (contiguous, noncontiguous) 437
space allocation 437
space considerations 437
space reuse 432
storage requirements 440
switching members 456
SYNCDEV macro 483
TRUNC macro restriction 351
TTR (track record address) 11
unreclaimed space 438
update 453
updating 464
performance
buffering 329
control interval access 191
cylinder boundaries 334
DASD, tape 391
data lookaside facility 395
Hiperbatch 395
improvement 159, 181
sequential data sets 390
sequential data sets 353, 419
performance chaining 400
physical block size 160
physical errors
analysis routine 218
analyzing 255
physical sequential data sets 511
maximum (16 extents) 396
PIPE (DSNTYPE) 31
POINT macro 10, 17, 98, 137, 143, 146, 147, 315, 386, 389, 398, 423, 451, 455, 459, 482, 493
retrieving records
VSAM 211
pointers 278
pool
resource
size 213
resources
shared 212
position volumes 340
positioning
direct access volume (to a block) 482
member address
FIND macro 417
PDS (partitioned data set) 417
sequential access 211
tapes (to a block) 482
volumes
magnetic tape 338, 340
post code 358

POST macro 489
prefix, key 574
primary
  space allocation 40
prime data area
  description 547
  space allocation 553, 555
prime index 91, 276, 278
PRINT command 14, 105, 127, 130, 135
print DBCS characters 533
PRINT parameter 132
printer
  overflow (PRTOV macro) 480
  record format 310
processing
  adding, replacing
    multiple PDSE members 443
  members (PDSE) 444
  modes 350
  open, close, EOV 337
Product-sensitive Programming Interface 183
profile
  generic, discrete 56
program library
  binder 442
Programmed Cryptographic Facility 66, 68, 69, 70
programming conventions 504
protect key of zero
  buffer pool 345
PROTECT macro 61, 62
protection
  access method services
    cryptographic option 65, 71
    offline data 65
  APF 62, 64
  data sets 55, 71
  deciphering data 65
  enciphering data 65
  non-VSAM data set
    password 61, 62
    RACF 56
  password 57, 62
  RACF 55
  system-managed data sets 57
  VSAM data set
    password 57, 60
    RACF 55, 63
PRTOV macro 389, 391, 480
PUT
  OPTCD=NUP 190
PUT macro 298, 411, 434, 496, 499
  control interval
    update contents 185
  deferring write requests 216
  description 359, 360
  indexed sequential data set 568, 569
  linear data set 144
  locate mode 346, 350
  relative record data set 144
  simple buffering 347, 350
  update record 149
PUT-locate 346
PUTIX macro
  processing the index 275
PUTX macro 298, 366, 386, 423, 425, 461
  description 360
  simple buffering 347, 350

Q

QISAM (queued indexed sequential access method)
  data set (EODAD routine) 494
  data set (SYNAD routine) 494
  ECB (event control block)
    conditions 495, 496
    exception code bits 494
  error conditions 581
  I/O status information 486
  load mode 501
  processing an indexed sequential data set 545
  scan mode 563
  sequential data sets 377
  sharing 366, 367
QSAM (queued sequential access method) 6, 385
  BUFNO
    chained buffers 391
  BUFNO parameter 391, 400
  creating (PDSE) 441
  defined 359
  direct data set restrictions 537
  extended format data sets
    sequential data striping 399
  Hiperbatch 15
  I/O status information 486
  like concatenation 384
  move mode 378
  parallel input processing 360, 523
  performance improvement 390
  printer control 311
  processing modes 346, 347
  read (PDSE directory) 431, 466
  reading
    PDS directory 427
  retrieving
    PDSE member 458
  sequential data sets 380
  sharing a data set 366, 367
  short block 436
  spanned variable-length records 295
  update
    PDSE member 464
  updating
    PDS directory 418
    PDS member 425
  user totaling 523
  using buffers 347
  USS files 401
queued access method
  buffer
    control, pool 341
  buffering macros 351
  sequential data set 351

quick reference
accessing records 353
backup, recovery 47
CCSIDs 593
data control block (DCB) 313
data sets, introducing 3
DBCS, using 533
direct access labels, using 527
direct access volume, space 37
direct data sets, processing 535
generation data groups, processing 469
HFS data sets 71
I/O device control macros 479
indexed sequential data sets 545
ISAM program, VSAM data sets 579
JCL, VSAM 265
KSDS Cluster Errors 235
KSDS, index 275
non-VSAM data sets, RECFM 289
non-VSAM data sets, sharing 365
non-VSAM user-written exit routines 485
PDS, processing 405
PDSE, processing 429
protecting data sets 55
sequential data sets 377
sharing resources 211
spooling and scheduling data sets 373
using SMS 27
VSAM data set
define 105
examples 129
organizing 75
processing 137
sharing 195
VSAM performance 159
VSAM RLS 223
VSAM User-Written Exit Routines 241
quick start
data sets
sequential, PDS 314

R

R0 record
capacity record data field 537
RACF (Resource Access Control Facility)
alter 55
checkpoint data sets 369
control 55
DASD data sets, erasing 63
DASDVOL authority 527
data sets 140
erase DASD data 62
protection 56
RACF command 57
read 55
update 55
z/OS Security Server 55
RAMAC Virtual Array 63
randomizing
indirect addressing 537
RBA (relative byte address) 84, 85, 96, 97, 112, 116, 139, 141, 147, 148, 151, 154
JRNAD
parameter list 218
recording changes 247
locate a buffer pool 219
RBA (relative record number)
slots 186
RBN (relative block number) 10, 398
RD (card reader) 311, 312
RDBACK parameter 320, 321, 517
RDF (record definition field) 116
format 186
free space 91
linear data set 91
new record length 90
records 76
slot 92
structure 187, 189
RDJFCB macro
allocation retrieval list 485, 504
BLKSZLIM retrieval 324
JFCB exit 513
RDW (record descriptor word) 391
data mode exception
spanned records 293
description 293
extended logical record interface 306
prefix 303
segment descriptor word 295
updating indexed sequential data set 566
variable-length records format-D 303, 305
read
access, password 140
backward, truncated block 291
forward
SF 389
integrity, cross-region sharing 202
READ macro 298, 354, 366, 385, 386, 389, 393, 423, 424, 493, 500, 564
basic access method 355
block processing 353
BSAM 357
description 355
direct data set 355
existing records (ISAM data sets) 563
spanned records, keys 355
read short block
extended format data set 394
READPW parameter 134
real buffer address 343
reblocking
records
PDSE 430
reblocking records
PDSE (partitioned data set extended) 435
RECATALOG parameter 109
RECFM (record format)
fixed-length 302
ISO/ANSI 302
magnetic tape 309

RECFM (record format) *(continued)*
parameter
card punch 311, 312
card reader 311, 312
sequential data sets 289
sequential data sets 309
spanned variable-length 294, 297
undefined-length 307
variable-length 292, 304
RECFM parameter 328
RECFM subparameter 290
recommendations
block size calculation 37
catalog, analyzing 237
EXAMINE command, use VERIFY before 237
record
access
KSDS (key-sequenced data set) 97, 100
access, password 140
access, path 152
adding to a data set 167
average length 410
block
boundaries 401
control characters 307
control interval size 160
data set address 9
definition field 76, 186
deleting 149
descriptor word (see BDW) 293
direct data sets 542
direct retrieval 148
ESDS (entry-sequenced data set) 144
fixed-length
full-track-index write option 551
parallel input 360
restrictions 291
format
device type 307
fixed length 291
fixed-length 290
ISO/ANSI 299
variable length 292
format (fixed-length standard) 290
free space 167
index
format 279
header 279
length 279
set 280
index, replicate 110
indexed sequential data set 566, 569
insert, add 144
insertion
free space 167
insertion, path 152
ISAM 356
KSDS 88
KSDS (key-sequenced data set) 144
length
DBCS characters 533
record *(continued)*
length (LRECL parameter) 328
logical 295
longer than
32 760 296
maximum size 160
nonspanned 187
number
PDSE 433
padded 302
parallel input
processing 360
restrictions on fixed-length format 291
retrieval
sequential 359, 576
retrieve 146
data set 360
rewrite, PDSE 464
segments 296
sequence set 280
sequential data set (add, modify) 386
sequential data set (add) 387
sequential retrieval 146
spanned 289, 294
basic direct 296
index entries 285
RDF structure 189
segments number 189
variable-length 360
spanned format-V 296
spanned, QSAM processing 295
undefined-length 298
parallel input 360
variable-length
format-V 360
parallel input 360
variable-length RRDS insertion 145
variable-length, sequential access method 294
VSAM sequential retrieve 147
VSE checkpoint 390
write, new 356
XLRI 389
record descriptor word (RDW) 391
record key 574
Record Length Check 488
Record Locator Tokens (RLT) 433
record locks
share and exclusive 230
record management 16
Record Not Found 491
RECORDS parameter 109
RECORDSIZE parameter 67, 109, 110, 122, 134
control interval 160
RECORG keyword 434
RECORG parameter 132, 133
recoverable file concept 227
recovery
EXPORT/IMPORT 50
program (write) 51
RECOVERY option 172, 174
RECOVERY parameter 117

recovery procedures 47
recovery requirements 507
Reduce Space Up To % attribute 42
register
  contents
    SYNAD exit routine 492, 498
RELATE parameter 121, 134
relative
  block address
    direct data sets 539
    feedback option 540
  byte address 219
  generation name 469, 473
  key position (RKP) parameter 551, 570
  track address
    direct access 539
    feedback option 540
relative address
  DASD volumes 10
relative block addressing 535
relative generation number 471
relative track addressing 535
release
  control interval 199
RELEASE command 334
release, partial 439
RELEX macro 544
  READ request 540
RELSE macro 341, 344, 351
RENAME macro 87
reorganization
  ISAM data set 569
  ISAM statistics 546
REPLACE parameter 49
REPRO command 47, 54, 67, 70, 105, 115, 116, 119, 130, 131
REPRO DECIPHER parameter 66
REPRO parameter 118
request
  macros 15
requirements
  HFS data sets 71
  REPRO command
    DECIPHER function 70
    ENCIPHER function 70
REREAD option 332
  tape last read backward 339
  tape last read forward 339
RESERVE macro 204, 205
Resource Access Control Facility 55
resource pool
  building 211
  connecting 215
  deferred writes 216
  deleting 215
  statistics 214
  types 212
restrictions
  alternate index, maximum nonunique pointers 100
  CCSID conversion 300
  CNTRL macro 479
restrictions *(continued)*
  compressed-format data set
    UPDAT option 386
    update-in-place 386
  concatenation of variable-blocked spanned data set 385
  control interval access
    compressed data sets 183
    key-sequenced data sets 183
    variable-length RRDSs 183
  data sets
    control interval access 183
    extended forma 95
  data sets, SMS-managed 29
  DEVTYPE macro 573
  Direct Optimized technique 173
  exit list 504
  extended format data sets 95
  fixed-length record format 291
  generation data set, model DSCB 473, 474
  HFS data sets 71
  IDRC mode 321
  JOBCAT statement 585
  JRNAD exit, no RLS support 245
  load failure 117
  name segment length 24
  note list 409
  PDSE
    alias name 434
    converting 468
    processing 434
  PRINT command, input errors 127
  sharing 588
  sharing violations 461
  SMS
    absolute track allocation 39
    ABSTR value for SPACE parameter 39
  STEPCAT statement 585
  system-managed data set 585
  tapes, Version 3 or 4 322
  TRKCALC macro 573
  UPDAT option, compressed-format data set 386
  update-in-place, compressed-format data set 386
  USS files 402
  USS files, simulated VSAM access 87
  VSAM data set processing 7
  VSAM data sets
    concatenation not allowed in JCL 16
  VSAM, space constraint relief 42
resume load mode
  extending
    indexed sequential data set 553
  partially filled track or cylinder 553
  QISAM 546
resume loading 550
retained locks
  non-RLS access
    SHCDS PERMITNONRLSUPDATE 230
retention period 330
RETPD keyword 330

retrieve
  sequential data sets 378
retrieving
  generation data set 477, 478
  PDS members 419, 423
  PDSE members 458, 460
  records
    directly 562
    sequentially 561
RETURN macro 499, 510, 511
REUSE parameter 66, 109, 118, 122
REWIND option 332, 340
rewriting
  PDS (partitioned data set) 426
RKP (relative key position) parameter 551, 570
RLS (record-level sharing) 105, 119, 223
RLS access 229
RLS CF caching 161, 224
RLSE parameter 334, 439
RLSE subparameter 112
RLSWAIT exit 254
RLSWAIT exit routine 253
RLT (Record Locator Tokens) 433
routine
  exit
    VSAM user-written 241
RPL (request parameter list) 151
  coding guidance 242
  create 140
  exit routine correction 243
  parameter 184
  transaction IDs 217
RPL macro 137, 142
RRDS (relative record data set) 6, 145
  free space 110
  hexadecimal values 190
  variable-length 160
RRDS (relative-record data set) 84, 92, 99
RRN (relative record number) 49, 92, 115

S

SAA (Systems Application Architecture) 22
SAM (sequential access method)
  buffer space 341
  null record segments 437
SBCS (single-byte character set) 533
scan mode 546, 563
SCHBFR macro
  description 219
SCRATCH macro 87
  IEHPROGM utility program 63
scratch tape requests
  OPEN or EOV routines 485
SDR (sustained data rate) 83
SDW (segment descriptor word)
  conversion 305
  description 295
  format-S records 305
  format-V records 295
  location in buffer 355
secondary
  space allocation 40, 82, 83
security
  APF protection 55, 62, 64
  cryptographic 55, 64, 71
  O/EOV security/verification exit 521, 523
  password protection 55, 57, 62
  RACF protection 55
security (USVR) 260
segment
  buffer 341, 346
  control code 295
  descriptor word
    indicating a null segment 296
    spanned records 295
  null 296
  PDSE restriction 296
Sequence Check condition 496
sequence set record
  index entries 278
sequence-set record
  format 279
  free-control-interval entry 280
  index entries 280
  RBA 280
sequential
  access
    RRDS 99
  processing control interval size 163
sequential access buffers 179
sequential bias 172
sequential concatenation
  data sets 379
  read directories sequentially
    PDSs/PDSEs 466
  USS files 403
sequential data set
  concatenation 426
  device
    control 479, 482
  modify 386
  queued access method 351
  update-in-place 386
sequential data sets 377
  chained scheduling 390
  device
    independence 387, 389
  modify 387
  quick start 314
  read 393
  record
    retrieve 378, 379
  record length 393
sequential data striping
  extended format data sets 399
  migrating extended format data sets 400
sequential extended format data sets
  compressed format 397
sequential insertion 144, 145, 146
sequential millisecond response 172

serialization
SYSZTIOT resource 503
serializing requests 210
serially
adding, replacing
PDSE members 442
server address space 223
service request block (SRB) 191
SETL macro 367, 494, 574, 576, 577
specifying 574
SETPRINT macro 485
SETPRT macro 375, 480, 512, 513
SETROPTS command 63
share options 205, 229, 265
shared
cross-system 204
data sets 204
cross-region 208
non-VSAM 365, 368
PDSE 430
data sets (PDSE) 460
resources
among data sets 211
I/O buffers 216
JRNAD exit 218
restrictions 220
subtasks 196, 201
shared DASD
checkpoint data sets (RACF) 369
SHAREOPTIONS parameter 110, 123, 153, 205
sharing
DCBs 461
multiple jobs/users 461
sharing protocol 463
sharing rules 461
shift in (SI) 533
shift out (SO) 533
short block
magnetic tape 291
reading 394
write 394
short block processing 436
short form parameter list 335
shortcut keys 611
SHOWCAT macro 213
description 214
SHOWCB macro 137, 142, 143, 213
action requests 217
buffer pool 214
SHRPOOL parameter 215
SI (shift in) 533
SIGPIPE signal 20
single unique connect identifier 415
size
control
area 164
interval 159
data control interval 161
dynamic variation
PDSE index 431
index control interval 163
size *(continued)*
limits
control interval 159
skip-sequential access 98, 99
fixed-length RRDS 145
variable-length RRDS 145
SL (IBM standard label) 57
SMB (system-managed buffering) 41, 175
requirements 170
SMBDFR
buffers, write 171
SMBHWT parameter 173
SMBHWT, Hiperspace buffers 171
SMBVSP
virtual storage 171
SMBVSP parameter 173
SMDE (system-managed directory entry) 414, 416, 446
connect identifier 451
input by name list 447
SMF (System Management Facilities)
data set 60
SMS (Storage Management Subsystem)
data sets (PDSE, PDS) 441
description 27
requirements 28
SMS constructs 20
SMSI parameter 573
SMSVSAM server 223
SMSW parameter 573
SO (Sequential Optimized) 171, 173
SO (shift out) 533
software end-of-file 186
space
calculated DASD (used) 400
free 159
extended format data sets 400
release 334
requirements, calculate 409
VIO data set 23
space allocation
calculation
data component 113
direct data set 536
indexed sequential data set 546, 553, 560
PDS (partitioned data set) 409, 410
PDSE (partitioned data set extended) 437
record length 38
spanned records 165
specifying 37, 45
VSAM, performance 161
space constraint relief
attributes
Dynamic Volume Count 42
Reduce Space Up To % 42
Space Constraint Relief 42
restriction, VSAM striped components 42
Space Constraint Relief attribute 42, 43
SPACE keyword 410
space management 439
Space Not Found 488

SPACE parameter 16, 23, 29, 32, 37, 132, 334, 409, 410, 437, 551
ABSTR value 548
SPANNED parameter 78, 79, 110, 159, 160
spanned records
space allocation 165
SPEED option 172, 174
SPEED parameter 117
SPEED|RECOVERY parameter 110
sphere
data set name sharing 199
spooling SYSIN and SYSOUT data sets
input streams 373
OPEN macro options 322
output streams 373
SRB (service request block) 155, 156, 191
SRB dispatching 191
STACK parameter 311
stacker selection 312, 479
control characters 290
STAE/STAI exit 544
standard label
DASD volumes 527
START command 70
status
following an I/O operation 487
STEPCAT DD statement 265
storage administrator 27
storage class
examples 330
JCL keywords 329
STORCLAS keyword 329
using 329
STORAGE macro 341, 504
STORAGECLASS parameter 111
STORCLAS parameter 30, 132
STOW ADD 455
STOW macro 16, 367, 406, 408, 411, 412, 413, 418, 423, 431, 434, 435, 444, 457, 505
update directory
PDS (partitioned data set) 418
STOW member
rename, delete 440
STOW REPLACE 455
PDSE member, extend 464
striped
alternate index 84
CA (control area) 83
data sets
RLS (not supported) 231
single, multi 387
data sets, SMS 45
extended-format sequential data sets 39
maximum extents 83
multivolume VSAM data sets 40
VSAM data sets 40
VSAM, space constraint relief 42
striped data sets
extended format data set 379
multi 398
multistriped 397
striped data sets *(continued)*
number of buffers 315
partial release request 399
sequential data 395
single-striped 397
striping
data 41
data (layering) 82
DB2 (partitioned table spaces) 114
guaranteed space attribute 42
Hiperbatch 395
improve performance 5
sequential data 395
sequential data (migrating extended format data sets) 400
space allocation 82
VSAM data 80
STRMAX field 214
STRNO parameter 175, 179
structural analysis, data sets 235
SUBALLOCATION parameter 111
subgroup, point
note list 411
subpool
resources
shared 212
subpool 252
user key storage 345
subpool, shared resources 221
subtask sharing 196
suppression, validation 337
SW (Sequential Weighted) 172, 174
switching from 24-bit addressing mode 400
SYNAD (physical error exit) 156
SYNAD exit routine 138, 155, 373, 490, 492, 494, 495
add records
ISAM data set 568
analyzing errors 255
changing address in DCB 331
deferred, write buffers 218
example 256
exception codes 496
macros used in 362, 364
programming considerations 255
register contents 497
DCB-specified 583
entry 498
SETL option 576
sharing a data set 367
synchronous
programming considerations 499
temporary close restriction 332
SYNAD exit routines 501
SYNAD parameter 485
SYNAD routine
CHECK routine 353
SYNADAF macro 324, 499, 500, 583
description 362, 364
example 591
message format 362, 364
SYNADRLS macro 500

SYNADRLS macro *(continued)*
  description 364
SYNCDEV macro 435, 468, 483
synchronizing data 483
synchronous mode 153
SYS1.DBBLIB 397
SYS1.IMAGELIB data set 480
SYSIN data set 373
  input stream 374
  logical record length 374
  routing data 375
SYSOUT data set 373
  control characters 307, 375
  routing data 373, 375, 481
system
  cross
    sharing 209
  determined block size
    tape data sets 326
  input stream 373, 375
  output stream 373, 375
system-determined block size 324
SYSTEMKEYS parameter 70
SYSZTIOT resource 503
  exit routines
    DCB OPEN exit 509

T

tape
  data set
    system-determined block size 326
  density 309
  end 358
  exceptional conditions 358
  library 3, 4, 27, 29
  mark 12, 358
  recording technique (TRTCH) 381
  to disk
    create direct data sets 538
    update direct data sets 543
  to print 378
  volume positioning 332
TAPEBLKSZLIM keyword 324
temporary
  close option 332, 338
  data set names 268
TEMPORARY attribute 50
TESTCB ACB macro 198
TESTCB macro 137, 142
  request parameter list 217
TFS files
  type of USS file system 18, 21
time sharing option 64
TIOT chain 503
TIOT option 379
track
  capacity 160
  format 8
  index
    entries 548
track *(continued)*
  index *(continued)*
    indexed sequential data set 547
    resume load 553
  number on cylinder 553
  overflow 9
TRACKS parameter 109
trailer label 514
transaction ID
  relate action requests 217
transactions, journalizing 245
TRANSID parameter 217
translation
  ASCII to/from EBCDIC 356, 359
TRC (table reference character, 3800) 290, 293, 298, 310, 375
TRKCALC macro 87, 435, 572
TRTCH (tape recording technique) 310, 381
TRTCH parameter 310
TRUNC macro 341, 344, 349, 351, 356, 392, 436
  description 351
  MULTACC 392
TSO/E (time sharing option)
  ALLOCATE command 34
  APF authorization 64
  DELETE command 63
  RENAME command 467
TTR (relative track address)
  BLDL list 445
  convert routines
    USS files 402
  directory entry list 445
  FIND macro 453
  format 10
  PDS (partitioned data set) 407
  PDSE (partitioned data set extended) 432, 433, 434
  POINT macro 455
TTR (relative track record) 11
TYPE=T parameter 332, 338

U

UBF (user buffering) 190, 200
UCS (universal character set) 480
UHL (user header label) 514, 517
UIW field (user-initiated writes) 214
unblocking records
  QISAM 545
Uncorrectable Input/Output Error condition 488
UNIQUE parameter 111
UNIQUEKEY attribute 152
UNIT parameter 273
unlabeled magnetic tape 7, 12
unlike concatenation
  data sets 385
unmovable, marked 407
Unreachable Block condition 489
UPAD exit routine 155
  cross-memory mode 259
  parameter list 259

UPAD exit routine *(continued)*
programming considerations 259
register contents at entry 257
user processing 257
UPAD wait exit 139
UPDAT mode 350
UPDAT option 324, 386, 423, 464
restriction 386
UPDAT parameter 321
update
ESDS records 149
ISAM records 561, 566
KSDS records 149
PDS (partitioned data set) 423, 425
PDSE 464
UPDATE function 452
update mode 493
UPDATEPW parameter 134
UPGRADE attribute 123
UPGRADE parameter 134
USAR (user-security-authorization record) 260
user
buffering 190, 200
catalog 265
CBUF processing considerations 206
header label (UHL) 514, 517
label
header 530
trailer 530
label exit routine 514, 517
trailer label (UTL) 514, 517
written exit routines 241, 256
user buffering (UBF) 190
user labels 543
creating, processing (data sets) 485
user-written exit routines
identifying 485
USS files
BSAM 401
buffer number default 344
NFS (Network File System) 3
path-related parameters 20
processing 18, 21
QSAM 401
RACF system security 21
sequential concatenation 403
services 87
SMF records 21
types of 401
utilities 87
VSAM 401
VSAM access, simulated 86
z/OS UNIX MVS fork service 21
USVR (user-security-verification routine) 55, 62, 121, 260
utility programs 408
IEHLIST 574
IEHPROGM 477
UTL (user trailer label) 514, 517

V

variable blocked spanned (VBS) 437
variable length
packing tracks 296
record (blocked) 393
record format-D 302, 304
record format-S 302, 304
record format-V
description 297
spanned 297
unblocked logical records 296
variable-length
record format-D 292
record format-V
description 292
segments 292, 295
spanned 295
SYSIN and SYSOUT data sets 295
relative record data set
control interval access 183
relative-record data set
record access 99
relative-record data sets
description 94
RRDS (relative-record data set)
description 93
variable-length records 439
variable-length RRDS 186
VBS (variable blocked spanned) 385, 437
VERIFY command 52, 54, 126, 154, 237
VERIFY macro 140, 207
GET macro 203
version number
absolute generation 470
vertical pointer, index entry 280
VIO (virtual I/O)
maximum size, SMS-managed data set 39
temporary data sets 22
Vio Maxsize parameter 39
virtual
resource pool 213
volumes
access exit 337
direct access 8, 309
dump 47
label 528
magnetic tape 11, 309, 527
multiple 45
multiple data sets, switching 339
positioning
CLOSE macro 332, 339, 340
EOV (end-of-volume) processing 338
releasing data set 338
releasing volume 338
security/verification exit 521, 523
separation data (index) 181
switching 357, 359, 379
system-residence 61
unlabeled tape 12
VOLUMES parameter 109, 121, 131, 134, 273
VRRDS (variable-length RRDS) 93

VSAM
RLS data, CF cache for 224
RLS data, coupling facility CF cache for 161
VSAM (virtual storage access method) 6
31-bit addresses
buffers above 16 MB 170
keywords 263
multiple LSR pools 212
addressing mode (31-bit, 24-bit) 15
allocate data sets 32
alternate index 100
backup program 51
buffer 159
catalog (generation data group base) 469
cluster (replacing) 50
control interval 110
control interval size 159
converting from ISAM to VSAM 585
Create Optimized (CO) 174
Create Recovery Optimized (CR) 174
data set
access 97
types 104
data set (logical record retrieval) 75
data sets
defining 105
types 84
Direct Weighted (DW) 174
entry-sequenced data set 97
description 85
entry-sequenced data sets
description 86
error analysis 138
EXAMINE command 235
Hiperspace 15
I/O buffers 169
ICI (improved control interval access) 192
ISAM programs for processing 580
JCL DD statement 265
key-sequenced data set 97
examining for errors 235
index processing 275
key-sequenced data sets
description 88, 91
KSDS (key-sequenced data set) 398
levels, password
control 55
master 55
read 55
update 55
linear data sets
description 91
logical record retrieval 75
look-aside processing 150
mode (asynchronous, synchronous) 153
performance improvement 159
processing data sets 15
programming considerations 241
relative- record data sets
variable length records 94
VSAM (virtual storage access method) *(continued)*
relative-record data set
accessing records 99
relative-record data sets
fixed-length records 92, 93
variable-length records 93
sample program 156, 157
Sequential Weighted (SW) 174
shared information blocks 205
sphere 199
string processing 170
striping 114
structural analysis 235
temporary data set 268
user-written exit routines
coding guidelines 242
functions 241
USS files 401
volume data sets 109
VSAM data sets
Extended Addressability 39
striped 40
VSAM RLS CF caching 161, 224
VSE (Virtual Storage Extended)
embedded checkpoint records 479, 482
chained scheduling 390
embedded checkpoint records (BSP) 481
embedded checkpoint records (CNTRL) 479
tapes 390
VSI blocks
cross-system sharing 205
data component 210
VTOC (volume table of contents) 7, 527
description 24
DSCB (data set control block) 530
ISAM data set 548
pointer 529
VTS (Virtual Tape Server) 27, 29
VVDS (VSAM volume data set) 109, 126
VVR (VSAM volume record) 127

W

WAIT macro 353, 356, 360, 385, 391, 392, 489, 535, 542, 544, 564
DECB 357
description 357
WAREA parameter 142
WORKFILES parameter 122
write integrity, cross-region sharing 203
WRITE macro 298, 354, 366, 386, 389, 394, 408, 411, 423, 424, 434, 446, 461, 482, 493, 499, 500, 537, 541, 568, 570
BISAM 357
block processing 353
description 356
K (key) 563
READ request 540
record return 355
write validity check 329
WRITECHECK parameter 110

writing
  buffer 217
WRTBFR macro 216, 221
  deferred, writing buffers 217
WTO macro 499

X

XDAP macro 21
XLRI (extended logical record interface)
  using 306

Z

z/OS Security Server 55

Readers' Comments — We'd Like to Hear from You

z/OS
DFSMS: Using Data Sets

Publication No. SC26-7410-02

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	☐	☐	☐	☐	☐

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	☐	☐	☐	☐	☐
Complete	☐	☐	☐	☐	☐
Easy to find	☐	☐	☐	☐	☐
Easy to understand	☐	☐	☐	☐	☐
Well organized	☐	☐	☐	☐	☐
Applicable to your tasks	☐	☐	☐	☐	☐

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? ☐ Yes ☐ No

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

Name _____ Address _____

Company or Organization _____

Phone No. _____

Readers' Comments — We'd Like to Hear from You
SC26-7410-02

Cut or Fold
Along Line

Fold and Tape | **Please do not staple** | Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
RCF Processing Department
M86/050
5600 Cottle Road
San Jose, CA U.S.A. 95193-0001

Fold and Tape | **Please do not staple** | Fold and Tape

SC26-7410-02

Cut or Fold
Along Line

Program Number: 5694-A01

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC26-7410-02