IBM z/VSE
VSE Central Functions

VSE/VSAM User's Guide and Application Programming

Version 7 Release 1

SC33-8246-00

IBM z/VSE
VSE Central Functions

VSE/VSAM User's Guide and Application Programming

Version 7 Release 1

SC33-8246-00

Note !

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

First Edition (March 2005)

This edition, which is an update of *VSE/VSAM User's Guide and Application Programming, VSE Central Functions 6.4*, SC33-6732-00, applies to Version 7 Release 1 of IBM VSE/VSAM, which is part of VSE Central Functions, Program Number 5686-CF7, and to all subsequent releases and modifications until otherwise indicated in new editions.

IBM VSE/VSAM for VM (Program Number 5686-081) is no longer available as an optional product with z/VSE 3.1. All references to it have been removed from the VSE/VSAM technical documentation. Customers having a need to run VSE/VSAM for VM should keep a copy of *VSE/VSAM User's Guide and Application Programming, VSE Central Functions 6.4*, SC33-6732-00, and *VSE/VSAM Commands, VSE Central Functions 6.4*, SC33-6731-00, as well as VSE/ESA documentation through release 2.7.

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

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

```
IBM Deutschland Entwicklung GmbH
Department 3248
Schoenaicher Strasse 220
D-71032 Boeblingen
Federal Republic of Germany
```

You may also send your comments by FAX or via the Internet:

```
Internet: s390id@de.ibm.com
FAX (Germany): 07031-16-3456
FAX (other countries): (+49)+7031-16-3456
```

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

© Copyright International Business Machines Corporation 1979, 2005. All rights reserved.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Figures ix

Tables xi

Notices xiii
Programming Interface Information xiii
Trademarks and Service Marks xiv

About This Book xv
Who Should Use This Book xv
How to Use This Book xv
Where to Find More Information xv

Abbreviations xvii

Summary of Changes xix
New Information xix
Device Dependencies xix
Buffer Hashing xix
IDCAMS SNAP Command xix
Changed Functionality xix
Other Changes xix
Deleted Material. xx

Chapter 1. Introduction to IBM VSE/VSAM 1
Overview 1
Advantages. 1
Functions of IBM VSE/VSAM 2
Concepts of Data Organization 4
File Types 4
Elements of Organization 5
Catalogs with VSE/VSAM. 6
Indexes with VSE/VSAM 7
How to Communicate with VSE/VSAM 8
IDCAMS Commands 8
VSE/VSAM Macros 10
Job Control Parameters to Access VSE/VSAM Files 11
z/VSE Interactive Interface 11

Chapter 2. Planning Information 13
Compatibility with IBM VSE/VSAM Version 2 . . . 13
Overview of Environment and Requirements . . . 13
What to Consider 13
Partition Space for Non-SVA-Eligible Routines. . 13
Device Dependencies 14
Storage for VSE/VSAM 14
Space for Running in Virtual Mode 14
Space for Running in Real Mode 14
Partition Requirement for Buffers and Control Blocks 14
Storage for the VSE/VSAM Space Management for SAM Function 17
SAM Access Routines 17
Space for Running in Real Mode 17
Partition Requirement for Control Blocks and Buffers 17
Partition Virtual Storage 18
GETVIS Requirements for Managed-SAM Access to SAM ESDS Files 18
Storage for the ISAM Interface Program (IIP) . . . 18
Storage for IDCAMS Including the VSE/VSAM Backup/Restore Function. 18
VSE/VSAM Backup/Restore Function 19

Chapter 3. Operation and Job Control 21
IPL Command Specifications for VSE/VSAM . . . 21
Assigning a Device to the Master Catalog . . . 21
Defining the Lock File. 21
Specifying the Number of Supervisor Buffers for Channel Programs 22
Volume Mounting 22
Mounting a Volume Through Job Control Specifications. 22
Mounting a Volume Through Automatic Assignment 22
Use of z/VSE Job Control Statements for VSE/VSAM 23
Job Control Statements for Catalogs 23
Job Control Statements for Files 25
// DLBL Statement. 26
Format of the DLBL Statement 27
File Disposition 29
// EXEC Statement. 36
Note to Users of the VSE/VSAM Space Management for SAM Function 37
Format of the EXEC Statement 37
// EXTENT Statement. 39
Format of the EXTENT Statement 39
Using Job Control for Catalog Definition 40
Overview of Catalogs 40
Specifying the Master Catalog 41
Specifying a User Catalog 42
Specifying a Job Catalog 42
Search Sequence of Catalogs. 44

Chapter 4. Tasks under VSE/VSAM. . . 47
Data and Space Management 47
About the VSE/VSAM Catalog. 47
Defining VSE/VSAM Data Spaces on a Volume 48
Defining VSE/VSAM Files 48
About Volumes and VTOCs 48
Work Files on Virtual Disk 51
Preparations for Use 51
Restrictions 52
Transporting Files between Systems 52

© Copyright IBM Corp. 1979, 2005

Transporting Catalog Information 52
Transporting Files between VSE/VSAM and DFSMSdfp VSAM or DFSMS/MVS 52
Transporting Files between VSE/VSAM and MVS/VSAM (not DFP) 52
Transporting Files between z/VSE Systems . . . 53
Catalog and File Migration 53
Definitions for Catalog Migration 53
Migrating Catalogs 55
Migrating VSE/VSAM Files to Another Device 55
NonVSAM Migration 57
Space Allocation through Modeling 57
Using an Object as a Model 57
About the MODEL Subparameter 58
Explicit Allocation Models 58
Explicit Noallocation Models 60
Implicit NOALLOCATION Models (Default Models) 61
How VSE/VSAM Determines Which Parameters to Use 62
Restrictions 63
Default Volumes 65

Chapter 5. Working With Compressed Files 67
Introduction to VSE/VSAM Compression 67
Advantages 67
Activating VSE/VSAM Data Compression . . . 67
How VSE/VSAM Data Compression Works Internally 68
Dictionary Creation 68
Compression States 69
Data Format of Records 69
How to Define the Compression Control Data Set 70
Which Data Set Types Are Eligible 71
Restrictions 71
The VSE/VSAM Compression Prediction Tool (IKQCPRED) 72
Using IKQPRED 72
Method of Operation 73
Interpreting IKQCPRED Results 73

Chapter 6. Device Dependencies . . . 77
VSE/VSAM Support of Large DASD 77
Making Use of the Support 77
Migrating to Large DASD Using IDCAMS Backup/ Restore 78
Performance Considerations (KSDS Only) . . . 78
Increased Size of the Catalog Index 79
Restrictions for VSE/VSAM Support of Large DASD 79
New or Changed Fields in LISTCAT Output . . 79
Support for FBA Disk Devices (FBA and SCSI) . . 80
Technical Considerations 80
Restrictions 81
Virtual Tapes 82

Chapter 7. Optimizing the Performance of VSE/VSAM 83
Number of Files Defined in a Catalog 83
Data Space Classification 83
Control Area (CA) Size 85
Minimum and Maximum CA Sizes 85
Performance Implications 86
Disk Storage Sizes 86
Control Interval (CI) Size 88
How to Specify 88
Data CI and Block Sizes 88
CI Size in a Data Component 89
CI Size in an Index Component 92
Key Compression 93
I/O Buffer Space (Using Non-Shared Resources) . . 94
Considerations 95
Buffer Specification 95
Buffer Allocation 96
I/O Buffer Space (Using Local Shared Resources) . . 99
Miscellaneous Notes on Buffer Allocation (LSR) 99
LSR Buffer Hashing 100
Preventing Deadlock in Buffer Contention 101
Multiple Volume Support 102
Key Ranges 102
Space Allocation 102
Examples: Allocation of Space on Multiple Volumes 103
Space Allocation 108
Possible Options 108
NOALLOCATION 109
Data Protection and Integrity Options 110
Distributed Free Space 111
Loading a File 111
CI/CA Splits 114
Examples: CI/CA Splits 114
Index Options 120
Number of Index Records in Virtual Storage . . 120
Index and Data on Separate Volumes 121
Consideration for ECKD Devices 121
Key Ranges 121
Performance Measurement 121
Displaying Statistics About Buffer Pools . . . 122

Chapter 8. Data Protection and Data Recovery 125
Data Protection 125
Passwords to Authorize Access 125
User Security-Verification Routine 128
Protecting Shared Data 129
Data Integrity 132
IDCAMS Commands and Command Options for Data Integrity 132
Using the DEFINE SPACE Command 132
Using the DEFINE CLUSTER Allocation Subparameter 133
Using the DEFINE USERCATALOG Command 133
Protecting VSE/VSAM Files and Volumes 134
Backup Considerations 134
Relationship of Catalog Entries to VSE/VSAM Files and Volumes 135
Creating Backup Copies of VSE/VSAM Files 135
Creating Backup Copies of Volumes 136
Protecting VSE/VSAM Catalogs 137
Creating Backup Copies of Catalogs 137

Rebuilding a Catalog 138
Guide to VSE/VSAM Recovery 139
About Data Organization and Recovery . . . 139
About the Recovery Process 139
Levels of Recovery 139
Tools for Data Integrity, Backup, and Recovery . . 139
Procedures for VSE/VSAM Recovery 143
File is Not Properly Closed 143
File is Inaccessible 145
Catalog is Unusable 146
Volume is Inaccessible 148
Quick Recovery 150
Procedure for Quick Recovery 150

Chapter 9. VSE/VSAM Support for SAM Files 151
Overview 151
About SAM ESDS Files 151
About the VSE/VSAM Space Management for SAM Function 152
Advantages in Using SAM ESDS Files 152
Dynamic Allocation 152
Simplified Job Control 152
Default Modeling 153
Implicit File Definition 153
Device Independence 153
IDCAMS Commands 153
Security and Integrity of Data 153
Data Recovery 153
Additional Functions Available for Managed-SAM Access 154
Planning for Files 154
Work Files 154
Disposition 155
Extending Existing SAM ESDS Files 155
Levels of Migrating Data and Programs from SAM to VSE/VSAM Control 155
Functions Available at the Various Migration Levels 156
Creating a SAM ESDS File 157
Setting Up a Quantity of Space 157
Defining a SAM ESDS File 158
Explicit Define Cluster (Using the DEFINE CLUSTER Command) 158
Implicit Define Cluster 161
Resetting and Reusing a Previously-Defined File 164
Using a SAM ESDS File 165
Access to a SAM ESDS File 165
Managed-SAM Access: Differences to (Unmanaged) SAM Access 165
Using SAM ESDS Files: Restrictions 167
VSE/VSAM Access of SAM ESDS Files: Considerations 169
The IDCAMS Commands for a SAM ESDS File . . 170
Implicit Deletion of a SAM ESDS File 173
Sample Programs and Job Streams 174
Example 1: Load a SAM ESDS File by Way of Managed-SAM Access 174
Example 2: Implicit Define of a SAM ESDS File 176
Example 3: Define a Default Model SAM ESDS File 177
Example 4: Define a Dynamic SAM ESDS File and Access 177
Differences Between VSE/VSAM ESDS and SAM ESDS File Format 179
How CIs are Formatted into CAs 179
Relationship of Physical and Logical Layout . . 181

Chapter 10. Using VSE/VSAM Macros 183
Groups of Macros 183
Declarative VSE/VSAM macros: 183
Macros to Share Resources Between Several Files 183
Request Macros 183
Control Block Manipulation Macros 184
OPEN/CLOSE Macros 184
Relating a Program and the Data 184
ACB: Specifying the Access Method Control Block 184
EXLST: Specifying the Exit List 185
RPL: Specifying the Request Parameter List . . 186
GENCB: Generating Control Blocks and Lists 187
Connecting and Disconnecting a Processing Program and a File 187
OPEN: Connecting a Processing Program to a File 187
CLOSE: Disconnecting a Processing Program from a File 187
TCLOSE: Securing Records Added to a File . . 188
Manipulating and Displaying the Information Relating Program and Data 188
MODCB: Modifying the Contents of Control Blocks and Lists 188
SHOWCB: Displaying Fields of Control Blocks and Lists 188
TESTCB: Testing the Contents of Control Blocks and Lists 188
Requesting Data Transfer, Positioning, and Deletion of Records 189
Displaying Catalog Information 189
SHOWCAT: Retrieving Information from a Catalog 189
Sharing Resources Among Files and Displaying Catalog Information 191
Data Set Name Sharing 192
Considerations 192
Processing 193
Specifying Manipulation Macros 193
Buffers and LSR Pools above 16MB Line of Storage 193

Chapter 11. Descriptions of VSE/VSAM Macros 195
Syntax of VSE/VSAM Macros 195
VSAM Executable Macros and Their Mode Dependencies 196
The ACB Macro 196
Format of the ACB Macro 197
OPEN/CLOSE/TCLOSE Message Area 204
The BLDVRP Macro 207
Deciding How Big a Pool to Provide 207
Displaying Information about an Unopened File 207

Displaying Statistics about a Buffer Pool . . . 207
Format of the BLDVRP Macro 208
Return Codes from BLDVRP 210
Connecting a File to a Resource Pool 210
Restrictions 210
The CLOSE Macro. 211
Format of the CLOSE Macro 211
The DLVRP Macro 212
Format of the DLVRP Macro 212
Return Codes from DLVRP 213
The ENDREQ Macro 213
Format of the ENDREQ Macro 214
The ERASE Macro. 214
Format of the ERASE Macro 214
The EXLST Macro 214
Format of the EXLST Macro 215
EODAD Exit Routine to Process End-of-File . . 216
EXCPAD Exit Routine 217
JRNAD Exit Routine to Journalize Transactions 218
LERAD Exit Routine to Analyze Logic Errors 220
SYNAD Exit Routine to Analyze Physical Errors 220
The GENCB Macro 221
Format of the GENCB Macro 222
Examples of the GENCB Macro 223
The GET Macro 224
Format of the GET Macro 224
The MODCB Macro 226
Format of the MODCB Macro 226
Examples of the MODCB Macro 227
The OPEN Macro 228
Format of the OPEN Macro 228
Return Codes from OPEN 229
The POINT Macro. 229
Format of the POINT Macro 230
The PUT Macro 230
Format of the PUT Macro 231
The RPL Macro. 231
Format of the RPL Macro 232
Specifying Processing Options for a Request . . 238
The SHOWCAT Macro 246
Format of the SHOWCAT Macro 246
Return Codes from SHOWCAT 249
The SHOWCB Macro. 251
Format of the SHOWCB Macro 252
Keywords of the ACB, EXLST, and RPL Macros 253
Length of a Control Block or List. 254
Attributes of an Open File 254
Example: The SHOWCB Macro 255
Example: Statistics on Use of LSR Buffer Pools 256
The TCLOSE Macro 257
Format of the TCLOSE Macro 257
The TESTCB Macro 257
Format of the TESTCB Macro 258
Operands of the ACB, EXLST, and RPL Macros 259
Length of a Control Block or List. 259
Attributes of an Open File or Index 260
The WRTBFR Macro 262
Managing I/O Buffers 262
Deferring Write Requests 262
Relating Deferred Requests by Transaction ID 263
Writing Buffers Whose Writing Has Been Deferred 263
Format of the WRTBFR Macro. 264
Examples: ACB, EXLST, and RPL Macros 265
Specifying VSE/VSAM Control Blocks 265
JCL to Open and Process a File 266
Examples of Request Macros 267
How to Retrieve a Record: GET Macro 268
How to Position for Subsequent Sequential Access: GET and POINT Macros 273
How to Chain Request Parameter Lists and Terminate a Request: ENDREQ Macro 276
How to Store a Record: PUT Macro 278
How to Update a Record: GET and PUT Macros 282
How to Delete a Record: GET and ERASE Macros 285
How to Use Extended User Buffering: GET and PUT Macros 287
Current User Buffering Support 287
Extended User Buffering Support. 288
Using Extended User Buffering 288
Errors 289
Return Codes of Request Macros 289
Return Codes from the Control Block Manipulation Macros 291
List, Execute, and Generate Forms of the Control Block Manipulation Macros. 291
List and Execute Forms 291
Generate Form 292
Examples of the List, Execute, and Generate Forms 292

Appendix A. Operand Notation and Parameter Lists for VSE/VSAM Macros 295

Operand Notation for VSE/VSAM Macros . . . 295
GENCB Macro Operands 297
MODCB Macro Operands 298
SHOWCB Macro Operands. 299
TESTCB Macro Operands 300
BLDVRP Macro Operands 302
DLVRP Macro Operands 302
SHOWCAT Macro Operands 303
WRTBFR Macro Operands 303
Parameter Lists for VSE/VSAM Macros 303
The GENCB Parameter List. 304
The MODCB Parameter List 306
The SHOWCB Parameter List 308
The TESTCB Parameter List 309
The BLDVRP Parameter List 312
The SHOWCAT Parameter List 313

Appendix B. Invoking IDCAMS from a Program 315

Invoking Macro Instructions 315
Explanation 316
User I/O Routines. 317

Appendix C. Advantages of the ISAM Interface Program (IIP) 321

Comparison of VSE/VSAM and ISAM 321
Differences Between ISAM and VSE/VSAM . . 321
VSE/VSAM Functions That Go Beyond ISAM 322
Preparations and Using the ISAM Interface Program 323
Step 1: Consider Restrictions in the Use of IIP and VSE/VSAM 324
Step 2: Define a VSE/VSAM File 324
Step 3: Load the VSE/VSAM File. 325
Step 4: Changing ISAM Job Control Statements 326
What the ISAM Interface Program Does 326

Appendix D. Compatibility With Other Products 329
Portability of VSE/VSAM Files to DFSMSdfp VSAM. 329
FBA Support 329
Dedicated VSE/VSAM Volume 329
Data Space Classification 329
Default Models. 330
Default Volumes 330
Multiple Volume Ownership 330
Catalog Check Services 330
Backup/Restore 330
Device Dependency 331
VSE/VSAM Data Compression 331
Compatibility of VSE/VSAM with DFSMSdfp VSAM. 331
Similarities between VSE/VSAM and ACF/VTAM 331

Appendix E. VSE/VSAM Labels 333
Types of VSE/VSAM Labels 333
Volume Label 333
Data Space Label 333
Space Continuation Label 334
VTOC Label. 334
Location of Labels. 334
Volume Layouts 334
Label Information Area 335
VTOC Label Processing 335
VSE/VSAM Data Spaces 335
VSE/VSAM Files 336
VTOC Labels for FBA Devices. 336
VSE/VSAM Data Space 337
VSE/VSAM Files 339
Job Stream Examples 340
Example - Define Data Spaces 341
Example - Define a File in a Catalog. 343
Example - Define a Unique File 343
Example - Process a File. 343

Appendix F. Diagnosis Tools 345
Catalog Check Service Aid (IKQVCHK) 345
In Case of Errors 346
How to Run a Check 346
Examples of Error Messages 346
Output of a Check. 348
SNAP Dump (IKQVEDA) 350
How to Run a SNAP Dump 351
Example: SNAP Dump 0001 353
Maintaining VTOC and VOL1 Labels on Disk (IKQVDU) 355
How to Run the IKQVDU 355
Error Message and Codes (from IKQVDU) . . 359

Glossary 363

Related IBM Manuals 371
IBM z/VSE 371
Various 371

Index 373

Figures

1. ESDS File Format: Records Stored as Received 4
2. KSDS File Format: Records Stored in Key Field Sequence 4
3. RRDS File Format: Fixed-Length Records Stored by Record Number 5
4. VRDS File Format: Variable-Length Records Stored by Record Number 5
5. VSE/VSAM File Type Structures 6
6. Example: Two Alternate Indexes for a Key-Sequenced File 8
7. Partition Requirements for Control Blocks and Buffers (with NSR) 15
8. Partition Requirements for Control Blocks and Buffers (with LSR) 16
9. How VSE/VSAM Allocates Buffers 17
10. Disposition: Values and Applicability 28
11. Disposition: Meaning of Values 29
12. VSE/VSAM Access: OPEN Disposition 31
13. Managed-SAM Access: OPEN Disposition -- OUTPUT/INPUT 32
14. Managed-SAM Access: OPEN Disposition -- WORKxxxx 33
15. VSE/VSAM Access: CLOSE Disposition 35
16. Managed-SAM Access: CLOSE Disposition 36
17. Relationship of Catalogs and Files 40
18. // DLBL Specifications and Search Sequence of Catalogs. 45
19. Explicit Allocation Model 58
20. Specifying the MODEL Parameter at the CLUSTER Level Only 60
21. Explicit NOALLOCATION Model 61
22. Implicit NOALLOCATION Models. 62
23. The Four Compression States of a Compressed Cluster 69
24. Sample IKQCPRED Output 73
25. Classification of Data Space 84
26. How VSE/VSAM Computes Physical Block Size 89
27. Relationship of CI Size to Physical Block Size for Data Component 91
28. Migration from SAM Control to VSE/VSAM Control 156
29. Valid Combinations of Access Methods and File Types. 156
30. Example of CA Format Using a VSE/VSAM Entry-Sequenced File 180
31. Example of Non-CA Format Using a SAM ESDS File 181
32. Comparison of a VSE/VSAM Block to a SAM Logical Block 182
33. Relationship of Catalog Entries 190
34. GENCB Macro Examples 224
35. MODCB Macro Examples 228
36. Example of an RPL Chain Built by Specifying the NXTRPL Operand. 235
37. Summary of Processing Options for Keyed and Addressed Access 239
38. Example of Backward Sequential Retrieval through a Path with Non-Unique Alternate Keys 241
39. Format of the SHOWCAT Work Area 250
40. SHOWCB Macro Example 256
41. SHOWCB Macro Example 256
42. TESTCB Macro Examples 262
43. Example of Specifying Control Blocks for a File 265
44. Example of JCL Needed to Open and Process a File 267
45. Request Macro Example 1: Keyed-Sequential Retrieval 268
46. Request Macro Example 2: Skip-Sequential Retrieval 269
47. Request Macro Example 3: Addressed-Sequential Retrieval 270
48. Request Macro Example 4: Keyed-Direct Retrieval 272
49. Request Macro Example 5: Addressed-Direct Retrieval 273
50. Request Macro Example 6: Keyed Positioning with POINT 274
51. Request Macro Example 7: Switching from Direct to Keyed-Sequential 275
52. Request Macro Example 8: Chaining Request Parameter Lists 276
53. Request Macro Example 9: Giving up Positioning for Other Request 277
54. Request Macro Example 10: Keyed-Sequential Insertion 278
55. Request Macro Example 11: Skip-Sequential Insertion 280
56. Request Macro Example 12: Keyed-Direct Insertion 281
57. Request Macro Example 13: Addressed-Sequential Addition 282
58. Request Macro Example 14: Keyed-Sequential Update. 283
59. Request Macro Example 15: Keyed-Direct Update. 284
60. Request Macro Example 16: Addressed-Sequential Update 285
61. Request Macro Example 17: Keyed-Direct Deletion 286
62. Request Macro Example 18: Addressed-Sequential Deletion. 287
63. Examples of the List and Execute Form 292
64. Example of the Generate Form. 293
65. Processor Invocation Argument List from a Program 316
66. Arguments Passed to and from a User I/O Routine 318
67. Using the ISAM Interface Program 327

© Copyright IBM Corp. 1979, 2005

68. Volume Layouts of VSE/VSAM Files 335
69. Examples: Defining VSE/VSAM Data Spaces 342
70. Example: Defining a VSE/VSAM File Suballocated from a Data Space 343
71. Example: Defining a Unique VSE/VSAM File (File-ID MSTRFILE) 343
72. Example: Processing a VSE/VSAM File with an Assembler Program 344
73. VSE/VSAM Diagnosis Tools 345
74. Example: Key-Range Names not Matching 347
75. Example: Incorrect Association Group Occurrence 348
76. Example: Output from the Catalog Check Service Aid (IKQVCHK) 348
77. Low-Key-Range Catalog Records and Codes 350
78. Example: SNAP Dump Output. 355
79. Display of IKQVDU Functions 357
80. Explanation to IKQVDU Functions 358

Tables

1. // DLBL Statement Required for Job Catalogs and User Catalogs 25
2. Job Control Statements Required for Files 25
3. Modeling of DEFINE Parameters 63
4. Minimum CI Sizes Depending on Key Length 79
5. Minimum and Maximum CA for Generic FBA Devices 81
6. Disk Storage Sizes for IBM CKD Devices 87
7. Disk Storage Sizes for IBM FBA (and SCSI) Devices 87
8. Register Settings on Passing Control to Authorization Routine 129
9. Tools for Integrity, Backup, and Recovery 140
10. VSAM Macros and Their Mode Dependencies 196
11. ERREXT Parameter List for ISAM Programs with IIP 327
12. FilenameC with IIP when IOROUT=ADD, RETRVE, or ADDRTR. 328
13. FilenameC with IIP when IOROUT=LOAD 328

© Copyright IBM Corp. 1979, 2005

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of the intellectual property rights of IBM may be used instead of the IBM product, program, or service. The evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, are the responsibility of the user.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785, U.S.A.

Any pointers in this publication to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement. IBM accepts no responsibility for the content or use of non-IBM Web sites specifically mentioned in this publication or accessed through an IBM Web site that is mentioned in this publication.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

```
IBM Deutschland GmbH
Department 0790
Pascalstr. 100
70569 Stuttgart
Germany
```

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

Programming Interface Information

This manual is intended to help the customer in efficiently using:

- The major facilities of the *IBM VSE/Virtual Storage Access Method* (VSE/VSAM).
- VSE/VSAM macro instructions to process data. To use the VSE/VSAM macros, you must be familiar with the High Level Assembler for MVS & VM & VSE (see *High Level Assembler for MVS & VM & VSE Language Reference* and *High Level Assembler for MVS & VM & VSE Programmer's Guide*).

It also documents intended Programming Interfaces that allow the customer to write programs to obtain the services of VSE/Virtual Storage Access Method.

Such information is enclosed in brackets as follows:

© Copyright IBM Corp. 1979, 2005

Programming Interface Information

End of Programming Interface Information

Trademarks and Service Marks

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

ACF/VTAM
DFSMSdfp
DFSMS/MVS
ECKD
Enterprise Storage Server
Enterprise System Architecture/390
eServer
ESCON
IBM
SQL/DS
S/390
VSE/ESA
VTAM
z/Architecture
z/OS
zSeries
z/VM

About This Book

This book contains guidance information for using the functions that are available with the *IBM VSE/Virtual Storage Access Method (VSE/VSAM)*. It describes the major facilities of the program and how to use them efficiently.

This book explains *concepts* of VSE/VSAM. Furthermore, it includes information about:
- Planning for VSE/VSAM.
- Using various diagnosis tools.
- Using VSE/VSAM macros.

z/VSE is the successor to IBM's VSE/ESA product. Many products and functions supported on z/VSE may continue to use VSE/ESA in their names.

Please be aware that the z/VSE operating system can execute in 31-bit mode only. It does not implement z/Architecture, and specifically does not implement 64-bit mode capabilities. The z/VSE operating system is designed to exploit select features of IBM eServer zSeries hardware.

Who Should Use This Book

This book is intended for the VSE/VSAM application programmer and for the end user.

How to Use This Book

- *Conceptual Information*:

  If you want *basic information* about VSE/VSAM, refer to Chapter 1, "Introduction to IBM VSE/VSAM," on page 1.
- *Requirements and Planning*:

  If you need to know about *VSE/VSAM requirements*, or want to *plan* and calculate storage space, refer to Chapter 2, "Planning Information," on page 13.

  If you are responsible for planning the *protection of data* at your installation, you should acquaint yourself with Chapter 8, "Data Protection and Data Recovery," on page 125.
- *For information on VSE/VSAM macros:*

  Chapter 10, "Using VSE/VSAM Macros" describes how to use the VSE/VSAM macros.

  Chapter 11, "Descriptions of VSE/VSAM Macros" shows the format for each VSE/VSAM macro and the meaning of each parameter.

  Appendix A, "Operand Notation and Parameter Lists for VSE/VSAM Macros" shows how to specify operands and how to use parameter lists of the various VSE/VSAM macros.

Where to Find More Information

- *VSE/VSAM Commands*, SC33-8245

© Copyright IBM Corp. 1979, 2005

gives an overview of and detailed information on the *IDCAMS utility program* (how to create and maintain files).

- *z/VSE Messages and Codes*, SC33-8226, SC33-8227, SC33-8228

  lists VSE/VSAM messages and their explanations.

For a list of all the manuals mentioned in this book, refer to "Related IBM Manuals" on page 371.

z/VSE Home Page

z/VSE has a home page on the World Wide Web, which offers up-to-date information about VSE-related products and services, new z/VSE functions, and other items of interest to VSE users.

You can find the z/VSE home page at:

`http://www.ibm.com/servers/eserver/zseries/zvse/`

Abbreviations

The following abbreviations are used in this book:

ACB	Access control block
ACF	Advanced Communications Function
AIX	Alternate index
ASA	American Standards Association
ASCII	American National Standard Code for Information Interchange
ASI	Automated system initialization
BG	Background
CA	Control area
CCDS	Compression control data set
CCW	Channel command word
CI	Control interval
CIDF	Control information definition field
CKD	count-key-data (device)
CP	Control program
CPU	Central processing unit
CRA	Catalog recovery area
DFSMSdfp	Data Facility Storage Management Subsystem Data Facility Product (MVS)
DLBL	Disk label
DLF	Define lock file
DOS	Disk Operating System
DSF	Device Support Facilities
DTF	Define the file
EBCDIC	Extended binary coded decimal interchange code
ECKD	Extended count key data
EOF	End of file
EOV	End of volume
ESA	Enterprise Systems Architecture
ESCON	Enterprise systems connection
ESS	Enterprise Storage Server
ESDS	Entry-sequenced data set
EXCP	Execute channel program
FBA	Fixed-block-architecture (device)
HALCRBA	High allocated relative byte address
ICCF	Interactive Computing and Control Facility
ICF	Integrated Catalog Facility
ID	Identifier
IIP	ISAM Interface Program
IPL	Initial program load
ISAM	Indexed-sequential access method
ISO	International Standards Organization
JCL	Job control language
JIB	Job information block
KB	Kilobyte (1024 bytes)
KSDS	Key-sequenced data set
LSR	Local shared resources
LU	Logical unit
MB	Megabyte (1,048,576 bytes)
MSHP	Maintain system history program
MVS	Multiple Virtual Storage

© Copyright IBM Corp. 1979, 2005

NSR	Non-shared resources
RBA	Relative byte address
RDF	Record definition field
RL	Record length
RRDS	Relative record data set
SAM	Sequential Access Method
SCSI	Small Computer Systems Interface
SDL	System directory list
SEOF	Software end-of-file
SIO	Start I/O
SP	System Package
SVA	Shared virtual area
UPSI	Use program switch indicator
VM	Virtual machine
VRDS	Variable-length relative-record data set
VSAM	Virtual Storage Access Method
VSE	Virtual Storage Extended
VTAM	Virtual Telecommunications Access Method
VTOC	Volume Table of Contents
XA	Extended Architecture
z/VSE	zSeries VSE

Summary of Changes

For a complete overview of the functions that are new with z/VSE, refer to *z/VSE Planning*, SC33-8221.

New Information

This section summarizes information that has been added since the previous edition of the manual. Some of these functions were introduced in previous VSE/ESA releases and have been documented in the respective *Release Guide*.

Device Dependencies

Information on support for Large DASD and SCSI disk is included in a new chapter entitled Chapter 6, "Device Dependencies," on page 77.

Buffer Hashing

For more information, refer to "LSR Buffer Hashing" on page 100.

IDCAMS SNAP Command

The IDCAMS SNAP command allows fast copying of a volume residing on an IBM Enterprise Storage Server (ESS), and containing a VSE/VSAM master or user catalog, by exploiting the ESS FlashCopy functionality. If a removable-media backup (such as to tape) is required, this can be performed from the "snapped" copy of the catalog, significantly reducing the time data sets are not available to online processing. It also ensures a completely synchronized backup, since none of the files in the catalog will be changed during the backup process. For more information, refer to *VSE/VSAM Commands*.

Changed Functionality

This section summarizes information on modified functionality.

- On any device, the definition of KSDS or VRDS files with the REPLICATE parameter, which causes each index record to be written on a track as many times as it will fit, is no longer supported. This was originally employed to reduce rotational delay and provide a possible performance improvement. With the current hardware architecture and widespread use of cache, this consideration is no longer valid. If this parameter is specified in an IDCAMS DEFINE job, the parameter will be rejected. Files that were backed up with IDCAMS BACKUP and defined with the REPLICATE parameter will be restored with a replicated index. Existing files defined with the REPLICATE parameter will continue to be supported.

Other Changes

This section summarizes miscellaneous changes:

- Update to reflect software rebranding (such as z/VSE, z/OS).
- Clarification of the DISP parameter of the // DLBL job control statement.

© Copyright IBM Corp. 1979, 2005

Deleted Material

Information on the following has been deleted from this manual:

- VSE/VSAM for VM (Program Number 5686-0819), which is no longer available as of z/VSE 3.1.
- The catalog recovery function, which is no longer supported:
  - Catalog recovery area (CRA)
  - IDCAMS commands
    - IMPORTRA
    - EXPORTRA
    - RESETCAT
    - LISTCRA
  - RECOVERABLE parameter
- The following additional parameters, which are no longer supported:
  - IMBED
  - REPLICATE
- Year 2000 considerations

References continue to be made to the catalog recovery function and to the indicated parameters with respect to existing structures originally defined using these parameters.

Chapter 1. Introduction to IBM VSE/VSAM

This Chapter ...

- Summarizes the **advantages** of using the *IBM VSE/Virtual Storage Access Method (VSE/VSAM)*.
- Highlights the use of programs and *functions* of VSE/VSAM.
- Provides you with **conceptual information** on physical file organization, and the elements that are used in managing and processing the files.
- Gives you an outline on the various means for **communicating** with VSE/VSAM.

For information on requirements and compatibility of VSE/VSAM, refer to Chapter 2, "Planning Information," on page 13.

Overview

IBM VSE/VSAM is an access method for the indexed or sequential processing of records on direct access devices. Records can be of fixed-length or variable-length.

VSE/VSAM is the preferred file management system for VSE environments.

Note: If you presently have SAM or ISAM files, you can convert such files to the VSE/VSAM format.

VSE/VSAM can handle:
- Batch and online processing
- Direct and sequential access
- Access by key, record number, or address
- Intermixed types of processing in a common data base

VSE/VSAM provides the following means of communication:
- The *IDCAMS utility program*. You use IDCAMS commands to create and maintain VSE/VSAM files on disk, independently of a specific program.
- The *VSE/VSAM macros*. You use the macros for processing such created files from a program.

Advantages

Central Control

You can centrally control all the VSE/VSAM files at your installation, because all information about files and their storage space is collected in VSE/VSAM catalogs. In the catalogs, you can define and delete files, and you can change information about the files.

Data Protection and Integrity

You can control the access to data by assigning passwords to various objects (for example, to files and catalogs).

© Copyright IBM Corp. 1979, 2005

Also, you can protect your data from accidental loss or destruction by using the *VSE/VSAM Backup/Restore Function*. This function allows you to easily store data on tape or disk and call it back.

Device Independence

You simply set aside an area of space to be used exclusively by VSE/VSAM. From this space, VSE/VSAM selects whatever space is needed for a file when it is defined. Space allocation is dynamic; if a file or catalog must be extended, VSE/VSAM allocates more space to it.

Portability of Data Between Systems

VSE/VSAM uses a record format that is common to the IBM operating systems z/VSE, z/VM, and z/OS [1]. Therefore, but with some exceptions, VSE/VSAM files are portable to MVS/VSAM. You can get full portability for files and volumes by using only those commands, file types, devices, and programming interfaces that are supported by all environments (z/VSE, z/VM, and z/OS). For information on portability requirements, refer to Appendix D, "Compatibility With Other Products," on page 329.

To move files between different operating systems, you can use the EXPORT/IMPORT commands of the *IDCAMS utility program*. Communication with VSE/VSAM is essentially the same for the z/VSE, z/VM, and z/OS operating systems, except for job control.

Ease of Conversion from SAM or ISAM to VSE/VSAM

To take advantage of VSE/VSAM processing capabilities, you can convert SAM (sequential access method) files and ISAM (indexed sequential access method) files to VSE/VSAM format:

- If you have SAM files, you can use the *VSE/VSAM Space Management for SAM Function* to convert the files. Then, you can use commands of the *IDCAMS utility program* to manipulate the converted files. For an overview, see below; for more details, refer to Chapter 9, "VSE/VSAM Support for SAM Files," on page 151.
- Your existing ISAM programs can use the *ISAM Interface Program (IIP)* to process the files. For further considerations, refer to Appendix C, "Advantages of the ISAM Interface Program (IIP)," on page 321.

Functions of IBM VSE/VSAM

With VSE/VSAM, the following functions are included:

- *VSE/VSAM Space Management for SAM Function*
- *VSE/VSAM Backup/Restore Function*

VSE/VSAM Space Management for SAM Function

You can use this function to manage your SAM files, including most system work files.

Use the function to convert a SAM file into a SAM ESDS file by placing the SAM ESDS file into VSE/VSAM space. Then, the SAM ESDS files can be accessed by SAM macros as well as by VSE/VSAM macros.

The *VSE/VSAM Space Management for SAM Function* allows you to:

- Define and delete a SAM ESDS file in VSE/VSAM space. Use IDCAMS commands, or define/delete implicitly at OPEN/CLOSE time.

1. The principal component of z/OS is MVS; references to MVS in this book should be understood as meaning the MVS element of the z/OS operating system.

- Access a SAM ESDS file.

  For files in CI-format, use DTFSD and DTFCP with DISK=YES.

  For files that are either in CI-format or non-CI-format, use DTFPH for disk with MOUNTED=SINGLE.

  SAM access is provided for all CKD and FBA devices that are supported by VSE/VSAM.
- Allocate dynamic secondary space during creation or extension of a SAM ESDS file.
- Access a SAM ESDS file through the VSE/VSAM macro ACB (that is, native VSE/VSAM) for files in CI-format.

  An existing VSE/VSAM program that processes a VSE/VSAM ESDS file can access a SAM ESDS without change (except for extending the file).

VSE/VSAM Backup/Restore Function

You can use this function to back up VSE/VSAM files to magnetic tape or disk devices, and restore the files again into a VSE/VSAM data set. You can use the two IDCAMS commands BACKUP and RESTORE.

Use the function to:

- Write and read data sets as follows:
  - Write from disk to magnetic tape or disk (BACKUP).
  - Read from magnetic tape or disk to disk (RESTORE).

  You can perform these operations for the following VSE/VSAM objects:
  - KSDS files
  - ESDS files
  - RRDS files
  - VRDS files
  - Alternate indexes
  - SAM ESDS files in CI format
  - Paths
- Handle several VSE/VSAM files with a single command, either with a generic name or as files of one catalog.
- Restore VSE/VSAM files to locations, volumes, and device types that are different from those where the files were before.
- Exclude files from a collective back up or restore operation.
- Tune the performance of VSE/VSAM by specifying the size of the buffers in the BACKUP command, and the number of buffers in both the BACKUP and RESTORE commands.

Also, the *VSE/VSAM Backup/Restore Function* allows you to:

- Back up and restore empty objects, where an empty object may be either a:
  - VSE/VSAM object defined with NOALLOCATION (such as a default model or a dynamic file), or
  - VSE/VSAM cluster that has not been loaded since being defined or reset.
- Change the allocation size for the data component of a file at restoration. You can specify allocation size in device-independent units by using the RECORDS parameter when the cluster is defined to facilitate restoration of objects.
- Change the index CI-size at restoration.

Note: You cannot process magnetic tape files that were created by the EXPORT command with RESTORE, or magnetic tape files that were created by BACKUP with the IMPORT command. REPRO files can only be processed by using REPRO.

Concepts of Data Organization

The following provides you with basic information (terminology and concepts) with which you have to be familiar to understand the information in other parts of this book.

File Types

IBM VSE/VSAM supports four types of physical file organization:

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

These files differ in the record lengths they allow and in the sequence in which they contain the records:

Type	Record Length	Sequence by
ESDS	Fixed or variable	Entry
KSDS	Fixed or variable	Key field
RRDS	Fixed only	Record number or entry
VRDS	Fixed or variable	Record number or entry

Formats of Files

The following figures show the file organization (or file format) for the different file types:

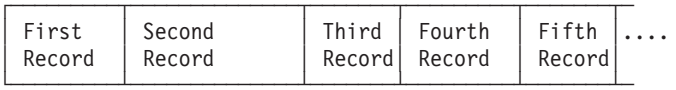


Figure 1. ESDS File Format: Records Stored as Received

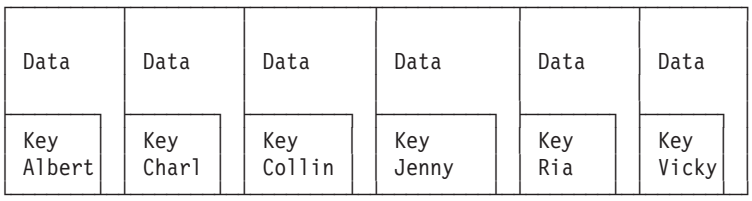


Figure 2. KSDS File Format: Records Stored in Key Field Sequence

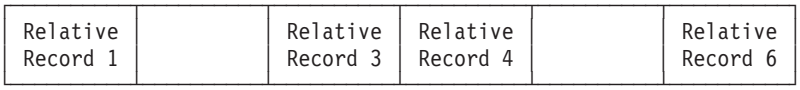


Figure 3. RRDS File Format: Fixed-Length Records Stored by Record Number. RRDS records are entered in one of two ways: Either you give the records a sequence number explicitly, or you just enter them one by one and they get their sequence number automatically.

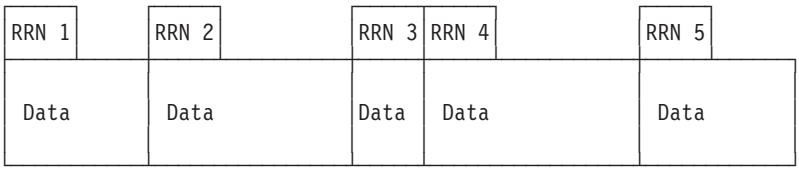


Figure 4. VRDS File Format: Variable-Length Records Stored by Record Number

Elements of Organization

Data Space

For the data that you want to include in VSE/VSAM files, you have to define *VSE/VSAM data space*. You define the space on a disk volume for exclusive use by VSE/VSAM. From this space, VSE/VSAM selects whatever room (control area) is needed for a file when it is defined. VSE/VSAM allocates space dynamically; that is, if a file or a catalog must be extended, VSE/VSAM allocates the space as required.

The VSE/VSAM data space you define is owned by a catalog. You establish the ownership when defining a catalog or data space.

Control Area (CA)

For a definition of "control area", refer to the "Glossary" on page 363.

A file occupies one or more control areas (CAs). Note that:

- For count-key data (CKD) devices, a CA cannot be larger than a cylinder and not smaller than a track. For details, refer to Table 6 on page 87.
- For every type of fixed block architecture (FBA) device, specific maximum and minimum CA-sizes exist. For details, refer to Table 7 on page 87.

VSE/VSAM determines the CA-size for a file, but you can influence it through the space allocation parameters of the IDCAMS command DEFINE CLUSTER.

Control Interval (CI)

For a definition of "control interval", refer to the "Glossary" on page 363.

Control intervals (CIs) are fixed-length parts of a CA. They are the unit of transfer between processor and external storage. You specify a CI-size for the file in the IDCAMS command DEFINE CLUSTER; however, if your specification is inappropriate, VSE/VSAM determines the correct CI-size.

Spanned Records

VSE/VSAM allows records to extend across, or span, CI boundaries. Such records are called spanned records. Spanned records can be used only in KSDS and ESDS files.

A spanned record must always begin on a CI boundary; such a record occupies two or more CIs within a given CA. The CI with the last portion of a spanned record may contain unused space that can be used only to extend the spanned record.

Clusters

Every type of VSE/VSAM file has a cluster name and a data component name; the cluster name must be different from the data component name.

Depending on the type of file, a *cluster* consists of a data component and corresponding index component, or just the data component. This is illustrated in Figure 5:

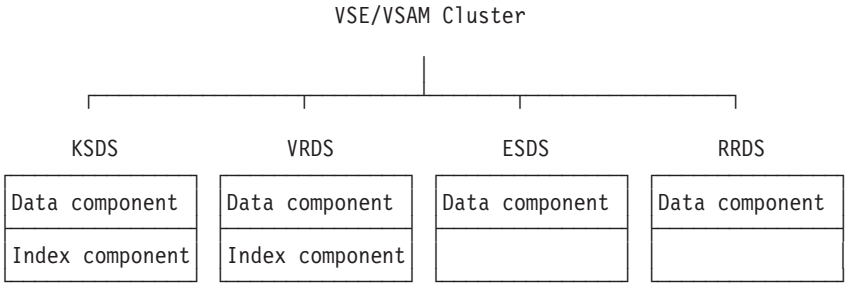


Figure 5. VSE/VSAM File Type Structures

KSDS and VRDS Files: The file types KSDS and VRDS have two components: a data component and an index component (as indicated in Figure 5).

The index component of KSDS and VRDS files are built and used by VSE/VSAM to locate the records in the data component. You can either treat the data and index components separately or together as a single unit. If you treat the two components as a single unit, it is called a “cluster”.

ESDS and RRDS: The file types ESDS and RRDS have a data component, but no index component (as indicated in Figure 5). Nevertheless, these file types are also referred to as “clusters”.

Catalogs with VSE/VSAM

A catalog is a central file that holds information about data spaces and files. VSE/VSAM uses catalogs for space and file management.

For a given environment, you have a *master catalog*, and you can have one or more *user catalogs*. VSE/VSAM creates such catalogs from the information you provide through IDCAMS commands.

Space Management

When you define a VSE/VSAM data space on a volume, you set up a relationship between that data space and a catalog. The data space is owned by the catalog. You can define other data spaces on that same or a different volume in the same catalog.

Thus, a catalog describes where and how much data space is available, the number and device characteristics of the volume, and other values. Whenever data space is allocated to a file, VSE/VSAM automatically updates the data space information in the catalog.

File Management

For each of your VSE/VSAM files, an entry must exist in a catalog. Making an entry in a catalog for a file is called "defining the file". Unless you have defined the file, you cannot, for example, load records into the file.

The entry in the catalog describes the location and attributes (for example, record size and key location) of the file.

Also kept in the catalog are dynamic statistics about the file (such as the number of records inserted since the file was created), and the number of CIs that have been split. This information provides you with the information you need in making a decision to reorganize your files, or for changing the current type of processing so as to improve performance.

Master and User Catalogs

As mentioned above, VSE/VSAM allows you to define several catalogs. This can have significant advantages for performance as well as for data security. Every catalog exists on a single volume; it is independent of other catalogs and controls exclusively its own data spaces and files.

In an environment with several catalogs, one of the catalogs is the *master catalog*. All other catalogs are *user catalogs* and are defined in the master catalog. By placing information about your files and storage volumes into user catalogs, you decentralize control and reduce the time required to search a given catalog. Note that you can have only one user catalog per volume.

Using several catalogs also allows you to:

- Transfer files between the IBM operating systems z/VSE, z/VM and z/OS. You can do so by using the EXPORT/IMPORT commands. ESDS, KSDS, and RRDS files are compatible between these operating systems. VRDS files are incompatible.
- Specify that one of the user catalogs is to be used as a *job catalog*. The job catalog will then be used to reference all VSE/VSAM files in the current job. You have the option of overriding the job catalog reference to a file through a VSE/VSAM job control statement.

Indexes with VSE/VSAM

For KSDS and VRDS files, VSE/VSAM builds an index. This index is called the *prime index*.

For KSDS and ESDS files, you optionally can specify that VSE/VSAM builds an *alternate index* (AIX).

Alternate Indexes - Their Advantages

An alternate index provides you with another way of gaining access to the records in a given KSDS or ESDS file. It eliminates the need for you to keep several copies of the same information organized in different ways for different applications. For example, you can take a KSDS payroll file that is indexed by employee name, and using the same base data, index it according to department number or social

security number (Figure 6). You can use any field in the records of the file as an alternate-index key field, as long as the field has a fixed length and fixed position in the record.

Paths to Base Clusters

To gain access from an alternate index to the file with its prime index (base cluster), you must define a path to it. The path sets up an association between the alternate index and the base cluster (Figure 6). The two alternate indexes shown make the records of the base cluster available to you in different orders.

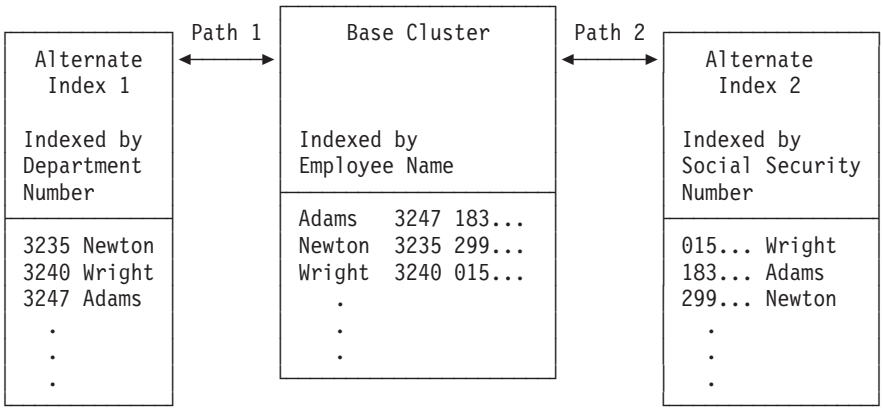


Figure 6. Example: Two Alternate Indexes for a Key-Sequenced File

How to Communicate with VSE/VSAM

To make your wants known to VSE/VSAM, you use:
- Commands of the IDCAMS utility program.
- VSE/VSAM macros.
- Job control (JCL) parameters.
- Dialogs of the z/VSE Interactive Interface.

IDCAMS Commands

The *IDCAMS utility program* is part of IBM VSE/VSAM. Use IDCAMS commands to define VSE/VSAM files, catalog such files, and request many other IDCAMS functions:
- Establish catalog(s)
- Create data spaces
- Create VSE/VSAM files and load records into the files
- Build an alternate index for a file
- Create backup copies of files and their associated catalog entries
- Print, copy, or reorganize files
- Delete files, data spaces, and catalogs
- Alter file definitions and file attributes
- Print catalog entries
- Move catalogs and files from one system to another
- Convert nonVSAM files to VSE/VSAM files
- Recover from damage to files or catalogs
- Copy entire volumes to support offline backup to tape from the target volume, for example
- Verify command syntax
- Merge two VSE/VSAM files

For details on the *IDCAMS utility program* and its commands, refer to the *VSE/VSAM Commands* under "The IDCAMS Utility Program."

The IDCAMS utility program supports two types of IDCAMS commands:

- *Functional commands*

  Used for requesting the actual work (for example, defining a file or moving a catalog).
- *Modal commands*

  Used for the conditional execution of functional commands.

Functional Commands

The functional IDCAMS commands can be grouped according to the following user tasks.

To Define, Alter, and Delete Objects:

DEFINE
to define catalogs, files, clusters, alternate indexes, paths, and data spaces.

ALTER
to change previously-defined attributes of an object.

DELETE
to delete catalogs, clusters, and data spaces.

BLDINDEX
to build an alternate index for an existing file.

To Move Data:

REPRO
to copy, convert, merge, and reorganize files.

EXPORT
to create a copy of a file on tape or disk for back up, or transport to another system.

IMPORT
to read a copy of a file into a system, and make it available for use in that system.

BACKUP
to create a backup copy of a file.

RESTORE
to restore a file backed up via the BACKUP command.

SNAP to snap (copy) a given set of source volumes within an IBM Enterprise Storage Server (ESS).

To Print Objects:

LISTCAT
to list entries from a catalog, or only certain information from every entry.

PRINT
to print all, or a specified range of records of a file. Several output formats are available: every byte printed as a single character, or every byte printed as two hexadecimal digits, or both side by side.

To Correct a Problem, To Cancel a Job or Job Step:

VERIFY
to prepare a file for the next access if it was not closed successfully the last time it was processed.

CANCEL
to cancel either a job or the current job step.

Modal Commands

The modal IDCAMS commands control command execution and establish options.

IF to test a condition code and run according to the results of the test. IF is followed by THEN and ELSE clauses which specify alternative actions.

DO, END
to denote the beginning and end of a functional command sequence (normally within a THEN or ELSE clause).

SET to change condition codes.

PARM to specify diagnostic aids and printed output options and change input record margins. With PARM, you can verify the syntax of your IDCAMS commands before running them.

VSE/VSAM Macros

Once you have defined your VSE/VSAM files with IDCAMS commands, you can load data into the files and process the records. Use VSE/VSAM macros in your programs to process VSE/VSAM files.

You can load the data by use of any programming language. The programs can use VSE/VSAM, SAM, or ISAM macros, but only the assembler language supports all VSE/VSAM functions.

For details on the macros, refer to Chapter 10, “Using VSE/VSAM Macros,” on page 183 and Chapter 11, “Descriptions of VSE/VSAM Macros,” on page 195.

To Relate the Program and the Data (Declarative Macros)

ACB specifies the file to be processed and the access type.

EXLST
specifies a list of user-supplied exit routines.

RPL specifies information for a particular request.

To Handle Declarative Macros

GENCB
specifies declarative parameters during program execution.

MODCB
changes declarative parameters.

SHOWCB
displays declarative parameters in effect.

TESTCB
checks declarative parameters (or their error codes) and sets the condition code accordingly.

To Display Data

SHOWCAT
displays data from the catalog in a buffer you have supplied.

To Connect/Disconnect a Program to/from a File

OPEN connects a program to a file.

CLOSE
prepares the separation and disconnects a program from a file.

TCLOSE
prepares the separation but leaves program and file connected.

To Share Resources Between Several Files (LSR)

BLDVRP
builds a VSE/VSAM pool of buffers, control blocks, and channel programs.

DLVRP
deletes such a resource pool.

WRTBFR
writes waiting buffer contents to satisfy a GET request.

To Handle Records

GET retrieves a record from a file for processing.

PUT inserts a record in a file.

ERASE
deletes a record in a file.

POINT
positions control to a specific address in the file.

ENDREQ
ends processing of a GET or POINT request.

Job Control Parameters to Access VSE/VSAM Files

Use job control parameters to complete or override the file information already stored in the catalog.

As most of the information normally coded with job control statements is available to VSE/VSAM in the catalog, you need to specify only a minimum of job control parameters with any one job. In most cases, only the DLBL statement has to carry VSE/VSAM information.

z/VSE Interactive Interface

The following highlights the functions and use of the *z/VSE Interactive Interface* as applicable to VSE/VSAM. For more information about the "interactive interface", refer to the *z/VSE Administration* manual.

To use the interface, start with the panel entitled *z/VSE Function Selection*. On this panel select:
- *Resource Definition* if you want to manage files or catalogs.
- *Operations* if you want to back up or restore VSE/VSAM objects, or transfer files.

If you select *Resource Definition* and then *File and Catalog Management*, you can make further selections to:
- Display or process a file
- Define a new file
- Define a library
- Define an alternate index (AIX) or name

- Display or process a catalog, or space
- Define a new user catalog.

For example, if you select to define a new file, you can specify elements such as the file ID and name, file organization (for example SAM ESDS organization), and space allocation. You can then select how the job is to be run.

If you select *Operations* and then *Backup/Restore VSE/VSAM Objects*, you can make further selections to export, import, back up, or restore VSE/VSAM files, and back up or restore master and user catalogs.

Chapter 2. Planning Information

This Chapter...

Provides information on **storage requirements** for IBM VSE/VSAM and related programs and utilities. It explains how to calculate the required **partition virtual storage** for your files.

Compatibility with IBM VSE/VSAM Version 2

IBM VSE/VSAM Version 7 is compatible with IBM VSE/VSAM Version 2. Files, programs, and jobs that were created under IBM VSE/VSAM Version 2 can be used without changes.

Overview of Environment and Requirements

IBM VSE/VSAM operates on any:

- IBM processor supported by any operating system under which it runs.
- IBM disk device supported by both VSE/VSAM and the operating system under which it runs.

IBM VSE/VSAM, which is part of VSE Central Functions Version 7 Release 1, 5686-CF7, runs under IBM z/VSE Version 3 Release 1.

What to Consider

Before IBM VSE/VSAM is used, you should plan for the storage needs of VSE/VSAM. Consider storage requirements for:

- IBM VSE/VSAM routines loaded automatically into the SVA during IPL. The routines occupy about 300KB in the SVA (shared virtual area).
- Routines that are not eligible for the SVA.

  See "Partition Space for Non-SVA-Eligible Routines," below. For information on the *ISAM Interface Program*, see "Storage for the ISAM Interface Program (IIP)" on page 18.
- Running VSE/VSAM in real mode.

  See "Space for Running in Real Mode" on page 14.
- VSE/VSAM buffers and control blocks.

  See "Partition Requirement for Buffers and Control Blocks" on page 14.
- The IDCAMS utility program.

  See "Storage for IDCAMS Including the VSE/VSAM Backup/Restore Function" on page 18.

Partition Space for Non-SVA-Eligible Routines

Most VSE/VSAM routines are automatically loaded into the SVA during IPL. Routines that are not reentrant are *not* eligible for SVA (non-SVA-eligible). Therefore, such routines require space in the partition GETVIS area. For example, modules of the *ISAM Interface Program* are non-SVA-eligible and must be loaded into every partition where they are used.

© Copyright IBM Corp. 1979, 2005

The partition where the non-SVA-eligible VSE/VSAM routines are loaded must have at least 128KB plus the partition GETVIS area. Add to this 28KB for the IDCAMS root modules, plus the extra space for every IDCAMS command, which varies between 2KB and 100KB depending on the command.

Device Dependencies

See Chapter 6, "Device Dependencies," on page 77 for information on special functions, restrictions, and exceptions when using certain devices.

Storage for VSE/VSAM

Space for Running in Virtual Mode

Many VSE/VSAM phases run in the SVA. This means that one copy of the VSE/VSAM modules is shared by all partitions.

Space for Running in Real Mode

If you specify REAL in the // EXEC statement, the system loads VSE/VSAM modules that normally reside in SVA into your partition. Your partition must have sufficient storage to accommodate these VSE/VSAM SVA modules.

Running programs in real mode in one partition can degrade the performance in other partitions. For more information, refer to "Format of the EXEC Statement" on page 37 (see REAL).

Partition Requirement for Buffers and Control Blocks

The partition in which VSE/VSAM files are to be processed must allow for a GETVIS area to accommodate VSE/VSAM buffers and control blocks; the user program resides in the same partition, below the GETVIS area.

The size of the partition GETVIS area depends on the number of VSE/VSAM files that are accessed, and on their CI sizes. The minimum requirement is 64KB.

For your files, you need to calculate the total required *partition virtual storage*. You have to consider that:

- Every open catalog needs 14KB for basic buffers and control blocks.
- During open and close processing, an additional 50KB is required for open control blocks and catalog check routines that are used in error analysis.
- Every file needs the partition virtual storage shown in:
  Figure 7 on page 15 if *non-shared resources* (NSR) is specified.
  Figure 8 on page 16 if *local shared resources* (LSR) is specified.

The following applies to both Figure 7 and Figure 8:

- To calculate the requirements for one file, add the values given in *one* column (according to the applicable path/input/output conditions). Complete this calculation for every file; then, add the individual results to obtain the total requirements.
- The buffer space (*n* in the figures) depends on the CI size(s) and on the buffer specifications. If upgrade is done, one set of buffers serves all alternate indexes in the upgrade set. This set of buffers includes two data buffers and one index buffer. (Buffer space can be specified in the IDCAMS command DEFINE, in the DLBL statement, or in the VSE/VSAM macro ACB. For more information, see "Buffer Specification" on page 95.)

- If *data set name sharing* is used, only the *first* cluster of a DSN structure uses the partition GETVIS space as calculated in Figure 7 or Figure 8.

  All the *subsequent* opens to ACBs (for those clusters that are connected to the existing DSN structure) need a minimum GETVIS space of 128 bytes per 28 ACBs.

If NSR is Specified

Item	No Path Specified		Path Specified	
	Input	Output	Input	Output*
Basic requirement (minimum)	7KB	7KB	9KB	9KB
Upgrade set (minimum)	0	(u+1)x2KB	0	u x 2KB
Buffers for base cluster	n	n	n	n
Buffers for alternate index	0	0	n**	n**
Upgrade buffers ***	0	n	0	n
For every string	(S - 1) x 1KB		(S - 1) x 2KB	

```
*   The file must be opened for output only,
    or for both output and input.
**  Always two data buffers and one index buffer.
*** If there is an upgrade set.

u = Number of alternate indexes in the upgrade set.
n = Buffer space.
S = Number of strings.
```

Figure 7. Partition Requirements for Control Blocks and Buffers (with NSR)

A file may exceed *minimum requirements* under any of these conditions:
- If the file has key ranges associated with it.
- If the file has several extents for data or for index.
- If SHAREOPTIONS(4) is used.
- If the length of the key field is very long.
- If the ACB or RPL is not created by GENCB, with the space allocation left up to VSE/VSAM, or if the GENCB requests are not done in the following sequence: 1. GENCB ACB, 2. GENCB RPL.
- If CCW areas are insufficient (see below).

Additional space is required for CCW areas for a) output files that use RECOVERY mode, and b) KSDS files in case of CA split. The size can be calculated by this formula:

```
(CI/CA)÷40
and rounded down to the next 2KB value
```

For example:

```
CI/CA = 450
450÷40 = 11.25
and rounded down = 10

That is, the required additional
GETVIS space is 10KB.
```

If LSR is Specified

If LSR is used to share control blocks among some files, the requirement for the VSE/VSAM resource pools must be taken into account. Refer also to "The BLDVRP Macro" on page 207.

For LSR, virtual storage is equal to the working set.

Figure 8 shows the partition virtual storage requirements when LSR is used.

Item	No Path Specified		Path Specified	
	Input	Output	Input	Output*
Basic requirement (minimum)	3.25KB	3.25KB	5.25KB	5.25KB
Upgrade set (minimum)	0	u x 2KB	0	(u-1)x2KB
Buffers for base cluster	0	0	0	0
Buffers for alternate index	0	0	0	0
Upgrade buffers **	0	0	0	0
For every string	(S - 1) x 1KB		(S - 1) x 2KB	

```
*   The file must be opened for output only,
    or for both output and input.
**  If there is an upgrade set.

u = Number of alternate indexes in the upgrade set.
n = Buffer space.
S = Number of strings.
```

Figure 8. Partition Requirements for Control Blocks and Buffers (with LSR)

To these values, add the requirement for the LSR pool, which consists of:

```
n          The total space specified for buffers.
72p        The space for subpools, where p is the number of subpools.
104b       The number of Buffer Control Blocks; b = number of buffers
s(920 + k) The space for ACB strings, where:
             s = the number of strings.
             k = the maximum key length for files
                 sharing the resource pool.

2048       Space for the channel program area.  Dynamically increase
           this value by 2048 if the resource pool is very active.
```

Round the result to the next page boundary. If you build a large resource pool, the VSE/VSAM working set will be somewhat reduced when resource pool activity is light.

Buffer Allocation above the 16MB Line of Storage

To keep the use of the GETVIS space below the 16MB line as little as possible, VSE/VSAM tries to allocate buffers *above* the 16MB line whenever possible.

The allocation depends on whether sufficient partition GETVIS space is available above the line. If no partition GETVIS space is available above the line,

VSE/VSAM allocates all buffers below the line. This means that buffer handling in partitions residing below the 16MB line is fully compatible with previous buffer handling.

Figure 9 shows the buffer allocation relative to the 16MB line. If you have:
- NSR, refer to the entries for the ACB specifications.
- LSR, refer to the entries for the BLDVRP specifications.

Specification in ACB or BLDVRP	Object	Allocation	
		Below	ANY
ACB RMODE31=ALL	All buffers		x
ACB RMODE31=BUFF	All buffers		x
ACB RMODE31=NONE	All cluster/path NSR data buffers All cluster/path NSR index buffers* All upgrade set NSR buffers *	x	 x x
BLDVRP RMODE31=ALL	All buffers		x
BLDVRP RMODE31=BUFF	All buffers		x
BLDVRP RMODE=NONE	All cluster/path NSR data buffers All cluster/path NSR index buffers* All upgrade set NSR buffers *	x x x	
* The buffers are not accessible by user applications.			

Figure 9. How VSE/VSAM Allocates Buffers

Storage for the VSE/VSAM Space Management for SAM Function

SAM Access Routines

The working set is the same as for unmanaged-SAM access. VSE/VSAM does not need additional work storage for managed-SAM access.

Space for Running in Real Mode

If you specify REAL in the // EXEC statement, the system loads VSE/VSAM modules that normally reside in SVA into your partition. Your partition must have an additional 340KB to accommodate these SVA modules.

Running programs in real mode in one partition can degrade the performance in other partitions. For more information, refer to “Format of the EXEC Statement” on page 37 (see REAL).

Partition Requirement for Control Blocks and Buffers

Specify additional work storage:
- 2KB for control blocks. The buffer must equal CISIZE.

The working set for DTFPH is determined by the user program.

Partition Virtual Storage

Add to the virtual storage requirements for VSE/VSAM:

- During open processing, additional 4KB is needed for open control blocks.
- For every file, the amount of virtual storage required is equal to the working set, except for DTFPH access, in which case the virtual storage requirement is determined by the user program.

GETVIS Requirements for Managed-SAM Access to SAM ESDS Files

When you run programs that issue SAM imperative macros to access managed-SAM files, the default GETVIS size of 48KB is inadequate. For more information, refer to "Note to Users of the VSE/VSAM Space Management for SAM Function" on page 37.

For programs that are invoked by using the EXEC statement, you must specify the SIZE parameter of the EXEC statement to provide adequate GETVIS storage.

For job control routines that process an INCLUDE statement when IJSYSLN has been defined as a managed-SAM file, both the minimum partition size of 128KB and the default GETVIS size of 48KB are too small. Proceed as follows:

1. Use the ALLOC command to adjust the partition size to provide the required GETVIS space, plus 80KB non-GETVIS space for job control routines.
2. Set aside adequate default GETVIS space in the partition with the SIZE command. GETVIS space for file OPEN and catalog handling is the same as for VSE/VSAM. See "Storage for VSE/VSAM" on page 14.

Storage for the ISAM Interface Program (IIP)

IIP modules are *non-SVA-eligible* and must be loaded into every partition where they are used.

To accommodate interface translation modules, add 6KB to the working set previously determined for VSE/VSAM record management modules. Also add approximately 6KB for the IIP phases. This is in addition to the storage required for buffers and control blocks,

Storage for IDCAMS Including the VSE/VSAM Backup/Restore Function

IDCAMS must be used for file definitions, catalog manipulation, and other functions. Because IDCAMS modules cannot be loaded into the SVA, their partition requirement depends on the functions required for the current job.

The required partition GETVIS area can be provided by specifying the job control statement:

```
// EXEC IDCAMS,SIZE=AUTO ...
```

For more information, refer to "// EXEC Statement" on page 36.

To operate efficiently, IDCAMS needs a working set of about 72KB.

In addition to the basic allocation for VSE/VSAM, IDCAMS needs up to 256KB of virtual storage in the partition in which it is to run.

VSE/VSAM Backup/Restore Function

Note: To use the function with a user-generated supervisor, you must generate the supervisor with the option RPS=YES.

Loading VSE/VSAM Backup/Restore into the SVA

At IPL, provide:
10 entries in the SDL
122KB of storage in the SVA

The VSE/VSAM Backup/Restore Function may be loaded into the SVA. Simply store the following statements in the job control procedure for the background partition (in procedure $0JCL):

```
SET SDL
   IDCBP01,SVA
   IDCBP03,SVA
   IDCCDBP,SVA
   IDCTSBP0,SVA
   IDCRT01,SVA
   IDCCDRT,SVA
   IDCBPDNC,SVA
   IDCBPDNT,SVA
   IDCRTDDC,SVA
   IDCRTDDT,SVA
/*
```

Note that you can use the skeleton SKJCL0 to update the job control procedure $0JCL. For more information on "$0JCL and skeletons", refer to the *z/VSE Administration* manual.

Partition Virtual Storage

In addition to basic VSE/VSAM and IDCAMS virtual storage requirements, you must provide sufficient virtual storage in the partition to accommodate the BACKUP or RESTORE command:

```
  BACKUP:   42KB + n * b           If COMPACT parameter is not used.
            42KB + (2n + 1) * b    If COMPACT parameter is used.



  RESTORE:  52KB + n * b           If restoration from a
                                   non-compacted backup file.

            52KB + (2n + 1) * b    If restoration from a
                                   compacted backup file.

where: n = Number of buffers.
       b = Size of one buffer in bytes.
           A buffer size of 32KB is recommended.
           In most cases, there is no advantage
           in providing larger buffers.
```

Chapter 3. Operation and Job Control

This Chapter ...

- Describes **operating procedures** that are unique to VSE/VSAM. You may also need to refer to the manual *z/VSE Operation*, for example, for details on "exporting VSE/VSAM files."
- Describes **job control commands** whose meaning for VSE/VSAM is different as compared to the meaning they have with other access methods.

  The information supplements the "job control" information contained in the manual *z/VSE System Control Statements*.

IPL Command Specifications for VSE/VSAM

The IPL commands described here are part of the pre-defined *automated system initialization* (ASI) procedure for z/VSE. The following specifications are important for VSE/VSAM:

- Assign a device to the master catalog (DEF command).
- Define the lock file (DLF command).
- Specify the number of supervisor buffers (SYS command, BUFSIZE operand).

Assigning a Device to the Master Catalog

To assign a device to the VSE/VSAM master catalog, you must first ready the device. Then use the IPL command DEF with the LU name of the master catalog, which is always SYSCAT:

```
DEF SYSCAT=cuu...

where: cuu = the device number of the
             assigned disk device.
```

The assignment is valid until the next IPL. The DEF command must follow the (optional) SET and precede the DPD command.

Defining the Lock File

If you are using the *z/VSE DASD Sharing facility*, you must define the *lock file* (the cross-system communication file) by specifying the DLF command at IPL.

The number of resources that can be locked by a lock file depends on the device type on which the lock file resides. For detail information, refer to the "DLF" command in the manual *z/VSE System Control Statements*.

Lock File Requirements

VSE/VSAM determines its lock requirements according to this formula:

```
n = Px(2xU+1) + 5C + (2xS3) + (3xS4) + O + P + 5

where:
  n  = number of lock table entries (optimal upper limit).
  C  = number of catalogs open concurrently.
  O  = number of VSE/VSAM components that are open but not
       accounted for by S3 and S4.
  P  = number of partitions.
  S3 = number of share option 3 VSE/VSAM components
```

© Copyright IBM Corp. 1979, 2005

```
     concurrently open for output.
S4 = number of share option 4 VSE/VSAM components (for
     example, key component or data component) concurrently
     open for output.
U  = number of user catalogs open concurrently.
```

All these values should reflect the situation that exists when n is at its maximum value. The value for n (calculated in the above manner) will cause sufficient space to be reserved for the variable resources to be used. Depending on the application, however, the number of resources actually required most of the time might be much lower.

Note: If the value substituted for n is too small and the pool of named resources gets exhausted, the VSE/VSAM partition is canceled and an error message is displayed.

Specifying the Number of Supervisor Buffers for Channel Programs

You must specify the number of supervisor buffers for channel programs. You do this in the BUFSIZE=n operand of the IPL command SYS. For details on this operand, see the "SYS" command in the manual *z/VSE System Control Statements*. After you have determined a value, add 40 for the use by VSE/VSAM.

Volume Mounting

To access VSE/VSAM files, the appropriate volume or volumes must be mounted on a device. There are two approaches that allow you to mount one or more required volumes.

Mounting a Volume Through Job Control Specifications

If full job control describes the file (DLBL, EXTENT, and ASSGN statements), the required volume must be mounted on the device specified in the job control.

If the requested volume (except for a catalog volume) is *not* mounted on the requested device, VSE/VSAM issues a message to inform you; then, you can correct the situation.

You should take advantage of job control simplification (by omitting a LU on an EXTENT statement), because it gives VSE/VSAM greater flexibility in providing the required volume. In this case, VSE/VSAM is free to use any device on which the required volume (as indicated by the VSE/VSAM catalog) is mounted or can be mounted.

Mounting a Volume Through Automatic Assignment

If the Volume is Mounted

If the required volume is already mounted on some device, VSE/VSAM attempts to automatically assign that device (if successful, it avoids the need for operator intervention).

For the automatic assignment to be successful, ensure that devices are up before mounting volumes, and do not reserve devices unnecessarily. Refer to the "DVCUP", "FREE", and "RESERV" commands in the manual *z/VSE System Control Statements*.

If the Volume is Not Mounted

If the required volume is not yet mounted, VSE/VSAM prompts you to mount it. If possible, VSE/VSAM recommends a device and reserves it while the mount is pending.

If you choose to use a device other than the recommended device (or if VSE/VSAM did not recommend one), you must ensure that the device you use is up and operational, and that mounting the required volume does not interfere with other users in the system.

To hold a device while a mount is pending, use the RESERV command. When the volume is mounted, the device becomes ready and the reserved status is reset to free. Your reply to the mount message allows VSE/VSAM to verify the volume mount and continue processing the file.

Use of z/VSE Job Control Statements for VSE/VSAM

In many jobs, you can omit from your job control the following z/VSE job control statements: **// DLBL**, **// EXTENT,** and **// ASSGN**. Under certain circumstances, however, you may have to explicitly specify EXTENT or ASSIGN statements for the catalogs, (for example, if your program uses CHECKPOINT/RESTART).

Table 1 on page 24 shows under which circumstances you have to specify *// DLBL* statements for *job catalogs* and *user catalogs*.

Table 2 on page 25 shows under which circumstances you have to specify job control statements for *files* when you want to run VSE/VSAM applications and want to use IDCAMS commands.

For a detailed explanation of the z/VSE "job control", refer to the manual *z/VSE System Control Statements*.

Job Control Statements for Catalogs

VSE/VSAM Application Programs

All VSE/VSAM application programs must specify a // DLBL statement for the master catalog; no // EXTENT statement is necessary. This also applies to ISAM programs that access VSE/VSAM through the *ISAM Interface Program* (IIP), and SAM programs that access SAM ESDS files through DTFs.

The // DLBL statement may be in the job stream, or in the system or partition standard label area.

If the program accesses a file in a user catalog, you must supply a file // DLBL statement for the VSE/VSAM file. You can refer to the user catalog by either:

- The CAT=filename parameter pointing to that user catalog,

  Or
- A job catalog // DLBL IJSYSUC statement pointing to that user catalog.

Irrespective of which way you specify, you do not need to supply // EXTENT and

// ASSGN statements.

Note that if an application program accesses files in several catalogs, you must supply a *user catalog // DLBL* for all files not in the job's default catalog.

IDCAMS Commands

From the job control that you specify to identify the catalog you are using, you may omit // EXTENT and // ASSGN statements. VSE/VSAM handles the distribution of logical units (LUs) to physical disk addresses in an optimized way. You do not need to reserve one logical unit for every file. However, when you run out of LUs, use // ASSGN statements, or cut the single job into several jobs.

For the master catalog (with filename IJSYSCT), you *always* require a // DLBL statement. Include the statement in the job stream, or in the system or partition standard label area.

For certain operations (for example, to alter file attributes in catalog entries), you can omit the // DLBL statement. You can do so if you specify the name of the catalog through IDCAMS commands. Depending on which IDCAMS command you issue, you have to specify the CATALOG, WORKCAT, or MODEL parameter; in the parameter, specify the name in the subparameter *catname*. Table 1 shows when you must specify a // DLBL statement for a job catalog (IJSYSUC) and, when applicable, for a user catalog (*not* a IJSYSUC).

Table 1. // DLBL Statement Required for Job Catalogs and User Catalogs

ALTER	No job catalog // DLBL statement is required, but you must specify CATALOG(*catname*) in the command if the catalog referenced is not the master catalog, or if a password is required.						
BACKUP	A job catalog // DLBL (IJSYSUC) is required if the catalog to be referenced is not the master catalog.						
BLD-INDEX	Location of Alternate index = = > Work file = = >	MCAT MCAT	UCAT1 MCAT	MCAT UCAT1	UCAT1 UCAT2	MCAT none	UCAT none
	Specify job cat // DLBL?	No	Yes	No	Yes (UCAT1)	No	Yes
	Specify BLDINDEX catalog parameter?	No *	Yes (MCAT) **	Yes (UCAT1) **	Yes (UCAT2) **	No *	No *
(*) Unless a password is required, in which case you must specify the CATALOG parameter. (**) Specify the WORKVOLUMES parameter, because it does not require a // DLBL for the work file. If you specify the WORKFILES parameter, you must also specify CAT= in the // DLBL statement.							

CANCEL	A job catalog // DLBL is not applicable.
DEFINE AIX CLUSTER or PATH	No job catalog // DLBL is required, but you must specify CATALOG(*catname*) and MODEL(*catname*) (if applicable) in the command whenever the catalog to be referenced is not the master catalog, or if a password is required.
DEFINE NONVSAM or SPACE	No job catalog // DLBL is required, but you must specify CATALOG(*catname*) in the command if the catalog to be referenced is not the master catalog, or if a password is required.
DELETE	No job catalog // DLBL is required, but you must specify CATALOG(*catname*) in the command if the catalog to be referenced is not the master catalog, or if a password is required.
EXPORT	A job catalog // DLBL (IJSYSUC) is required if the catalog to be referenced is not the master catalog.
IMPORT	No job catalog // DLBL is required, but you must specify CATALOG(*catname*) in the command if the catalog to be referenced is not the master catalog, or if a password is required.
LISTCAT	No job catalog is required, but you must specify CATALOG*(catname)* in the command if the catalog to be referenced is not the master catalog, or if a password is required.
PRINT	INFILE in master catalog: Do not specify a user catalog // DLBL or a job catalog // DLBL. INFILE in user catalog: Specify either a user catalog // DLBL (CAT=parameter) or a job catalog // DLBL (IJSYSUC). INFILE is nonVSAM: A user catalog // DLBL or a job catalog // DLBL statement is not applicable.
REPRO	INFILE and OUTFILE in same catalog: Master catalog: Do not specify a user catalog // DLBL or a job catalog // DLBL. User catalog: Specify either a user catalog // DLBL(CAT=parameter) or a job catalog // DLBL(IJSYSUC). INFILE and OUTFILE in different catalogs: Specify a user catalog // DLBL for every catalog that is not the default catalog.
RESTORE	No job catalog // DLBL is required, but you must specify CATALOG(*catname*) in the command if the catalog to be referenced is not the master catalog, or if a password is required.
SNAP	A job catalog // DLBL (IJSYSUC) is required if the catalog to be referenced is not the master catalog.
VERIFY	A job catalog // DLBL (IJSYSUC) is required if the catalog to be referenced is not the master catalog, or if a password is required.

Job Control Statements for Files

In specifying job control statements for user files, // DLBL, // EXTENT, and // ASSGN statements may or may not be required. Table 2 indicates when you should specify these statements.

Table 2. Job Control Statements Required for Files

File Job Control			
Type of Processing	**DLBL Required**	**EXTENT Required**	**ASSGN Required**
Files to be implicitly opened. (For example, accessing a file through AIX or path during which VSE/VSAM must open index files without user specification.)	No	No	No
Files to be explicitly opened. (The // DLBL *filename* must match the ACB DDNAME parameter, or the ACB name if DDNAME is omitted, and the *file-ID* must be the name of the object opened.)	Yes	No	No
ISAM programs accessing VSE/VSAM files through the ISAM Interface Program	Yes	No	No
SAM programs accessing SAM ESDS files through DTFs. (The // DLBL *filename* must match the DTFxx name field.)	Yes	No [1]	No

Table 2. Job Control Statements Required for Files (continued)

File Job Control			
Type of Processing	**DLBL Required**	**EXTENT Required**	**ASSGN Required**
IDCAMS Commands			
ALTER	No	No	No
BACKUP to tape	No	No	Yes
BACKUP to disk	Yes	Yes	Yes
BLDINDEX	No	No	No
CANCEL	No	No	No
DEFINE AIX/CLUSTER UNIQUE	Yes	Yes	No
DEFINE AIX/CLUSTER not unique	No	No	No
DEFINE MASTERCATALOG	Yes	No	No
DEFINE NONVSAM/PATH/SPACE	No	No	No
DEFINE USERCATALOG	No	No	No
DELETE	No	No	No
EXPORT OUTFILE (SAM file on disk)	Yes	Yes	Yes
EXPORT all others	No	No	No
IMPORT INFILE (SAM file on disk)	Yes	Yes	Yes
IMPORT OBJECTS FILE UNIQUE unless predefined	Yes	Yes	No
IMPORT all other	No	No	No
LISTCAT	No	No	No
PRINT VSAM file	Yes	No	No
Print nonVSAM file (SAM or ISAM file on disk)	Yes	Yes	Yes
Repro VSAM file	Yes	No	No
REPRO nonVSAM file (SAM or ISAM file on disk)	Yes	Yes	Yes
RESTORE from tape	No	No	Yes
RESTORE from disk	Yes	Yes	Yes
SNAP	Yes	No	No
VERIFY	No	No	No

Note:
[1] Exception: An EXTENT statement *is* required for the implicit definition of an output or work file for which no implicit model exists.

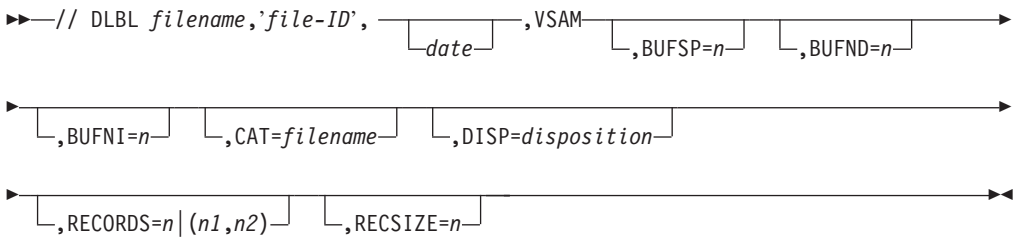
// DLBL Statement

To determine when you must supply a // DLBL statement, refer to "Use of z/VSE Job Control Statements for VSE/VSAM" on page 23.

If you specify many // DLBL parameters, you may need to use a continuation statement. If so, column 72 (on the first statement) must contain a continuation character. The columns between the last comma and the continuation character must be blank, and the continuation statement must start in column 16 (no // in columns 1 and 2).

Format of the DLBL Statement

The following describes the // DLBL statement and its operands in the context of VSE/VSAM.



filename

filename has 1-7 characters; for VSE/VSAM, the following is true:

- filename=dname of the FILE(dname) parameter in an IDCAMS command.
- DDNAME=filename parameter of the Access Method Control Block (ACB) in the processing program that identifies the file. If DDNAME is omitted, the filename must be placed in the symbolic name field of the ACB.

'file-ID'

For VSE/VSAM, specify 'file-ID' when accessing a file. The file-ID is identical to the name of the file that was specified in the DEFINE command of IDCAMS and listed in the VSE/VSAM catalog.

When a new VSE/VSAM data space or file is defined, the file-ID is ignored if it is specified.

date With one exception, this parameter is ignored for VSE/VSAM; the expiration date used is that specified in the IDCAMS DEFINE command.

The only case in which the // DLBL date parameter applies to VSE/VSAM is for implicit definition of managed-SAM files. VSE/VSAM files (that have been explicitly defined) or data spaces can only be deleted through the DELETE command, even though the expiration date has been reached.

VSAM

Indicates a VSE/VSAM file label.

BUFSP=n

For VSE/VSAM, this parameter specifies the number of bytes of virtual storage (0 - 9999999) to be allocated as buffer space for the file. VSE/VSAM uses the maximum of the following:

- The BUFFERSPACE value specified in the IDCAMS command DEFINE CLUSTER
- The BUFSP parameter specified in the ACB macro
- The BUFSP parameter specified in the DLBL statement

When you access a cluster using an alternate index, the DLBL BUFSP value applies only to the alternate index.

BUFND=n

Specifies the number of I/O buffers to hold control intervals containing data records. Each buffer is the size of one data control interval. This specification overrides the value given for BUFND in the ACB macro.

When you access a cluster using an alternate index, the DLBL BUFND value applies to both the alternate index and the base cluster.

BUFNI=n
Specifies the number of I/O buffers to hold control intervals containing index records. Each buffer is the size of one index control interval. This specification overrides the value given for BUFNI in the ACB macro. When you access a cluster using an alternate index, the DLBL BUFNI value applies to both the alternate index and the base cluster.

CAT=filename
Specifies the file name (1 through 7 alphameric characters) of the // DLBL statement for the catalog owning this VSE/VSAM file. The system searches only this catalog for the file-ID when the VSE/VSAM file is to be opened. Specify this parameter only if you want to override the system's assumption that the job catalog or, if there is no job catalog, that the master catalog owns the file.

The only IDCAMS commands that use the CAT parameter to specify a non-default catalog are the PRINT and REPRO commands.

Job catalogs are discussed under "Specifying a Job Catalog" on page 42.

DISP=disposition
Specifies what VSE/VSAM is to do with a reusable file during open and close processing.

For **disposition**, you can specify one of the following formats:

```
d1
(d1,d2)
(d1,d2,d3)
```

If you use the parenthesis syntax, each value (but not the separating commas) can be omitted. For example, the following three specifications are equivalent:

```
DISP=NEW
DISP=(NEW,)
DISP=(NEW,,)
```

Figure 10 shows the possible disposition values and applicability for d1, d2, and d3. Figure 11 on page 29 shows the *values* and their meaning. For exact specification, refer to "File Disposition" on page 29.

dn	The value for dn may be:	Applies when the file is:
d1	NEW or OLD	Opened
d2	DELETE, KEEP, or DATE	Regularly closed
d3	DELETE or KEEP	Abnormally terminated

Figure 10. Disposition: Values and Applicability

Value	Indicates that the:
NEW OLD KEEP DELETE	File is to be reset at OPEN File is not to be reset at OPEN File is to be kept at CLOSE Data is to be made inaccessible at CLOSE
DATE	Disposition is the same as for: KEEP if the expiration date has not been reached. DELETE if the expiration date has been reached.

Figure 11. Disposition: Meaning of Values

RECORDS=n | (n1,n2)

This operand applies only with the *VSE/VSAM Space Management for SAM Function*.

The RECORDS (and RECSIZE) parameter is used to determine allocation sizes for implicit file definition during open of a SAM ESDS file.

If you specify RECORDS=n, n indicates the number of records for primary allocation of the file. VSE/VSAM uses a value of 20% of n to indicate the number of records for secondary allocation. Do not specify 0 for n.

If you specify RECORDS=(n1,n2), n1 indicates the number of records for primary allocation, and n2 indicates the number of records for secondary allocation. If you do not specify a value for n2, 20% of n1 is used. Do not specify 0 as a value for n1.

If you specify the RECORDS parameter, you must also specify the RECSIZE parameter.

RECSIZE=n

This operand applies only with the *VSE/VSAM Space Management for SAM Function*.

The RECSIZE (and RECORDS) parameter is used to determine allocation sizes for implicit file definition during open of a SAM ESDS file.

n indicates the average record length for the file. Do not specify 0 as a value for n.

If you specify RECSIZE, you must also specify the RECORDS parameter.

For details on the RECORDS and RECSIZE parameters, see "Defining a SAM ESDS File" on page 158.

File Disposition

Programming Interface Information

Disposition processing applies to reusable files only. Implicitly defined SAM ESDS files are always reusable.

For *VSE/VSAM access*, the options available at OPEN and the disposition of the file at CLOSE depend on the DISP parameter of the // DLBL statement or the MACRF/CLOSDSP parameters of the ACB macro. Options specified for DISP

override those specified for MACRF/CLOSDSP. The default for the DISP parameter depends on the file opened or closed. For VSE/VSAM access, the default is:

```
DISP=(OLD,KEEP,KEEP)

where:
  OLD is the default when the file is opened.
  The first KEEP is the default when the file is normally closed.
  The second KEEP is the default when the job is abnormally ended.
```

For *managed-SAM* access, the options available at OPEN and the disposition of the file at CLOSE depend on the DISP parameter of the // DLBL statement and options specified in the DTF.

Each option of the DISP parameter has a corresponding option in the MACRF/CLOSDSP parameters that causes the same function to be performed. The NEW, OLD, RST, and NRS options apply when the file is opened; KEEP, DELETE, and DATE apply when the file is closed. VSE/VSAM allocates space, resets the file, or implicitly defines a file (for managed-SAM access of a SAM ESDS file that is not already defined in the catalog) when the ACB/DTF for the file is opened. At close, VSE/VSAM keeps, resets, deallocates, or deletes the file, depending on which function has been specified.

Figure 12 through **Figure 16** list the VSE/VSAM actions when opening and closing different kinds of files. The following definitions apply to the figures:

- **Keep** means to retain a file's data and accessibility.
- **Reset** means to set a file to empty and release its secondary extents.
- **Deallocate** means to set a file to empty and release its primary and secondary extents.
- **Allocate** means to provide primary disk space, as specified by the user at DEFINE time.
- **Define** means to place information describing the file into the VSE/VSAM catalog.
- **Delete** means to remove all references to the file from the catalog, and release the file's space.

OPEN Disposition

A file may appear in one of the following four states when it is opened for output:

Unallocated: A file is unallocated if its catalog records have no information of suballocated space. This happens for one of two reasons. Either the file was defined as a dynamic file (NOALLOCATION, REUSE) and it has never been opened, or the file was defined as a dynamic file (NOALLOCATION, REUSE) and it has been closed with DISP and/or MACRF/CLOSDSP options that caused deallocation.

All open options cause space to be suballocated for the file, providing enough space is available. If enough space is not available, the open fails. The ACB user is informed by an ACB return code; the DTF user's job is canceled.

Allocated for Native VSE/VSAM Access: The options DISP=NEW and/or MACRF=RST cause the file to be reset to its primary allocation; its secondary extents are released. Although the file records are not erased, the file is considered empty. The options DISP=OLD and/or MACRF=NRS do not cause the file to be reset to empty and allow updating and extension of the file.

Allocated for Managed-SAM Access of a SAM ESDS File: Same as for "Allocated for Native VSE/VSAM Access" on page 30, above.

Undefined for Managed-SAM Access: All options of the DISP parameter cause the file to be implicitly defined. The native VSE/VSAM user cannot implicitly define a file.

When a file with suballocated space is opened for input, the options DISP=NEW and/or MACRF=RST are invalid, and the options DISP=OLD and/or MACRF=NRS cause the file to be opened without resetting the file to empty.

Figure 12 shows the action performed by VSE/VSAM when you try to open a file that is allocated for *VSE/VSAM access*.

Files with REUSE Attribute			
OPEN (ACB)	**File Status**	**DISP on DLBL or MACRF on ACB**	
		NEW/RST	**OLD/NRS**
OUT	Unallocated	Allocate space for the file.	Allocate space for the file.
	Allocated	Reset the file. (DISP=NEW prevents access to any data that exists prior to open.)	File is not reset. Output operations allow updating and extension of the file.
	Undefined	Open fails.	Open fails.
IN	Allocated	Open fails.	File is not reset. If the file is already empty, open fails.

Figure 12. VSE/VSAM Access: OPEN Disposition

Figure 13 on page 32 and **Figure 14 on page 33** show the action performed by VSE/VSAM when you try to open a file that is allocated for *managed-SAM access*. For explanations to the "See ( )" references in the two figures, refer to "Explanations" on page 34.

Files with REUSE Attribute				
OPEN (DTF)	File Status	DISP is NOT Specified See (A)	DISP on DLBL See (B)	
			NEW	OLD
OUT-PUT	Unallocated SAM ESDS file. See (1)	Allocate space for the file. (DISP=NEW).	Allocate space for the file.	Allocate space for the file.
	Allocated for SAM ESDS file. See (1)(2)	Reset the file. Position to the beginning of the file. (DISP=NEW). See (B)	Reset the file. Position to the beginning of the file.	File is not reset. Position to the end of the file for extension.
	Undefined.	Implicitly define a SAM ESDS file. (DISP=NEW)	Implicitly define a SAM ESDS file.	Implicitly define a SAM ESDS file.
INPUT	Allocated for SAM ESDS file.	File is not reset. Position to the beginning of the file for input. (DISP=OLD)	Invalid. File is not reset. Open fails.	File is not reset. Position to the beginning of the file for input.

Figure 13. Managed-SAM Access: OPEN Disposition -- OUTPUT/INPUT

Files with REUSE Attribute				
OPEN (DTF)	File Status	DISP is NOT Specified See (A)	DISP on DLBL See (B)	
			NEW	OLD
WORK	Unallocated SAM ESDS file. See (1)	Allocate space for the file. (DISP=NEW).	Allocate space for the file.	Allocate space for the file.
	Allocated for SAM ESDS file. See (1)	Reset the file. Position to the beginning of the file. (DISP=NEW).	Reset the file. Position to the beginning of the file.	File is not reset. Position to the beginning of the file The file may be read.
	Undefined.	Implicitly define a SAM ESDS file. (DISP=NEW)	Implicitly define a SAM ESDS file.	Implicitly define a SAM ESDS file.
WORK-IN See	Unallocated SAM ESDS file.	Invalid. Open fails. (DISP=OLD)	Invalid. Open fails.	Invalid. Open fails.
	Allocated for SAM ESDS file.	File is not reset. Position to the beginning of the file. The file may be read. (DISP=OLD)	File is not reset. Position to the beginning of the file. The file may be read.	File is not reset. Position to the beginning of the file. The file may be read.
	Undefined.	Invalid. Open fails. (DISP=OLD)	Invalid. Open fails.	Invalid. Open fails.
WORK-INUP See	Unallocated SAM ESDS file.	Invalid. Open fails. (DISP=NEW)	Invalid. Open fails.	Invalid. Open fails.
	Allocated for SAM ESDS file.	File is not reset. Position to the beginning of the file. The file may be read. (DISP=NEW)	File is not reset. Position to the beginning of the file. The file may be read.	File is not reset. Position to the beginning of the file. The file may be read.
	Undefined.	Invalid. Open fails. (DISP=NEW)	Invalid. Open fails.	Invalid. Open fails.

Figure 14. Managed-SAM Access: OPEN Disposition -- WORKxxxx (Part 1 of 2)

Files with REUSE Attribute				
OPEN (DTF)	File Status	DISP is NOT Specified See (A)	DISP on DLBL See (B)	
			NEW	OLD
WORK-MOD	Unallocated SAM ESDS file. See (1)	Allocate space for the file. See (1) (DISP=NEW).	Allocate space for the file.	Allocate space for the file.
	Allocated for SAM ESDS file.	Reset the file. Position to the beginning of the file. (DISP=NEW).	Reset the file. Position to the beginning of the file.	File is not reset. Position to the end of the file The file may be read.
	Undefined.	Implicitly define a SAM ESDS file. (DISP=NEW)	Implicitly define a SAM ESDS file.	Implicitly define a SAM ESDS file.

Figure 14. Managed-SAM Access: OPEN Disposition -- WORKxxxx (Part 2 of 2)

Explanations: In Figure 13 on page 32 and Figure 14 on page 33:

(A) The default value is given in parentheses, for example (DISP=NEW).

(B) Do not specify the DISP parameter for IJSYSLN (SYSLNK file).

(1) If the characteristics of the file do not match those specified in the DTF, open fails and the file cannot be implicitly deleted. In particular, the maximum logical block that may be written (DTF BLKSIZE) must not be greater than the maximum allowed in the file (DEFINE maximum RECORDSIZE). If DTFSD is used, the file must be in CI format.

(2) DISP=NEW prevents access to any data existing prior to open.

CLOSE Disposition

Close disposition takes effect only after the file has been successfully opened. If you open a file but do not close it, close disposition takes effect during automatic close at the end of the job step.

VSE/VSAM Access: If you specify DELETE as the only disposition at CLOSE, VSE/VSAM always deletes the data by deallocation or resetting the file. The contents of the file is lost. The next open for INPUT will fail because the file is empty. If any other DTF (or ACB) is open for the same file, the close is completed, but the file is not reset, deallocated, or deleted; the operator and the invoking program are notified by a return code.

If you specify a *second* close disposition in the // DLBL DISP parameter, this specification takes over the function of the first close disposition if the job is canceled by the operator or is ended abnormally for any other reason before the file was closed.

Note: A nonzero job return code is *not* an abnormal end of job. This means:

- The *first* close disposition will be performed.

- The *second* close disposition will *not* be performed.

If, for example, you open a reusable file through a // DLBL statement with the close disposition specified as ...DELETE,KEEP then this file is only deleted if the job comes to a *normal* end. In any other case the file is not deleted and you can rerun the job without reloading the file.

Figure 15 shows the action performed by VSE/VSAM when you try to close a file that is allocated for *VSE/VSAM access*.

Files with REUSE Attribute				
CLOSE (ACB)	DISP on DLBL or MACRF on ACB			
	KEEP	DELETE	Date	
			Expired	Unexpired
File was explicitly defined (NOALLOCATION).	Keep	Deallocate	Deallocate	Keep
Reusable (suballocated).	Keep	Reset	Reset	Keep
File was implicitly defined.	Keep	Reset	Reset	Keep

Figure 15. VSE/VSAM Access: CLOSE Disposition

Managed-SAM Access: If you specify DELETE as the only disposition at CLOSE, VSE/VSAM always deletes the data by deallocation, resetting, or implicit deletion. To avoid sharing-problems, however, if any other DTF (or ACB) for the same file is open at the same time, no deletion occurs; the operator is notified by a message with a warning return code, and close processing continues. With DELETE specified at CLOSE, the contents of the file is lost. The next open for OUTPUT WORK will write new data. If the file has been deallocated or reset, an OPEN for INPUT will be successful, but the first GET will cause control to be passed to the EOFADDR routine because the file is empty.

If you specify a *second* close disposition in the // DLBL DISP parameter, this specification takes over the function of the first close disposition if the job is canceled by the operator or is ended abnormally for any other reason before the file was closed.

Note: A job return code of *not 0* is *not* an abnormal end of job. That means:
- The *first* close disposition will be performed.
- The *second* close disposition will *not* be performed.

If, for example, you open a reusable file through a // DLBL statement with the close disposition specified as ...DELETE,KEEP then this file is only deleted if the job comes to a *normal* end. In any other case the file is not deleted and you can rerun the job without reloading the file.

Figure 16 on page 36 shows the action performed by VSE/VSAM when you try to close a file that is allocated for *managed-SAM access*.

Files with REUSE Attribute					
CLOSE (DTF)	DISP not specified	DISP on DLBL See (A)			
		KEEP	DELETE	Date	
				Expired	Unexpired
File was explicitly defined (NOALLOCATION).	Keep See (1)	Keep	Deallocate	Deallocate	Keep
Reusable (suballocate).	Keep See (1)	Keep	Reset	Reset	Keep
File was implicitly defined.	Keep See (1)	Keep	Delete	Delete	Keep
(A) Do not specify the DISP parameter for IJSYSLN (SYSLNK file). (1) DISP is DELETE if TYPEFILE=WORK and DELETFL is specified.					

Figure 16. Managed-SAM Access: CLOSE Disposition

Additional Considerations

- Specifying DISP=NEW in the // DLBL statement overrides MACRF=NRS in the ACB, such that the result is as if MACRF=RST were specified. Because MACRF=RST is mutually exclusive with MACRF=IN and MACRF=LSR, open fails if DISP=NEW is specified for a file opened through DTF TYPEFLE=INPUT, or ACB MACRF=IN, or MACRF=LSR.
- If the close disposition specified for the file results in the resetting or deallocation of the file, and if the file is password-protected, the ACB must specify (or the operator will be prompted for) the update- or higher-level password of the file at open. If the close disposition specified for the file results in the implicit deletion of the file, there is no prompting for the entry password because an implicitly defined file cannot be password-protected. If the catalog itself is password-protected, the operator is prompted for the master password of the catalog at CLOSE.

  Using DISP could eliminate data inadvertently if the wrong parameter is specified. You may want to use an entry password to protect against inadvertent destruction of data. A catalog password may also provide protection for files owned by the catalog. If the file is accessed through DTF, the password must be supplied by the operator. If the file is accessed through ACB, the password may be supplied in the ACB, by the operator, or through IDCAMS commands.

End of Programming Interface Information

// EXEC Statement

To run a job or job step, enter the EXEC command with the SIZE parameter.

Note to Users of the VSE/VSAM Space Management for SAM Function

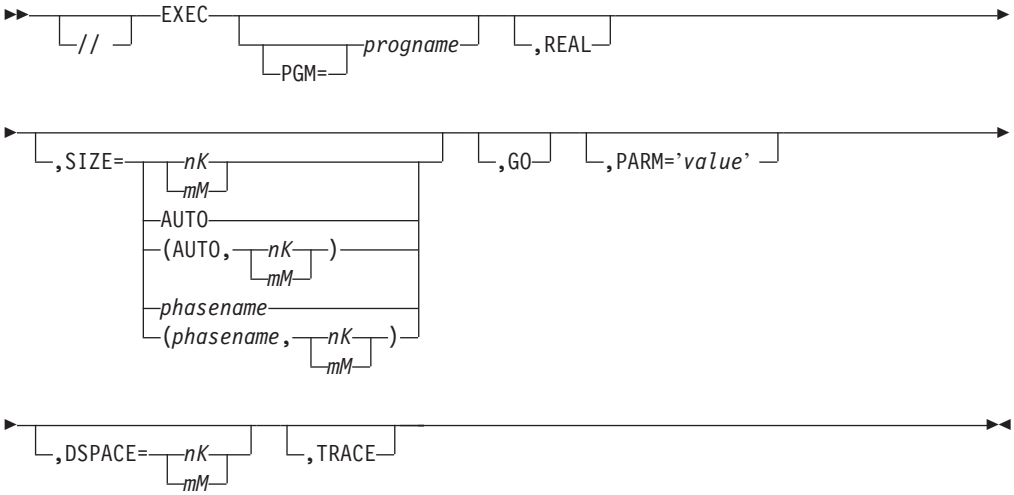
When job control routines process linkage editor control statements (such as ACTION, ENTRY, INCLUDE, or PHASE) with IJSYSLN defined as a managed-SAM file, both the minimum partition size of 128KB and the default GETVIS space of 48KB are too small. Before attempting to run the linkage editor with a managed-SAM IJSYSLN file, tell the system operator to issue the ALLOC command to adjust partition size to provide the required GETVIS space, plus 80KB non-GETVIS space for job control routines. The operator must then issue the SIZE *command* to set aside an adequate amount of default GETVIS space in the partition.

To determine how much GETVIS space is adequate, consider the following:

- You must provide enough storage to access the catalog(s) to locate the SAM ESDS file(s). For every catalog required, provide at least 40KB of GETVIS space.
- For every SAM ESDS file you wish to have open at any given time, you should provide at least an additional 20KB of GETVIS space.

For example, to use 4 work files cataloged in the same user catalog, provide 40KB (user catalog) + 40KB (master catalog) + 80KB (4 SAM ESDS files) = 160KB of GETVIS space. This is in addition to the space for the program you intend to run in that partition.

Format of the EXEC Statement



For information about parameters not described here, see "EXEC" in the *z/VSE System Control Statements*.

REAL Tells the system to execute the program in real storage, without paging. In VSE/VSAM, use of this parameter causes the system to load VSE/VSAM modules that normally reside in the SVA into your partition. Your partition must have an additional 300KB to accommodate these VSE/VSAM SVA modules. To run the *VSE/VSAM Space Management for SAM Function* in real mode, add another 40KB (340KB total) to your partition.

There are only a few cases (for example, time-dependent applications) in which VSE/VSAM should run in real mode instead of virtual mode. Running programs in real mode in one partition can significantly degrade performance in other partitions, so you should use real mode sparingly. Do

not specify REAL on the // EXEC IDCAMS invoke statement, because the partition cannot accommodate both VSE/VSAM and IDCAMS modules at the same time.

SIZE=size

Specifies how much storage is needed for loading the specified program. For ease of use, specifying SIZE=AUTO is recommended. This indicates that the program size, as calculated by the system, is to be taken as the value for SIZE. For other possible specifications of SIZE, see "EXEC" in the *z/VSE System Control Statements*.

You must specify SIZE for VSE/VSAM programs (including IDCAMS), ISAM programs using the *ISAM Interface Program* (IIP), and SAM files using the *VSE/VSAM Space Management for SAM Function*. SIZE specifies the size of that part of the partition that is directly available to the program to be executed. The remainder of the partition may be used as GETVIS storage area.

The non-SVA-eligible VSE/VSAM phases and IDCAMS must be accommodated in the partition GETVIS area. The partition GETVIS area must contain at least 40KB for VSE/VSAM buffers and control blocks for every catalog that is open, plus 12KB for every KSDS and 10KB for every ESDS or RRDS (assuming a CI size of 2KB or less). Additional space for modules, buffers, and control blocks is required if you use any non-SVA-eligible VSE/VSAM phases (for example, the *ISAM Interface Program*) or IDCAMS. For exact storage requirements, see "Storage for VSE/VSAM" on page 14.

To **invoke** IDCAMS through job control, specify:

```
// EXEC IDCAMS,SIZE=AUTO,PARM='value'
```

If you do not specify the SIZE parameter, IDCAMS terminates your job immediately. When you specify SIZE=AUTO, the system determines the amount of storage required for the IDCAMS root segment and leaves the rest of the partition free for the GETVIS area.

PARM='value'

The parameter is optional. The syntax and meaning of the PARM parameter are identical to that of the PARM IDCAMS command (modal command) as described in the *VSE/VSAM Commands* under "PARM."

Follow IDCAMS syntax rules for coding 'value', but follow VSE/VSAM job control rules for coding continuation statements. (Use a nonblank character in column 72 and continue the statement in column 16.) The maximum number of characters between the quotes is 100 and consists of all data in columns 16 - 71, including blanks. IDCAMS treats this data as one 100-character line. Do not code the IDCAMS continuation dash.

Examples:

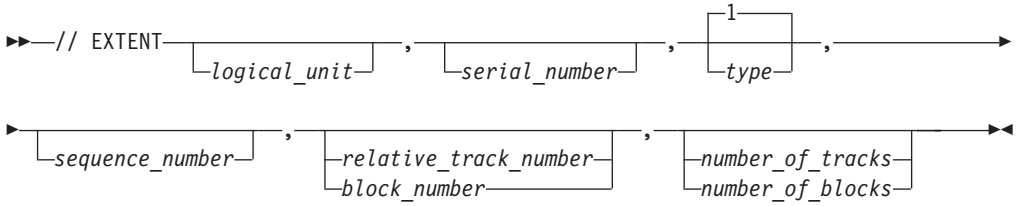
```
                16                                                  72
                 |                                                   |
                 V                                                   V

  // EXEC IDCAMS,SIZE=AUTO,PARM='GRAPHICS(CHAIN)                     X
                 MARGINS(10 80)'

  // EXEC IDCAMS,SIZE=AUTO,PARM='TEST(TRACE FULL((IOGR 3 3)          X
                 (IOPR 3 3)(IOVY 4 4)))'
```

```
In the examples, there is a continuation character in column 72,
and the continuation line begins in column 16.
In the second example, the FULL dump IDs are IOGR, IOPR, and IOVY.
```

// EXTENT Statement

To determine when you must supply a // EXTENT statement, refer to "Use of z/VSE Job Control Statements for VSE/VSAM" on page 23.

Format of the EXTENT Statement



logical_unit
Specifies a six-character field indicating the logical unit (SYSxxx) of the volume on which this extent resides. VSE/VSAM does not require this parameter; if you do not specify a LU, VSE/VSAM will assign one.

If you specify this parameter, you must supply full job control statements (// DLBL, // EXTENT, and // ASSGN) for *all* volumes (including candidate volumes) for the file and its associations.

serial_number
VSE/VSAM users are required to specify the serial number of the volume this extent is on. For data integrity reasons, do not have two volumes with the same serial number in your system (even if one of the volumes contains no VSE/VSAM space).

type For VSE/VSAM, a value of 1 is assumed.

sequence_number
This parameter is ignored for VSE/VSAM users, but if it is specified incorrectly, it is flagged by job control.

relative_track_number|block_number
This parameter indicates the number of the track (CKD), or number of block (FBA) on which the extent is to begin. You must specify it when a file with the UNIQUE option is created (DEFINE or IMPORT command).

This parameter is not required (and is ignored) if it is specified for a VSE/VSAM file that is created within an existing data space. In this case, VSE/VSAM suballocates the space for the file from direct-access extents it already owns. You are not required to specify this parameter for a VSE/VSAM input file, because the extents are obtained from the VSE/VSAM catalog.

This parameter and the number of tracks|blocks parameter must either both be present or both be omitted.

number_of_tracks|number_of_blocks
This parameter indicates the number of tracks (CKD), or number of blocks (FBA) to be allocated to the file or space. You must specify it when a file with the UNIQUE option is created (DEFINE or IMPORT command).

This parameter is ignored when a VSE/VSAM file is created within an existing data space, because VSE/VSAM suballocates the space for the file from direct-access extents it already owns. This parameter is not required for VSE/VSAM input files, because the extents are obtained from the VSE/VSAM catalog.

For an implicitly defined SAM ESDS file that does not specify RECORDS and RECSIZE (on the // DLBL statement), VSE/VSAM uses the number of tracks|blocks parameter to determine the primary allocation size. A secondary allocation size equal to 20% of the primary size is used.

This parameter and the relative track|block parameter must either both be present or both be omitted.

Using Job Control for Catalog Definition

Overview of Catalogs

VSE/VSAM catalog(s) are central information points for files and volumes. A catalog contains the information VSE/VSAM needs to allocate space for files, verify authorization to gain access to files, compile use statistics on files, and relate relative byte addresses (RBAs) to physical locations. Each VSE/VSAM catalog also contains entries that describe the catalog itself. Figure 17 shows the *relationship* between a *master catalog* and *user catalogs*, as well as their relationships with VSE/VSAM and nonVSAM files.

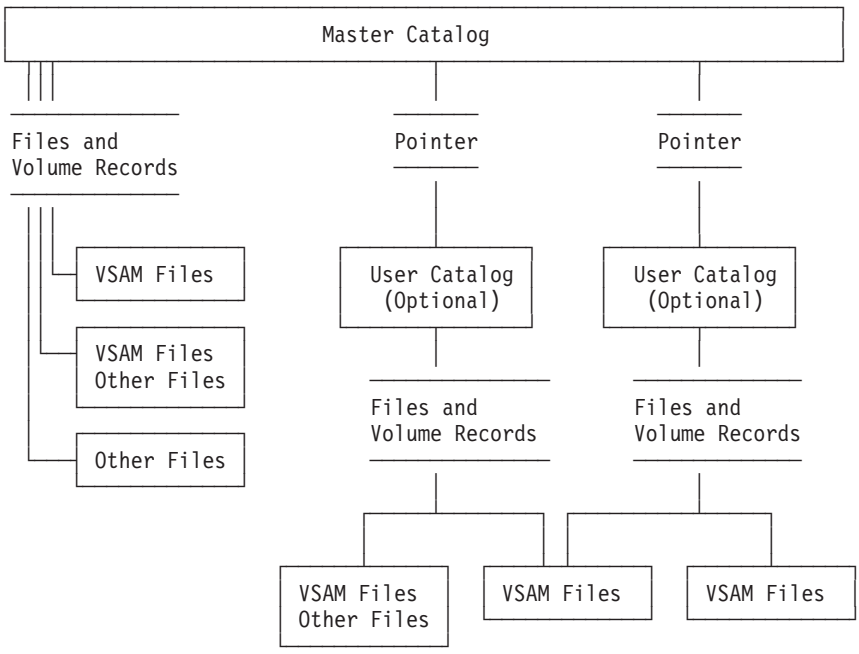


Figure 17. Relationship of Catalogs and Files

Master Catalogs

This type of catalog is mandatory. A master catalog *must* be defined in your system. Defining a master catalog is the first job that needs to be done after you have installed VSE/VSAM in your system. You can have several master catalogs at your installation; however, only one can be connected to the system at a time.

The master catalog volume must always be mounted whenever a VSE/VSAM file or catalog is to be processed. If the VSE/VSAM file to be processed is defined in a user catalog, the user catalog volume must be mounted also.

The master catalog volume is connected to the system at IPL (initial program load) by the DEF SYSCAT=cuu command. It is always on a LU named SYSCAT.

User Catalogs

This type of catalog is optional. A user catalog has the same structure and function as the master catalog. If defined, a user catalog is pointed to by the master catalog.

One or more user catalogs can be defined in your system. They are used to increase data integrity and security, improve performance, and provide volume portability.

Files and Catalogs

All VSE/VSAM files (except implicitly defined SAM ESDS files) *must* be defined (have an entry) in a catalog. To make such an entry, or to perform other actions on a file, you do *not* act on the file, but on a VSE/VSAM catalog. For example: to establish a file, you have to create an entry in a catalog by using the IDCAMS command DEFINE; to delete a file involves removing an entry from the catalog; to move a file from one system to another involves moving an entry from one system's catalog to another system's catalog.

Note that VSE/VSAM either:

- Uses a catalog to *access a file* (as in the PRINT command, where VSE/VSAM locates the file to be printed through the catalog), or
- Accesses the catalog information only and does not access a file (as in the ALTER command, where VSE/VSAM changes an entry in the catalog).

Catalog Volumes

Several catalogs can own space on a volume, but with the restriction that only one catalog can *reside* on a volume.

If a VSE/VSAM file resides on several volumes, every one of those data spaces must be owned by the same catalog.

Note that information requests to a catalog might be answered more quickly if the information is distributed across several catalogs. For example, if the master catalog primarily contains pointers to user catalogs, which in turn contain entries for most files and volumes, catalog search time can be reduced, and the effect of an inoperative or unavailable catalog is minimized.

The following discussion (except where noted) pertains to the accessing of a file.

Specifying the Master Catalog

To define a master catalog you must supply a *master catalog // DLBL* statement, and you must specify extent information, either in the form of an // EXTENT statement or by DEFINE command parameters.

```
// DLBL    IJSYSCT,'VSAM.MASTER.CATALOG',,VSAM
```

The // DLBL statement in the above example identifies:

- The filename: must be IJSYSCT.
- The file-ID: VSAM.MASTER.CATALOG

This can be any name you choose (it must match the NAME parameter in the DEFINE MASTERCATALOG command).

- The access method.

You can omit the master catalog // DLBL statement from the job stream if you place the statement in the system or partition standard label area. You do this by preceding it with *one* of the following job control statements:

```
// OPTION   STDLABEL=ADD
// OPTION   PARSTD=ADD
```

Another way of referring to the master catalog (after its initial specification) is by coding the CAT=filename parameter in a VSE/VSAM file's // DLBL statement. For further explanation to the CAT=filename parameter, see below.

Specifying a User Catalog

To *define* a user catalog you supply a // DLBL statement for the *master* catalog only. But to *access* files in a user catalog, specify a user catalog // DLBL statement. (For information on // DLBL requirements for IDCAMS commands, refer to Table 1 on page 24.) No // EXTENT statement is required.

Specifying a Job Catalog

With VSE/VSAM, you can designate one (but only one) of your user catalogs as a *job catalog*.

You specify a job catalog by coding the filename, IJSYSUC, in the // DLBL statement that specifies the user catalog; for example:

```
// DLBL    IJSYSUC,'JOBCAT',,VSAM
```

When you specify a job catalog, VSE/VSAM will always use that one catalog for all catalog and file access in the current job, unless it is specifically overridden by:

- The CAT=filename parameter of a VSE/VSAM file's // DLBL statement.
- The CATALOG or WORKCAT parameter of an IDCAMS command.

Using a Job Catalog

The following example makes use of the REPRO command (data is to be copied from one file to another) to show how you use a job catalog. It is assumed that the input file, PAY, and the output file, PAYROLL, were already defined (cataloged) in the job catalog. It is also assumed that the // DLBL statement for the master catalog has been placed in the system or partition standard label area and so need not be included in the example.

Example:

```
     // JOB      Specify a job catalog
(a)  // DLBL     IJSYSUC,'USER1',,VSAM
(b)  // DLBL     VSAMIN,'PAY',,VSAM
(c)  // DLBL     VSAMOUT,'PAYROLL',,VSAM
     // EXEC     IDCAMS,SIZE=AUTO
        REPRO    INFILE(VSAMIN) OUTFILE(VSAMOUT)
     /*
     /&
```

In this example, VSE/VSAM finds a // DLBL statement with a filename of IJSYSUC (a). VSE/VSAM interprets this to mean that files PAY (b) and PAYROLL (c) have their respective entries in the job catalog. It, therefore, searches the job catalog to locate the entry for input file PAY and output file PAYROLL.

Assume that you want to process a file, but the file is not cataloged in the job catalog. In this case, you can override the job catalog by using either:
- The // DLBL CAT=filename parameter, or
- The CATALOG parameter of an IDCAMS command.

The following explains these two methods of explicit catalog specifications.

Explicit Catalog Specification (With a VSE/VSAM File's // DLBL CAT Parameter)

The following example directs VSE/VSAM to search a catalog other than the job catalog (specified in the previous example). Assume that the input file PAY was defined in job catalog USER1 as before, but output file PAYROLL was defined in user catalog USER2. Also assume, as before, that the // DLBL statement for the master catalog has been placed in the system or partition standard label area.

Example:

```
     // JOB      USING the // DLBL CAT PARAMETER
(a)  // DLBL     IJSYSUC,'USER1',,VSAM
(b)  // DLBL     VSAMIN,'PAY',,VSAM
(c)  // DLBL     VSAMOUT,'PAYROLL',,VSAM,CAT=PRIVCAT
(d)  // DLBL     PRIVCAT,'USER2',,VSAM
     // EXEC     IDCAMS,SIZE=AUTO
        REPRO    INFILE(VSAMIN) OUTFILE(VSAMOUT)
     /*
     /&
```

VSE/VSAM encounters the filename IJSYSUC in a // DLBL statement (a), but it also finds CAT=PRIVCAT in a file's // DLBL statement (c). CAT=PRIVCAT directs VSE/VSAM to search catalog (d) USER2 (rather than the job catalog) for PAYROLL's file entry. (Filename PRIVCAT links the CAT parameter to the appropriate // DLBL user catalog statement.)

VSE/VSAM locates the entry of file PAY (b) in the job catalog as before because, in this case, you have not overridden the job catalog specification.

The // DLBL CAT=filename parameter is used with the PRINT and REPRO commands. (Each of these commands is used to access data.) The // DLBL CAT=filename parameter can also be used for VSE/VSAM application program access.

Explicit Catalog Specification (With the IDCAMS CATALOG Parameter)

Master Catalog: A master catalog // DLBL statement is always required. You may include it in the job stream or in the partition or standard label area.

User Catalog: If you specify the CATALOG parameter in an IDCAMS command, generally no // DLBL is needed for a user catalog. Only the PRINT and REPRO commands require a user catalog // DLBL because they are used with nonVSAM files. For information on the requirement of // DLBL for IDCAMS commands, see Table 1 on page 24.

```
The format of the CATALOG parameter is:

   CATALOG  (catname/password)
```

Specify `password` only if needed.

Search Sequence of Catalogs

VSE/VSAM follows a certain order in searching for the catalog of a file. The established hierarchy that determines the specific catalog to be searched is as follows:

1. Explicitly specified user or master catalog.

   This is the catalog that is specified by the IDCAMS CATALOG parameter or by the CAT=filename parameter of a VSE/VSAM file's // DLBL statement.

2. Job catalog.

   If the above catalog is not specified, the job catalog (IJSYSUC) specified as the filename of a // DLBL statement is searched.

3. Master catalog.

   If the above catalogs are not specified, the master catalog (IJSYSCT) is searched.

Figure 18 on page 45 shows which catalog is searched, depending on your // DLBL specification.

The *default catalog* is the catalog that VSE/VSAM searches if you do not specify CAT=filename in the // DLBL statement, or if you do not use the CATALOG parameter in an IDCAMS command.

Normally, the default catalog is specified by the // DLBL IJSYSUC statement (also referred to as *job catalog*). If not specified, the default catalog is the master catalog (// DLBL IJSYSCT).

// DLBL IJSYSUC Specified?	// DLBL CAT=filename	CATALOG Parameter Specified? See (1)	Catalog to be Searched
Yes	None	No	Job Catalog
Yes	Filename of User Catalog // DLBL	No	User Catalog
Yes	'IJSYSCT'	No	Master Catalog
Yes	'IJSYSUC'	No	Job Catalog
Yes	None	Yes	CATALOG(catname)
No	None	No	Master Catalog
No	Filename of User Catalog // DLBL	No	User Catalog
No	'IJSYSCT'	No	Master Catalog
No	'IJSYSUC'	No	Master Catalog See (2)
No	None	Yes	CATALOG(catname)

(1) For more information on the CATALOG parameter, see Table 1 on page 24 .
(2) If the filename for the job catalog is specified but not a job catalog, VSE/VSAM defaults to the master catalog.

Figure 18. // DLBL Specifications and Search Sequence of Catalogs

Chapter 4. Tasks under VSE/VSAM

This Chapter ...

Explains the **relationship** between a catalog, the data space on a volume, and VSE/VSAM files.

The chapter includes **how to do** information for:
- Defining data space and files, and handling ownership of space and volumes.
- Transporting files.
- Migrating catalogs and files.
- Modeling a new object from existing definitions.
- Changing recoverability of a catalog.

Data and Space Management

About the VSE/VSAM Catalog

When you define a catalog under VSE/VSAM, the catalog is the first object contained on a volume, and VSE/VSAM allocates a specific amount of data space to the catalog. This data space is "owned" by the catalog, and it is managed by VSE/VSAM. Of this data space, you can make portions available to other VSE/VSAM objects; that is, you can *suballocate* space.

Information Contained in the Entries of a Catalog

The VSE/VSAM catalog is a key-sequenced file composed of a *data part* and an *index part*. The data part of the catalog consists of:

- *Cluster entries* that describe files.

  Cluster entries contain the information that VSE/VSAM requires to properly access a file, verify access authorization (if required), and provide statistics on operations performed on a file.
- *Volume entries* that describe direct-access volumes in terms of the allocation of data spaces and the location of available space.

  Volume entries in a catalog enable VSE/VSAM to keep track of data spaces and free storage areas.

The index part of the catalog allows VSE/VSAM to find the *cluster entry* through its 44-byte name (file-ID), and to find the *volume entry* through the volume serial number.

The information contained in VSE/VSAM catalogs is sufficient to enable VSE/VSAM to suballocate and deallocate space for files on the available volumes. Because these volumes need *not* be mounted on a device of the system, file management is less dependent on job control information, or on information specified in processing programs. In certain cases, however, volumes must be mounted; refer to "Volume Mounting Needs" on page 51.

Except for clusters that have been defined with the UNIQUE attribute, VSE/VSAM can allocate and deallocate space for files on cataloged volumes that are not mounted.

© Copyright IBM Corp. 1979, 2005

Defining VSE/VSAM Data Spaces on a Volume

To define VSE/VSAM data space on a volume, and to identify the volumes that will contain the VSE/VSAM clusters, you use the IDCAMS command DEFINE SPACE.

The space you define will be:
- Identified in the *volume table of contents* (VTOC) of the volume.
- Controlled entirely by the VSE/VSAM catalog in which it is defined.

Note that the volumes that will contain VSE/VSAM files must be mounted.

Defining VSE/VSAM Files

VSE/VSAM files (or clusters) are stored in VSE/VSAM data spaces. Usually, you first define a data space, then you define the files.

All VSE/VSAM files of an installation must be cataloged in a VSE/VSAM catalog. You *catalog* a file by *defining* it through the IDCAMS command DEFINE CLUSTER. IDCAMS, then, enters the name of the file and other characteristics into the catalog.

When you define VSE/VSAM files, you normally do not need any // DLBL and // EXTENT statements. This is because VSE/VSAM automatically allocates space for the files from existing data spaces.

When you define a file with the UNIQUE attribute (to enable the file to be allocated a space of its own), you do not define the data space beforehand. Instead, you provide extent information. You do this through // DLBL and // EXTENT statements in the IDCAMS job stream that defines that file. The data space is then set up at the same time as the entry for the file is created. The volume(s) must be mounted, as in defining a data space.

Note that you can also identify nonVSAM files in a VSE/VSAM catalog, but you cannot suballocate nonVSAM files within VSE/VSAM data space.

About Volumes and VTOCs

Volume Ownership

A given catalog controls (*owns*) any space that is defined in it. This includes the space in which the catalog resides, as well as the VSE/VSAM data space occupied by VSE/VSAM files.

The VSE/VSAM data space occupied by VSE/VSAM files is recorded in the *volume entries* in a catalog. The *ownership* of the volume, and the *use* of VSE/VSAM data space on a volume are indicated by label entries in the VTOC of the volume.

VSE/VSAM volume ownership does not affect nonVSAM files that reside on the volume. NonVSAM files can exist on a volume owned by a catalog but can be cataloged as nonVSAM entries in a catalog that does not own the volume. (NonVSAM files do not have to be defined in a VSE/VSAM catalog.)

Label Entries in the VTOC: The *data secure file bit* in the format-1 VTOC (identifier) label of every VSE/VSAM data space on the volume is set to indicate both read and write protection.

The *ownership bit* in the format-4 VTOC label is set to 1 if the volume contains a VSE/VSAM data space, or if the volume is a candidate volume for a VSE/VSAM object. The ownership bit indicates that the volume is owned by one or more catalogs, but does not identify the volumes.

Volume Entries in a Catalog: Every catalog contains a *volume entry* for every volume it owns. The volume entry describes:
- The characteristics of the direct-access volume
- Every extent of the VSE/VSAM data space
- Every VSE/VSAM object that uses the space of the volume

Note that volumes with duplicate volume serial numbers cannot be owned by the same catalog.

Handling Ownership

Removing Volume Ownership: To remove a volume ownership from a catalog, you must delete all VSE/VSAM objects and data spaces owned by that catalog on the volume.

If you cannot use the DELETE command because IDCAMS can no longer access the volume (due to the damage that resulted from a system or hardware failure), you can reset the ownership bit by using the IKQVDU program.

Note: Do not use IKQVDU if more than one catalog owns space on the volume. This is because IKQVDU resets the ownership bit even if other catalogs own space on the volume. For more information on the IKQVDU program, see "Maintaining VTOC and VOL1 Labels on Disk (IKQVDU)" on page 355.

Releasing Space from Ownership: To release space from ownership by a catalog, you must delete all VSE/VSAM objects that reside in that space. The catalog contains a volume entry, which describes the volume and its VSE/VSAM data spaces.

After deleting the VSE/VSAM objects, you must issue the DELETE SPACE command. The DELETE SPACE command deletes the VSE/VSAM data spaces owned by that catalog, removes the volume entry from the catalog, deletes the format-1 label, and revises the format-4 label in the VTOC (if no other catalogs own space on that volume).

The following fields in the format-4 VTOC label are reset only when all catalogs have released their VSE/VSAM space on the volume:

```
Offset  Length  Description
  77      8     VSE/VSAM time stamp 1 is set to the system's time of day
                when VSE/VSAM acquires volume ownership in a catalog.  This
                time stamp is modified whenever physical space allocated to
                VSE/VSAM is acquired, either by allocation of an extent
                or any time VSE/VSAM physical space is returned to the
                VTOC by VSE/VSAM catalog management routines.

  85      1     VSE/VSAM indicators:
                  Bit 0 set to 1 = One or more VSE/VSAM catalogs owns space
                                   on the volume.
                  Bit 1 set to 1 = No significance for VSE.
                  Bits 2 - 7     = Reserved (set to binary zeros).

  86-87         Not used
```

```
88      8     VSE/VSAM time stamp 2 is the VSE/VSAM-only timestamp.  (Set
              only for MVS compatibility and not used by VSE.)
```

Recognizing VSE/VSAM Data Space Names in the VTOC

VSE/VSAM generates *names* for data spaces and enters the names in the VTOC of the applicable volume. You want to be able to recognize the names that relate to VSE/VSAM when you list the volume's VTOC, when you reinitialize the volume, or when you dump the volume to a magnetic tape.

The VTOC contains the:

- Name of every VSE/VSAM data space on the volume, and
- For unique files, the names for the data and index components of a cluster or alternate index (the format-1 VTOC label is identified with the object's entry name).

The names generated by VSE/VSAM have the following format:

- For a data space containing suballocated VSE/VSAM objects, the VSE/VSAM-generated name is:

  ```
  Z999999n.VSAMDSPC.Taaaaaaa.Tbbbbbbb

  where:
    n=2 if no catalog resides in the data space
    n=4 if a user catalog resides in the data space
    n=6 if the master catalog resides in the data space
    aaaaaaabbbbbbb is the time stamp value
  ```

- For a unique data space (defined as a data space that cannot contain more than one cataloged VSE/VSAM object), the VSE/VSAM-generated name is:

  ```
  VSAMDSET.DFDyyddd.Taaaaaaa.Tbbbbbbb

  where:
    yyddd is the date (year and Julian day)
    aaaaaaabbbbbbb is the time stamp value
  ```

Relating Names Created for Unique Data Spaces

When you define a VSE/VSAM file with the UNIQUE attribute, VSE/VSAM creates a unique data space. If you specify a name for the data and/or index component, VSE/VSAM places the name you specify in the format-1 VTOC label rather than generating a name.

To relate the VSE/VSAM-generated name with a VSE/VSAM cluster, alternate index, catalog, or data space, you have to list the catalog that owns the volume. Issue a LISTCAT command to list the content of the catalog. The LISTCAT output, then, relates the VSE/VSAM-generated names with user-assigned entry names for cataloged objects.

Time Stamps

Every volume owned by a catalog contains a time stamp that is written in the VTOC when the volume is first cataloged. Both time stamps, the one in the VTOC and the one in the volume entry in the catalog, are updated whenever the catalog is updated in response to the following IDCAMS commands:

DEFINE SPACE
DEFINE CLUSTER (with UNIQUE attribute)
DEFINE ALTERNATEINDEX (with UNIQUE attribute)
DEFINE MASTERCATALOG
DEFINE USERCATALOG
DELETE SPACE

DELETE CLUSTER (with UNIQUE attribute)
DELETE ALTERNATEINDEX (with UNIQUE attribute)
DELETE MASTERCATALOG
DELETE USERCATALOG
EXPORT PERMANENT a cluster or alternate index with the UNIQUE attribute. (The cluster or alternate index is deleted.)
IMPORT a cluster or alternate index with the UNIQUE attribute. (Any old copy, if present, is deleted, and a new version is defined.)

If the time stamp of the volume is earlier than the time stamp of the catalog, the volume is considered down-level. IDCAMS will not open a file on a down-level volume.

Volume Mounting Needs

Volumes *must* be mounted in the following cases:

- If it is the owning VSE/VSAM catalog.
- If a volume contains a *unique* file.
- If there is not enough unused data space to contain a file, you must mount one or more volumes to allocate new data space, or you have to assign to the file other volumes that contain unused data space.
- In all cases where files are actually accessed (for example, VSE/VSAM application programs, PRINT, REPRO, DELETE ERASE, work files, EXPORT, IMPORT), you have to mount the volumes.
- In all cases where a VTOC update is necessary (for example, DEFINE or DELETE SPACE, DEFINE or DELETE a UNIQUE file, ALTER NEWNAME NONVSAM, ALTER NEWNAME UNIQUE), you have to mount the volume(s) for the affected VTOC(s).

Work Files on Virtual Disk

Work files may reside on real disk devices, but also on *virtual disks*, that is: in virtual storage that has been reserved for z/VSE data spaces. If a work file resides on virtual disk, data is moved to or from data space (instead of being written to or from a real disk device).

Virtual disk processing should only be used in conjunction with temporary work files, because the information in a z/VSE data space is lost whenever the system is restarted.

Preparations for Use

To prepare for the use of virtual disk, in general proceed as follows:

1. At IPL time, add one or more virtual disks by using the “ADD” command, FBAV operand.
2. Define z/VSE data space by using the “SYSDEF” command.
3. After IPL, define the layout of the virtual disk(s) by using the “VDISK” command.

   (Using the VDISK command makes the virtual disk available automatically.)

For information on the commands, refer to the manual *z/VSE System Control Statements*.

To use the virtual disk support for VSE/VSAM:

1. Define one or more user catalogs on the prepared virtual disks by using the IDCAMS command DEFINE USERCATALOG.

   A catalog can own VSE/VSAM space on one or more virtual disks (up to 123 volumes).
2. Catalog the VSE/VSAM objects in the defined catalog by using the DEFINE commands of IDCAMS.

For an example of definitions, refer to the manual *VSE/VSAM Commands* under "Defining Work Files on Virtual Disk."

Restrictions

Files residing on virtual disk are managed by VSE/VSAM as if they resided on real devices. However, note the following restrictions:

- Master catalogs must *not* reside on virtual disk.
- If the user catalog resides on a:
  - Virtual disk, and if defining an object on a real disk, the define will fail.
  - Real device, and if defining an object on a virtual disk, the define will fail.

  That is, cataloging an object to a user catalog is only successful if both - object and catalog - are either on real volumes or virtual volumes.
- Whenever virtual disks are lost for VSE/VSAM (for example, on re-IPL or detach), you have to EXPORT DISCONNECT the corresponding user catalog(s) before defining a new user catalog of the same volume serial name.

Transporting Files between Systems

Transporting Catalog Information

Because all VSE/VSAM files must be cataloged, moving a file from one system (or set of systems if in a disk sharing environment) to another requires that catalog information be moved along with it or that the copy of the file moved be cataloged in the receiving system. If the catalog information is copied with the file, it must be in a format that both systems can process.

Transporting Files between VSE/VSAM and DFSMSdfp VSAM or DFSMS/MVS

Use EXPORT and IMPORT to copy VSE/VSAM files and their catalog information to DFSMSdfp VSAM (which uses the Integrated Catalog Facility - ICF). The only way you can move MVS files cataloged with the new catalog format to VSE/VSAM is to export those files to tape while running on DFSMSdfp VSAM. Then you can import the tape files to VSE/VSAM. You cannot mount a DFSMSdfp VSAM format volume on a z/VSE system. VSE/VSAM cannot process DFSMSdfp ICF catalog information.

Do not use BACKUP and RESTORE to transport files from z/VSE to MVS, because MVS does not have BACKUP and RESTORE commands.

Transporting Files between VSE/VSAM and MVS/VSAM (not DFP)

Use EXPORT and IMPORT to transfer files and their catalog information between systems. Files and volumes are portable between VSE/VSAM and MVS/VSAM "old" catalog format systems.

You can use BACKUP and RESTORE to back up MVS "old" catalog format files on z/VSE and restore them on z/VSE. This procedure is recommended only for a one-time move of files from MVS to z/VSE. The only way you could move the files back to MVS is to use EXPORT/IMPORT, because MVS does not support BACKUP and RESTORE.

The description under "Transporting Files between z/VSE Systems" also applies to transferring files between z/VSE and MVS "old" catalog format systems.

Transporting Files between z/VSE Systems

You can move individual files and user catalogs from one z/VSE system to another by using the EXPORT and IMPORT commands. When you move a user catalog from one system (or set of systems) to another, its VSE/VSAM volume ownership moves along with it. Thus, a VSE/VSAM volume (without compressed data sets) is portable between systems together with all VSE/VSAM data spaces and files contained on the volume(s). Any VSE/VSAM volume including compressed data sets is portable between z/VSE systems.

The entire VSE/VSAM master catalog and the VSE/VSAM volumes owned by the master catalog can be moved from one z/VSE system (or set of systems) to another.

To use a VSE/VSAM master catalog from another system, you need only assign it by use of the DEF SYSCAT=cuu command during initial program load. All VSE/VSAM volumes owned by that catalog are then available to the receiving system. In addition, a // DLBL statement for the master catalog must be provided either in the job stream or in the label area.

Catalog and File Migration

The following explains how to proceed if you want to migrate VSE/VSAM catalogs and files from one device type to another. This includes movement from one CKD device type to another, one FBA device type to another, or from a CKD device to an FBA device (and vice versa). Note that a catalog on an FBA device can own CKD volumes (and their VSE/VSAM files), and a CKD catalog can own FBA volumes.

There are two ways to migrate objects and their catalog information from one device type to another. The simpler method is to use the VSE/VSAM Backup/Restore Function, but there are some restrictions; the other way is to use a combination of EXPORT/IMPORT and DEFINE commands. For a description of the methods, refer to "Migrating Catalogs" on page 55.

Definitions for Catalog Migration

Defining a Catalog

The following applies to both master and user catalogs. Whenever you use BACKUP/RESTORE or EXPORT/IMPORT for migration, you must first define the catalog that will own the VSE/VSAM objects after migration.

VSE/VSAM defines a VSE/VSAM data space from which the catalog (and catalog recovery area) is suballocated. This is done on CKD devices using the DEDICATE, ORIGIN, CYLINDERS, TRACKS, or RECORDS subparameter of DEFINE MASTERCATALOG|USERCATALOG. FBA devices require the same process,

except that the DEDICATE, ORIGIN, BLOCKS or RECORDS subparameter must be specified. (CYLINDERS or TRACKS is not accepted.) Therefore, you must convert a CYLINDERS or TRACKS value to a BLOCKS or RECORDS quantity.

If you specify DEDICATE for the CKD device, no conversion is necessary.

Convert the number of tracks or cylinders into the number of bytes, using LISTCAT to determine the number of bytes per track and tracks per cylinder. Divide the number of bytes by 512 to determine the BLOCKS value. Adjust it accordingly if you want more or less space allocated.

The beginning-block-number specification in the ORIGIN parameter depends on where you want the data space to be on the volume (VSE/VSAM always rounds it to the next minimum CA boundary). Use the LVTOC utility program to determine what space is available on the volume. The catalog will be located at the beginning of the defined data space. Ín *VSE/VSAM Commands*, refer to

- the description of the NAME|VOLUME... parameter under "LISTCAT"
- the description for special fields BLKS/MAX-CA under "Volume Entry, Special Fields for (V)" to the description for special fields .

You may wish to change other subparameters of MCAT or UCAT (for instance, the volume serial number), but there are no special considerations for FBA devices.

Specify the actual space to be suballocated for your catalog using the BLOCKS or RECORDS subparameters of DATA and INDEX. Do *not* try to directly convert a CKD catalog size definition to a fixed block definition. Instead, calculate the desired values; refer to the instructions in the manual *VSE/VSAM Commands* under "Defining a Catalog." To avoid an overly small, inefficient CA size, make the secondary allocation value at least as large as the desired CA size.

Defining a VSE/VSAM Data Space

The considerations for data space definition are essentially the same as for catalog definition. Differences are:

- A catalog is not suballocated from the data space.
- Both BACKUP/RESTORE and EXPORT/IMPORT assume that you have already defined a VSE/VSAM data space on the new volume.

If CANDIDATE is specified with DEFINE SPACE, fixed block data space definition is the same as CKD data space definition.

Defining a Non-Unique Cluster or Alternate Index

Because these files (or their components) are suballocated from VSE/VSAM data spaces, there are no job control considerations for FBA devices. For FBA devices, you must convert the TRACKS or CYLINDERS subparameters to BLOCKS or RECORDS. (The RECORDS subparameter does not require conversion.) This conversion is the same as described above for catalog conversion.

Defining a Unique Cluster or Alternate Index

If a cluster or alternate index contains both a non-unique component and a unique component, conversion considerations for the non-unique component are as described above.

For every unique component (data and, if present, index), you must convert EXTENT statement parameters and the TRACKS|CYLINDERS subparameters. Both conversions are required because a unique component occupies its own

VSE/VSAM data space. If the component is to be on more than one volume, specify a new EXTENT statement for every volume.

Migrating Catalogs

Catalog Migration Using BACKUP/RESTORE

You cannot actually back up and restore catalogs under BACKUP/RESTORE, but when you back up and restore objects (including empty objects), their catalog information is backed up and restored too. This makes it possible for you to use BACKUP and RESTORE to copy objects and their catalog information into a different catalog. If the new catalog already contains an entry name for the object restored, the original object is deleted, and the restored object is added to the new catalog.

Catalog Migration Using EXPORT/IMPORT

Moving a Master Catalog to Another Volume:

1. Using EXPORT, create portable copies of all files that are to be in the new catalog (procedure described below). For EXPORT, DISCONNECT any user catalogs to be reconnected to the new catalog.
2. IPL with the master catalog assigned to the new volume, using the IPL DEF SYSCAT=cuu command.
3. Define the new master catalog (procedure described above).
4. Define any VSE/VSAM data spaces required for the volumes. (You need not delete files and catalogs belonging to another catalog.) Note that the define catalog operation has already defined a data space on the catalog volume. Any space to be occupied by unique files should be left unallocated.
5. Using IMPORT, copy VSE/VSAM files to volumes belonging to the new catalog. (For considerations on moving to a different device type, refer to "Migrating VSE/VSAM Files to Another Device.") If IMPORT was used, you can IMPORT CONNECT user catalogs.

Moving a User Catalog to Another Volume:

1. Using EXPORT, create portable copies of all files that are to be in the new catalog (procedure described below).
2. Delete or disconnect the previous user catalog entry unless it is owned by a different master catalog.
3. Define the new user catalog (procedure described above).
4. Define any VSE/VSAM data spaces required for the volumes. (You need not delete files and catalogs belonging to another catalog.) Note that the define catalog operation has already defined a data space on the catalog volume. Any space to be occupied by unique files should be left unallocated.
5. Using IMPORT, copy VSE/VSAM files to volumes belonging to the new catalog. (For considerations on moving to a different device type, refer to "Migrating VSE/VSAM Files to Another Device.")

Migrating VSE/VSAM Files to Another Device

File Migration Using BACKUP and RESTORE

Note: VSAM will tolerate the use of IDCAMS BACKUP/RESTORE for migration from a non-SCSI device to a SCSI device, or vice versa. However, not every cluster can be migrated in this manner. In these cases, IDCAMS

EXPORT/IMPORT must be used instead. EXPORT/IMPORT is the recommended method of data migration.

The VSE/VSAM Backup/Restore Function can back up the following objects and their catalog information onto tape or disk volumes:
- Key-sequenced data sets (KSDS)
- Entry-sequenced data sets (ESDS)
- Relative-record data sets (RRDS)
- Variable-length relative record data sets (VRDS)
- Alternate indexes (AIX)
- SAM ESDS files in CI format

Empty objects can be backed up and restored. An empty object is an object that was defined using the NOALLOCATION parameter, an object that has never been loaded with data, or an object that has not been loaded since reset. Although they cannot be specified in the command, paths are backed up and restored automatically when their respective path entry clusters are backed up or restored.

VSE/VSAM Backup/Restore supports backing up onto tape and disk, and restoring from tape and disk; this support applies to all tape and disk devices that are supported by z/VSE.

You can back up and restore multiple objects with a single command. If you specify BACKUP (*), all objects defined in a specific catalog will be backed up. If you specify RESTORE (*), all objects residing in a specific backup file are restored. You can also specify generic names representing groups of related objects to be backed up or restored. Because the generic specification may include objects you do not want backed up or restored, you can exclude objects by specifying either their entry names or other generic names. For information on the use of generic names for back up and restore, refer to the manual *VSE/VSAM Commands* under:
- "Using BACKUP and RESTORE"
- "Generic Names"

To copy objects from one volume to another volume of a different device type, specify the *volser* representing the new volume in the RESTORE command.

Other Methods of File Migration

In addition to Backup/Restore, there are three ways to move VSE/VSAM files from one volume to another. They may or may not require moving from one catalog to another.

DEFINE/REPRO:

- To move files between two volumes owned by different catalogs, DEFINE every file on the new volume, using its old name. REPRO every file onto the new volume, and delete it from the old one.
- To move files between two volumes owned by the same catalog, DEFINE every file on the new volume, using a temporary name that is not already in the catalog. REPRO every file onto the new volume, and delete it from the old volume. Using ALTER, rename the new copy with the name the file had on the old volume.

In both cases, alternate indexes can be copied. You must redefine all paths for the new copy.

EXPORT/IMPORT: With EXPORT/IMPORT, every file to be migrated is first exported to a temporary SAM file (tape or disk). For EXPORT PERMANENT, this frees the space (and volumes if all files on them are exported) that is potentially reusable during the IMPORT phase.

To ensure the desired space allocation, DEFINE the files importing them. If files are imported but not defined, too much or too little space may be allocated to them. Then IMPORT the files.

Unique files require extent values specified on an EXTENT statement. Path definitions are implicitly transferred.

NonVSAM Migration

Catalog entries can be moved also into catalogs on FBA devices (as described above) through DEFINE NONVSAM and DELETE, but they cannot have fixed block specified as their device type.

Space Allocation through Modeling

If a user catalog, cluster, or alternate index migrated from one device type to another had its space allocation defined by modeling, you should consider changing to explicit specification, or modeling it on a catalog, cluster, or alternate index on the new device type. Otherwise, you will allocate space based on the track and/or cylinder capacity of the old device type rather than the new device type. This can cause wasted space, excessive secondary allocation, and inefficient or even invalid CA or CI sizes.

For further information about modeling, see "Using an Object as a Model."

Using an Object as a Model

You can use the entry of an already-defined alternate index, catalog, cluster, or path as a *model* for the definition of another object of the same type. When one entry is used as a model for another, its attributes are copied as the new entry is defined.

Modeling permits you to set your own parameter defaults to override system defaults. Once defaults are established, you need not specify them every time you define new objects. An explicit parameter specification, however, overrides defaults established by you (through modeling) and by the system.

The normal IDCAMS DEFINE CLUSTER or DEFINE ALTERNATEINDEX procedure is greatly simplified by reducing the numbers of parameters required. This in turn can reduce the number of errors that are likely to occur, and the number of parameters to which the user needs to be exposed. At the same time, it permits application- and installation-associated standards.

There are three kinds of models; they are referred to as:
- Explicit Allocation (Example in Figure 19 on page 58)
- Explicit NOALLOCATION (Example in Figure 21 on page 61)
- Implicit NOALLOCATION (Example in Figure 22 on page 62)

With *explicit* modeling, you have to specify the name of the model you wish to use. With *implicit* modeling, VSE/VSAM chooses a default model based on the kind of object you are trying to define.

About the MODEL Subparameter

You can specify the MODEL parameter in the DEFINE commands ALTERNATEINDEX, CLUSTER, PATH, and USERCATALOG.

Using the MODEL parameter, you can easily define files that are identical, except for their names and security attributes. When you use the MODEL parameter, ensure that your job is not terminated because of allocation problems when you explicitly do any of the following:

- Specify a different type of device with the VOLUMES parameter.
- Change the length or position of the keys with the KEYS parameter.
- Change the size of records, buffer space, or CIs with the RECORDSIZE, BUFFERSPACE, or CONTROLINTERVALSIZE parameters.
- Change the type of cluster (that is, entry-sequenced, key-sequenced, or relative-record), the type of alternate index (that is, key-pointer or RBA-pointer), or the allocation-type of the object (that is, unique or non-unique).
- Change the *unit of allocation* with the BLOCKS, TRACKS, CYLINDERS, or RECORDS parameters.

When you explicitly specify any of the above parameters for your to-be-defined object, you might have to make corresponding changes to other related parameters.

Explicit Allocation Models

Figure 19 shows an explicit model that occupies data space and can be used as a normal VSE/VSAM object. You must explicitly specify the *entryname* subparameter of the MODEL parameter to identify the object to be used as a model. This is the only form of modeling that is valid for paths and user catalogs. If MODEL is specified as a parameter of PATH:

- The attributes of the model are used for the path defined.
- Any attributes explicitly specified as parameters of the defined path are defined and override those of the model.

Figure 19. Explicit Allocation Model

Figure 20 on page 60 shows how parameters are merged from a *Model Cluster* and an *Explicit Specification* into *New Cluster*. Once the merge is completed, *New Cluster* contains a new list of cluster parameters which VSE/VSAM uses to create a cluster.

In the figure, it is assumed that MODEL is specified at the *cluster level* in the DEFINE CLUSTER command. It is also assumed that MODEL is *not* specified at the *data component level* and *index component level* of the command.

The following explains the step numbers shown in the figure:

(1) The non-propagating cluster level attributes (entryname, passwords, AUTHORIZATION, ATTEMPTS, CODE, OWNER, TO, FOR, and allocation attributes) of the model are used for the defined user catalog, cluster, or alternate index.
(2) Any non-propagating cluster level attributes *explicitly specified* as parameters of the defined object are applied to and override those of the model.
(3) The attributes of the model are used for the data and index components of the alternate index, cluster, or user catalog.
(4) Attributes explicitly specified at the cluster level are propagated to the data and index components of the cluster.
(5) Attributes that are explicitly specified for the data and index components of the object (that is, specified as subparameters of the DATA or INDEX parameter) are defined.

Note that attributes specified for every step (n) override the attributes specified by the previous step.

If MODEL is Specified in DATA or INDEX Parameter

(not applicable to a user catalog):

- Attributes explicitly specified at the cluster level are propagated to the data and index components of the object.
- Attributes of the model specified for the data or index component are defined (that is, the model specified with the MODEL subparameter of the DATA or INDEX parameter).
- Attributes explicitly specified for the data and index components are defined (that is, the attributes specified with subparameters of the DATA or INDEX parameter).

Attributes specified for every step override the attributes specified by the previous step.

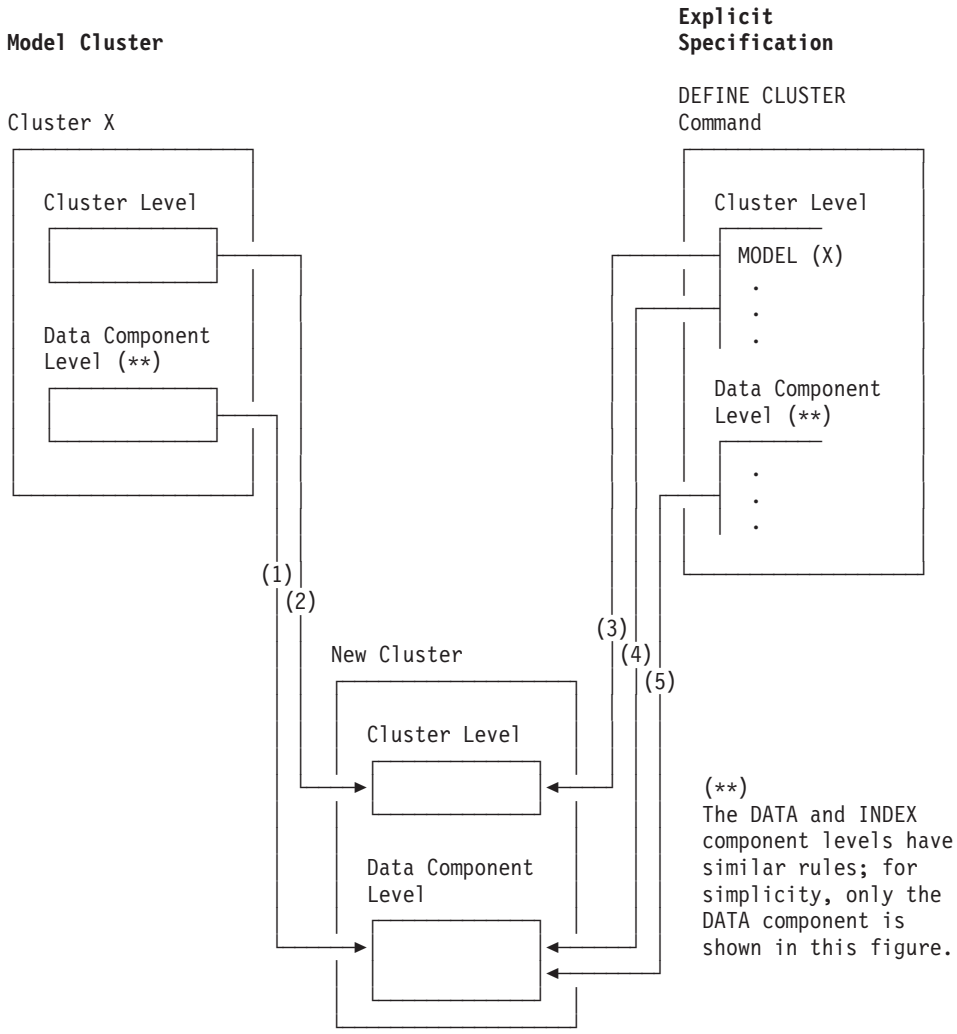


Figure 20. Specifying the MODEL Parameter at the CLUSTER Level Only

Explicit Noallocation Models

Using explicit noallocation and default models, the defined object exists *only* as a model; no space is suballocated to it. The model is represented by entries in the VSE/VSAM catalog.

Figure 21 on page 61 is an *explicit* model because you must specify MODEL(entryname) for the cluster you wish to use as a model. It is a *NOALLOCATION* model because no storage is allocated to it.

Figure 21. Explicit NOALLOCATION Model

Implicit NOALLOCATION Models (Default Models)

In the case of an *implicit* model, you do not have to specify the name of the model in order to reference it. It is a *NOALLOCATION* model because no storage is suballocated to it.

The *implicit model* is a default model.

When you define the model, specify the entryname subparameter of the NAME parameter as one of the following:

DEFAULT.MODEL.KSDS
(key-sequenced file)

DEFAULT.MODEL.ESDS
(VSAM entry-sequenced file)

DEFAULT.MODEL.ESDS.SAM
(managed-SAM file)

DEFAULT.MODEL.RRDS
(relative-record file)

DEFAULT.MODEL.VRDS
(variable-length relative record file)

DEFAULT.MODEL.AIX
(alternate index)

Every catalog may have six implicit models, one of every type.

As shown in Figure 22 on page 62, you need only specify INDEXED, NONINDEXED, RECORDFORMAT(...)NONINDEXED, NUMBERED, or AIX for VSE/VSAM to locate the appropriate model.

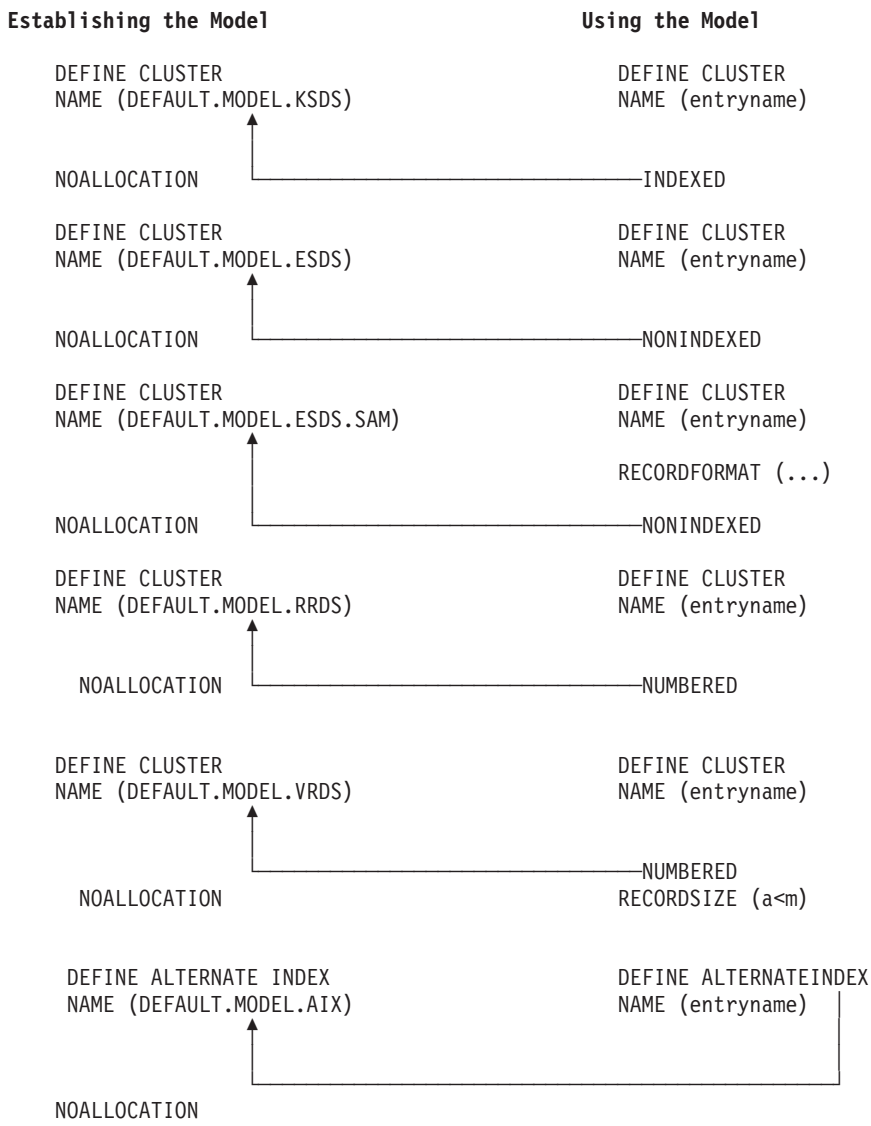


Figure 22. Implicit NOALLOCATION Models

How VSE/VSAM Determines Which Parameters to Use

VSE/VSAM goes through the following sequence in determining which parameter to use in the definition of a cluster or alternate index.

1. Did you explicitly specify a parameter in the define? If yes, VSE/VSAM uses it. (If you explicitly specify a space allocation parameter (CYLINDERS, TRACKS, BLOCKS, or RECORDS) at *any* level of DEFINE CLUSTER/AIX, the space allocation parameter(s) in your model are ignored.)
2. Did you specify MODEL parameter in the define (refer to Figure 19 on page 58)? If yes, go to step 4, below; VSE/VSAM creates a file using the parameters specified in MODEL(entryname).
3. Did you specify the NOALLOCATION parameter with a DEFAULT.MODEL.xxxx in a previous DEFINE command, thereby creating a default model (refer to Figure 22)? If yes and the file organization matches the entryname, VSE/VSAM uses the parameters specified in the default model.
4. If none of the above apply, VSE/VSAM uses the system default (if one exists).

Restrictions

The following restrictions exist for modeling of VSE/VSAM objects.

- If you specify DEFINE CLUSTER or DEFINE ALTERNATEINDEX and the cluster name begins with DEFAULT.MODEL., VSE/VSAM assumes that you are establishing a model. The rest of the name must be KSDS, ESDS, ESDS.SAM, RRDS, VRDS or AIX. It is not possible to open a file or component whose name begins with DEFAULT.MODEL.. DEFINE CLUSTER and DEFINE ALTERNATEINDEX ignores user-specified DATA and INDEX component names for clusters that have the DEFAULT.MODEL. prefix. Instead, these components are implicitly assigned a name constructed from the cluster or alternate index name with the additional qualifier of DATA or INDEX. A message will tell you any data/index names that have been generated in this way.
- If space parameters (CYLINDERS, TRACKS, RECORDS, or BLOCKS) are specified at any level of DEFINE CLUSTER or DEFINE ALTERNATEINDEX, they override any modeled defaults.
- You can model the USECLASS parameter only if one of the following is true:
  - If space parameters (CYL, TRK, REC, BLK) are *not* specified
  - If space parameters are specified at a different level

  You cannot model USECLASS if both specifications, USECLASS and space parameter, are at the same level (that is, both specifications are at cluster, data, or index level). If you do specify the same level, VSE/VSAM cannot model from a default model.

  However, you can, for example, model the USECLASS at cluster level if space parameters are specified at the data or the index level.
- You cannot rename (through ALTER NEWNAME or IMPORT NEWNAME) any catalog entry such that the name is changed to or from DEFAULT.MODEL.xxxx. An attempt to do so causes the command to terminate with an error message.
- When a file is defined *implicitly* (through managed-SAM access) and if you have not provided volume information in an EXTENT statement, VSE/VSAM attempts to construct a volume list of up to 16 volumes from the DEFAULTVOLUMES parameter in a managed-SAM ESDS default model (DEFAULT.MODEL.ESDS.SAM). No other parameters from the SAM ESDS default model are used for an implicit define.
- When a VRDS has to be defined through a default model, and to indicate a VRDS, you have to define the recordsize (a<m). If recordsize is not defined, an RRDS is assumed.
- The use of this utility in a VSE/VSAM environment requires special considerations, because both the volume VTOC and the catalog contain space mapping information about the volume which has to be synchronized to insure accessibility and to avoid damage to data.

Table 3 lists the various DEFINE parameters and shows for each one if it can be modeled explicitly with 'MODEL(entryname)' and implicitly with 'DEFAULT.MODEL.xxxx'.

Table 3. Modeling of DEFINE Parameters

Parameter	Modeling		System Default if Parameter Not Modeled	Notes
	Explicit	Implicit		
ATTEMPTS	Yes	Yes	Yes	Not propagated to other levels.
AUTHORIZATION	Yes	Yes*	No	Not propagated to other levels.

Table 3. Modeling of DEFINE Parameters (continued)

Parameter	Modeling		System Default if Parameter Not Modeled	Notes
	Explicit	Implicit		
BLOCKS	Can model only if not explicitly specified at any level.		No	Propagated via algorithm from cluster or data levels.
BUFFERSPACE	Yes	No	Yes	
CLASS	No	No	Yes	See USECLASS Parameter.
CODE	Yes	Yes*	No	Not propagated to other levels.
COMPRESSED	n/a	n/a	No	Must not be used with a model data set.
CONTROL INTERVALSIZE	Yes	No	Yes	
CYLINDERS	Can model only if not explicitly specified at any level.		No	Propagated via algorithm from cluster or data levels.
DEDICATE	No	No	No	
DEFAULTVOLUMES	No	No	Yes	
ERASE	Yes	Yes	Yes	Propagated to data level only; NOERASE is the default.
EXCEPTIONEXIT	Yes	Yes	No	
FILE	No	No	No	
FOR	Yes	Yes	Yes	Specified at cluster level only; propagated to data or index.
FREESPACE	Yes	Yes	Yes	(0 0) is the default.
INDEXED	See last column.	No	Yes	KSDS is created if nothing or INDEXED is specified.
KEYRANGES	Yes	Yes	No	
KEYS(AIX)	Yes	Yes	Yes	
KEYS (cluster)	Yes	Yes	Yes	Not specified or modeled for INDEX.
NOALLOCATION	Yes	No	Yes	SUBALLOCATION is the default.
NOERASE	Yes	Yes	Yes	NOERASE is the default.
NONINDEXED	See last column.	No	Yes	ESDS or SAM ESDS.
NONSPANNED	Yes	Yes	Yes	NONSPANNED is the default.
NONUNIQUEKEY	Yes	Yes	Yes	NONUNIQUEKEY is the default.
NOREUSE	Yes	Yes	Yes	NOREUSE is the default.
NOUPGRADE	Yes	Yes	Yes	UPGRADE is the default.
NOWRITECHECK	Yes	Yes	Yes	NOWRITECHECK is the default.
NUMBERED	See last column.	No	Yes	RRDS is created if recordsize (a=m), VRDS is created if recordsize (a<m).
ORDERED	Yes	Yes	Yes	UNORDERED is the default.
OWNER	Yes	Yes	No	Not propagated to other levels.

Table 3. Modeling of DEFINE Parameters (continued)

Parameter	Modeling		System Default if Parameter Not Modeled	Notes
	Explicit	Implicit		
Passwords	Yes	No	No	No propagation from cluster level, but lower level password is propagated to master if no master password is specified.
RECORDFORMAT	Yes	n/a	Yes	For SAM ESDS models only.
RECORDS	Can model only if not explicitly specified at any level.		No.	Propagated via algorithm from cluster or data levels.
RECORDSIZE	Yes	No	Yes	
RECOVERY	Yes	Yes	Yes	
RELATE	No	No	n/a	
REUSE	Yes	Yes	Yes	NOREUSE is the default.
SHAREOPTIONS	Yes	Yes	Yes	
SPANNED	Yes	Yes	Yes	NONSPANNED is the default.
SPEED	Yes	Yes	Yes	
SUBALLOCATION	Yes	No.	Yes	SUBALLOCATION is the default.
TO	Yes	Yes	Yes	Specified at cluster level only; propagated to data and index.
TRACKS	Can model only if not explicitly specified at any level.		No	Propagated via algorithm from cluster or data levels.
UNIQUE	Yes	No	Yes	SUBALLOCATION is the default.
UNIQUEKEY	Yes	Yes	Yes	NONUNIQUEKEY is the default.
UNORDERED	Yes	Yes	Yes	UNORDERED is the default.
UPGRADE	Yes	Yes	Yes	UPGRADE is the default.
USECLASS	Only if space parms are specified at different level, or if space parms are not specified.		Yes	
VOLUMES	Yes	Yes	No	
WRITECHECK	Yes	Yes	Yes	NOWRITECHECK is the default.
(*) To implicitly model this parameter, the object must be defined with at least one password, and the master catalog password must be specified in the CATALOG parameter.				

Default Volumes

Default volume lists are derived from the volumes list of a default model that is of the same type as the object defined. For example, if a VSE/VSAM ESDS cluster is defined without a VOLUMES parameter, an ESDS default model (DEFAULT.MODEL.ESDS.DATA) is used to build the volumes list for the ESDS. Because volume selection from the default volume list is done randomly for every component, the data and index components of a KSDS or AIX could reside on

different volumes and even different device types. You can eliminate the possibility of different device types by including devices of only one type when defining the KSDS or AIX model.

When a file is defined implicitly (through managed-SAM) and if you have not provided volume information in an EXTENT statement, VSE/VSAM attempts to construct a volumes list of up to 16 volumes from a managed-SAM ESDS default model (DEFAULT.MODEL.ESDS.SAM). No other information is used (from the SAM ESDS default model) for an implicit define.

DEFAULTVOLUMES forces a default model to override an explicit model for purposes of determining the volumes list. There are three sources of volumes lists:

1. Explicit specification (VOLUMES parameter)
2. Explicit model (MODEL parameter)
3. Default model (DEFAULT.MODEL.xxxx plus VOLUMES parameter)

These sources are listed in order of precedence. 1 overrides 2, and 3 takes effect if 1 and 2 are missing. If only 2 and 3 are present, however, specifying DEFAULTVOLUMES causes the volumes list in 2 to be bypassed in favor of the volumes list in 3. You cannot specify the DEFAULTVOLUMES parameter to bypass 2 if 3 does not exist. (At least one of these options (1, 2, or 3) must be specified or modeled.)

DEFAULTVOLUMES cannot be explicitly modeled because it is not retained as an attribute in the catalog. Do not try to use default volume lists with KEYRANGES, because VSE/VSAM does not order the volumes in any way when allocating space to them.

Chapter 5. Working With Compressed Files

This Chapter ...

- Introduces you to data compression.
- Describes how it works internally.
- Explains what you need to do to work with compressed data.
- Differentiates the files for which data compression works and under which circumstances.
- Describes the IKQCPRED tool, which examines VSE/VSAM data for its suitability and calculates how well it would compress.

Introduction to VSE/VSAM Compression

If a cluster is defined with the COMPRESSED attribute, VSE/VSAM attempts to minimize the external storage needs by compressing each record written to the file. The compression algorithm is compatible with the zSeries and ESA/390 hardware compression facility, that is if the processor supports the CMPSC instruction, then this hardware instruction is used to compress or expand data. The ESA/390 compression facility is further described in *Enterprise Systems Architecture/390: Data Compression* SA22-7208. On processors without the ESA/390 compression facility, an equivalent software emulation is performed. See also the *Principles of Operation* manual for your processor.

The compression facility compresses and expands data using a *dictionary*. This dictionary contains the information which substrings of the data are to be encoded and how to expand the encoded strings. When data is loaded into a compressed cluster, VSE/VSAM attempts to build a dictionary that can compress that data. As soon as this dictionary is built, all records written to the file are compressed using this dictionary, and all records read from the file are expanded using this dictionary. The information on what the dictionary looks like is stored in the VSE/VSAM *Compression Control Data Set (CCDS)*, which needs to exist in each catalog that holds compressed data sets.

Advantages

Working with compressed files has several advantages. The reduction in DASD space is the most obvious one, but not the only one:

- Since records in a compressed file are smaller, the resulting relative byte address (RBA) is smaller. While the maximum size of a VSE/VSAM data set (excluding extended-addressed KSDS data sets) is still 4GB (x'FFFFFFFF'), more user data can be stored within a single data set.
- More data can be stored per control interval or buffer. The advantages are:
  - For sequential workloads, a new buffer is required less often. This reduces the number of I/O requests.
  - For random access workloads, the control interval size might be decreased, which in turn might speed up the data transfer time.

Activating VSE/VSAM Data Compression

This involves the following steps:

© Copyright IBM Corp. 1979, 2005

1. Identify which data sets are eligible for compression and which ones you want to have compressed.
2. Define the Compression Control Data Set (CCDS)

   While the CCDS is *required* for those catalogs that contain compressed clusters, it is recommended to have a CCDS defined for each catalog.

   Catalogs that were newly defined using the VSE/VSAM Interactive Interface already have a CCDS defined for them. Otherwise, see "How to Define the Compression Control Data Set" on page 70 on how to define a CCDS or use the skeleton SKVSAMDC in the ICCF library 59.
3. Define the cluster with the COMPRESSED attribute.
4. Load data into the file.

   It is necessary to use *load mode* since VSE/VSAM can determine the dictionary only during the initial load mode. You could use, for example, IDCAMS REPRO to load the data from an existing file to DASD or tape.

How VSE/VSAM Data Compression Works Internally

As mentioned previously, the ESA/390 compression facility, as well as its software emulation, requires a *dictionary* to compress and expand data. The dictionary is the key to an effective compression.

Dictionary Creation

Unfortunately there is no dictionary that is able to compress all kinds of data effectively. The dictionary is data-dependant and needs to be constructed by the actual data, or, to be more precise, by a subset of data that should be representative for the total data. On the other hand, the vast majority of data would actually consists of certain elements that are likely to re-occur. Thinking of an English text, such elements could be English suffices (-ion, -ing) or character sequences such as a comma followed by a space. Of course there are many and not just text-related elements that are likely to re-occur, and hence might be good candidates for a compression. For each of these elements it is known how to compress them, this information is contained in a data structure called *dictionary building block (DBB)*. Each DBB can be viewed as a small, very specialized dictionary. VSE/VSAM has several hundred of them.

VSE/VSAM determines the dictionary for each compressed cluster by examining the first set of data loaded into the cluster. This examination consists of two phases:

1. Interrogation: In this phase VSE/VSAM examines the data written to the file and attempts to find out which elements make up the data, that is it identifies a number of DBBs that might be fit to compress the data.
2. Sampling: Eventually the DBBs selected in the interrogation phase are used to compress the encountered data. Those that perform best are selected and a real dictionary is then assembled from all the selected DBBs.

If the interrogation and sampling phase successfully ends with the creation of a dictionary, then all records written subsequently to the file are compressed with this dictionary. The dictionary remains associated ('mated') with the cluster for the lifetime of the cluster.

The information about which DBBs make up the dictionary is stored in the CCDS. Each record in the compression control data set identifies one compressed cluster and holds information about the compression state of the cluster, and usually which DBBs constitute the dictionary for the cluster.

Compression States

A compressed cluster always assumes one of four possible compression states, as outlined in Figure 23. These states are reported in the LISTCAT output.

States Explanation

CMPPENDING
If a compressed cluster is newly defined by the IDCAMS command DEFINE CLUSTER **1**, nothing is known about the data that will be loaded into it. When records are loaded into the file, VSE/VSAM interrogates the data in order to create a dictionary for it.

CMP-ACTIVE
If VSE/VSAM has successfully determined how to compress the data **2**, the compression state changes to ACTIVE. From now on all records written to the file are compressed.

CMP-REJECT
If VSE/VSAM cannot compress the data, the compression state is changed to REJECTED **3**. You can access a cluster in this state just like any other compressed cluster; the only difference is that the records are not compressed. Possible reasons for rejection are:
- The data is already in some compressed format.
- You closed the file before the interrogation phase completed, that is you did not write enough data during the initial load mode.

CMP-UNDET
If the information how to compress and expand a cluster is lost **4**, then the compression state is undeterminable. In this case the compression control data set might be deleted or become inaccessible.

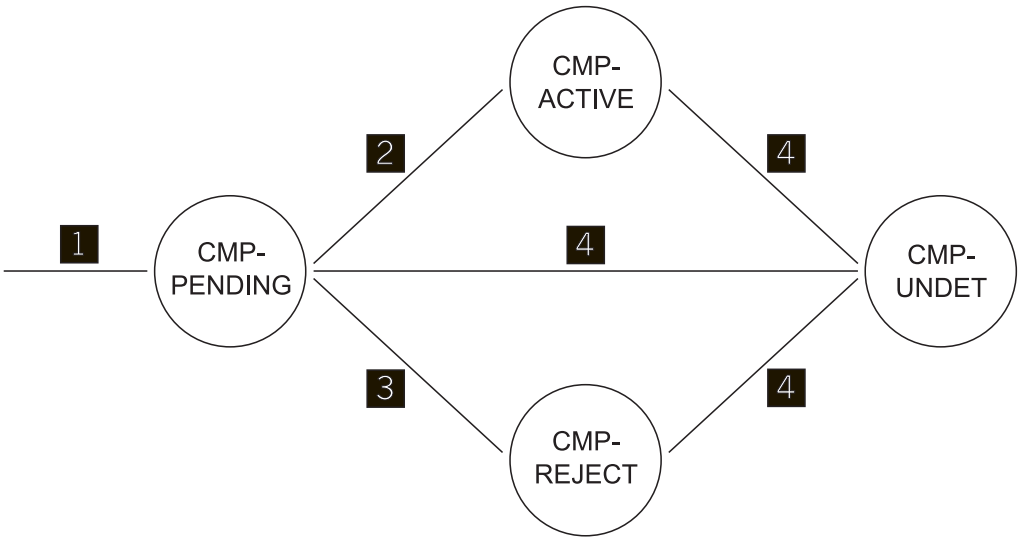


Figure 23. The Four Compression States of a Compressed Cluster

Data Format of Records

In general, the format of data records in compressed and nocompressed format is very similar. The control area and the RDF (record descriptor field) in the control interval is identical. However, the record as stored within the control interval has a different format. Each record can consist of the following parts:

1. The compressed record prefix. This prefix has a length of 3 bytes for nonspanned records, and a length of 5 bytes for spanned records. Its format is:

Offset Dec	Hex	Bytes	Hex Digit	Description
0	0	1		Flags
			X'40'	Record is compressed
1	1	2		Length of expanded record (nonspanned)
1	1	4		Length of expanded record (spanned)

1. The non-compressed part of the record. This applies only to files with a key, it is the first part of the record, up to (and including) the prime key.
2. The compressed part of the record.

How to Define the Compression Control Data Set

The VSE/VSAM Compression Control Data Set (CCDS) is an indexed cluster called `VSAM.COMPRESS.CONTROL`. A cluster with this name is always used as a CCDS, hence no other data set of this name must exist in any catalog. The compression control data set must be defined for each catalog into which compressed clusters are defined. It is recommended, however, to define a compression control data set for every catalog.

The ″Define a New User Catalog″ dialog of the z/VSE Interactive Interface defines a CCDS for each newly defined catalog.

You can also define a CCDS manually using the IDCAMS `DEFINE CLUSTER` command, as outlined below. Once the compression control data set is defined, compressed clusters can be defined or restored.

```
DEFINE CLUSTER -
       (NAME(VSAM.COMPRESS.CONTROL) -          1
        VOLUME(volser) -                       2
        RECORDS(200 100) -                     3
        KEYS(44 0) -
        RECSZ(128 500)-
        SHR(4 4) -                             4
        NOREUSE -
       ) -
       DATA (NAME(VSAM.COMPRESS.CONTROL.@D0)) -
       INDEX(NAME(VSAM.COMPRESS.CONTROL.@I0)) -
       CAT(catalog.id)                         5
```

Explanation:

1 The name identifies this cluster as a compression control data set.

2 The compression control data set should reside on the same volume as the catalog to which it is defined.

3 The compression control data set contains one record per compressed cluster. Specify a number that is sufficiently large to accommodate the number of compressed files you anticipate.

4 The CCDS is always defined with the SHAREOPTIONS(4 4) attribute.

5 Specify the name of the catalog for which you want to define the CCDS. Alternatively, you could identify the catalog using a `DLBL IJSYSUC` statement.

Which Data Set Types Are Eligible

The following types of data sets can be defined with the COMPRESSED attribute:

- KSDS files, that is clusters with the INDEXED attribute, under the following conditions:
  - The maximum record length must be greater than the sum of key offset, key length, and 40: `record_length - (key_offset + key_length) > 40`
- ESDS files, that is clusters with the NONINDEXED attribute, under the following conditions:
  - The maximum record length must be greater than 40.
  - The RECORDFORMAT attribute must not be specified (this implies that SAM files in VSE/VSAM-managed space cannot be compressed).
  - The file is not defined for use as a virtual tape.
- VRDS files, that is clusters with the NUMBERED attribute, under the following conditions:
  - The average record size is not equal to the maximum record size.
  - The maximum record length must be greater than 40.

The following types of data sets **cannot** be defined with the COMPRESSED attribute:

- Alternate index files
- Relative Record Data Sets (RRDS)

  If you would like to use VSE/VSAM data compression with your existing relative record data sets, you could attempt the following approach: change the `DEFINE CLUSTER` for the RRDS to specify a maximum record size that is larger than the average record size. This actually defines a VRDS rather than a RRDS, but the VRDS is eligible for data compression and offers a user interface that is almost identical to the user interface of a RRDS.
- SAM ESDS files
- ESDS files defined for use as virtual tapes.

Restrictions

The following restrictions apply to compressed data sets:

1. In a compressed file, you cannot update existing records using addressed access (RPL OPTCD=ADR). This implies that records of an entry-sequenced file cannot be replaced.
2. An application must not compute record positions (RBA) itself. Rather use `SHOWCB RPL=...,FIELDS=(RBA),...` instead.
3. A compressed file cannot be opened in control interval mode (MACRF=CNV in the ACB), except if the ACB also specifies MACRF=CMP. In this case all data passed to the application and expected from the application is in compressed (not expanded) format.
4. The data or index component of a compressed cluster cannot be opened by itself. VSE/VSAM would allow, however, an input open of a compressed data component.

The VSE/VSAM Compression Prediction Tool (IKQCPRED)

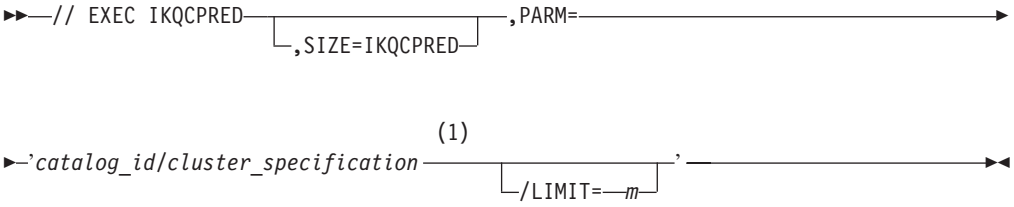
Before you compress data, you may want to know which of your VSE/VSAM data sets are suitable candidates for compression. IKQCPRED is a program that examines VSE/VSAM data and calculates how well it would compress.

A measure for compressibility is the *compression ratio*, which is the ratio of the length of the data in uncompressed format to the length in compressed format.

IKQCPRED can compute the anticipated compression ratio for one or more data sets residing in the same VSE/VSAM catalog on a z/VSE or VSE/ESA Version 2 system.

Using IKQPRED

Invoke IKQCPRED with the following JCL statement:



Notes:

1 The last or only character of *cluster_specification* can be the wild-card character *, indicating all clusters with a matching name up to the asterisk.

The parameters for IKQCPRED are specified on the PARM parameter of the EXEC IKQCPRED statement, and are separated by slashes(/):

- The first parameter is the target catalog name. This is mandatory, and must be fully qualified.
- The second parameter is the cluster specification. You can specify cluster names generically with an asterisk (*). IKQCPRED can process up to 400 clusters.
- The third, optional parameter LIMIT=*m*, where *m* is an integer greater than 0, specifies an upper limit (in megabytes) to the amount of data to be examined per cluster.

  Where *m* is 0, no upper limit is set, and each cluster is examined completely.

For performance reasons, IKQCPRED should be run in a relatively large partition. **2MB GETVIS(any)** should be sufficient. Specifying SIZE=IKQCPRED on the EXEC statement allows for maximum use of the partition GETVIS.

IKQCPRED Examples

Here are some typical uses for IKQCPRED, with examples of the control statements you could use, and a short description of what these statements would result in:

- Examine all files in a catalog

  ```
  // JOB IKQCPRED PREDICT VSAM DATA COMPRESSION RATIO
  // EXEC IKQCPRED,PARM='VSESP.USER.CATALOG/*/LIMIT=2'
  /&
  ```

  IKQCPRED scans *all* the data sets in catalog VSESP.USER.CATALOG. Up to 2 megabytes of data per cluster are scanned.

- Examine a group of files in a catalog

  ```
  // JOB IKQCPRED PREDICT VSAM DATA COMPRESSION RATIO
  // EXEC IKQCPRED,SIZE=IKQCPRED,PARM='VSESP.USER.CATALOG/TST*'
  /&
  ```

  IKQCPRED scans all data sets in catalog VSESP.USER.CATALOG whose names begin with 'TST'.

- Examine a single file

  ```
  // JOB IKQCPRED PREDICT VSAM DATA COMPRESSION RATIO
  // EXEC IKQCPRED,PARM='VSESP.USER.CATALOG/TST.KSDS3/LIMIT=8'
  /&
  ```

  IKQCPRED scans only the cluster TST.KSDS3 in catalog VSESP.USER.CATALOG. A maximum of 8 megabytes of data is scanned.

Method of Operation

IKQCPRED works internally in three phases:

1. IKQCPRED searches the specified catalog, and selects all cluster entries matching the specification on the PARM statement.
2. Clusters with inappropriate data-set attributes, such as NOCIFORMAT files (VSE libraries), are excluded from examination.
3. Each of the remaining data sets is opened, and each record fed into the VSE/VSAM data compression routines for interrogation and compression. The process ends at the end of a data set, or when the threshold specified in LIMIT=*m* is exceeded.

   The length of time required for this step depends on the amount of data to be scanned.

IKQCPRED prints the results for each examined cluster to SYSLST. The output is described in "Interpreting IKQCPRED Results."

Interpreting IKQCPRED Results

This section explains the output of the IKQCPRED program with reference to the sample output in Figure 24:

```
// EXEC IKQCPRED,PARM='VSESP.USER.CATALOG/TST.*/LIMIT=5'


                      VSE/VSAM Data Compression Prediction -       (Copyright IBM , 1995, 1996)
IKQ5000I Computing compression ratios for files in catalog VSESP.USER.CATALOG                      07/16/96
IKQ5003I This processor supports hardware data compression

Cluster Name                    Type     CmpStatus Ratio AvgLRECL  # Records Open  FDBK   Close (HU-)RBA
TST.KSDS1                       KSDS     CmpActive  1.38      256       3949 00/00 000000 00/00 0024C000
TST.SAM.ESDS1                   SAMESDS  CmpActive  7.88      256      12067 00/00 000000 00/00 0035E000
TST.ESDS1                       ESDS     CmpActive  1.01      256        300 00/00 000000 00/00 00015800
TST.ESDS2                       ESDS     CmpActive  7.42      256      12067 00/00 000000 00/00 0035E000
TST.ESDS3                       ESDS-CMP CmpActive  8.69      256      12107 00/00 000000 00/00 00056000
TST.ESDS4                       ESDS     CmpReject   .92       80        875 00/00 000000 00/00 00011800
TST.KSDS2                       KSDS-CMP CmpActive  1.30      256       1236 00/00 000000 00/00 00150000
TST.SAM.ESDS.IMPLICIT           SAMESDS  CmpActive  2.69      256        120 00/00 000000 00/00 00009000
TST.VRDS                        VRDS     CmpActive 12.36      256          * 00/00 000000 00/00 00E38000

1S55I  LAST RETURN CODE WAS 0000
EOJ CMPRATIO  MAX.RETURN CODE=0000                    DATE 07/16/96,CLOCK 09/56/06,DURATION   00/00/41
```

Figure 24. Sample IKQCPRED Output

Message IKQ5000I tells you which catalog was examined. Message IKQ5003I, as shown in the example, states that the processor supports the CMPSC instruction (**hardware** data compression).

If the processor does not support this instruction, you will see the message:

```
IKQ50004I This processor does not support hardware data compression
```

This indicates that software emulation has been used instead of hardware data compression. Performance is slower than on a processor that supports hardware data compression.

The results of the data compression prediction tool, arranged into several columns, show:

Cluster Name
The names of the clusters that were examined. Only cluster entries can be examined; other objects, such as AIXs, are not eligible for VSE/VSAM data compression.

Type The type of the data set being examined. The following types may appear:

Type	Remarks
ESDS	Entry sequenced (flat) file. ESDS files are eligible for VSE/VSAM data compression, but with one restriction: Existing records must not be updated. When you have compressed an ESDS, you can only append new records to it.
KSDS	Key sequenced (indexed) file. Only the part of the record following the prime key can be compressed, and only if it has a length of at least 40 bytes. Consequently, placing the key near the beginning of the record allows optimum compression.
RSDS	Relative record (numbered) file. RSDS files are NOT eligible for VSE/VSAM data compression. IKQCPRED examines them because it might be possible to define them as VRDS files, which can be compressed.
SAMESDS	CIFORMAT SAM file in VSE/VSAM managed space. SAMESDS files are NOT eligible for VSE/VSAM data compression. IKQCPRED examines them because it might be possible to define them as native ESDS files, which can be compressed.
VRDS	Variable length relative record file. Only records longer than 40 bytes can be compressed.
-CMP	The suffix -CMP indicates that the examined file was already defined with the COMPRESSED attribute.

CmpStatus
The compression status anticipated for the file:

CmpActive
Actual compression will achieve a significant result.

Pending
There is not enough data in the file to determine whether compression will achieve results. If possible, load more data into the file and rerun IKQCPRED.

CmpReject
The data set is expected to be compression-rejected, because the data is not suited for VSE/VSAM data compression.

Ratio The **compression ratio**. IKQCPRED computes the approximate compression ratio as:

```
          (sum of lengths of uncompressed data records)
Ratio =  ---------------------------------------------
           (sum of lengths of compressed data records)
```

The larger the value of `Ratio`, the better the file will compress. In other words, the file is expected to shrink to `1/Ratio` of the size of the uncompressed file.

Because of rounding effects and the way the records happen to fit into the control intervals, the computed compression ratio may differ from the compression ratio that could actually be achieved. However, this effect should be significant only for SPANNED data sets with relatively short records. Each record in a spanned data set begins in a new control interval.

AvgLRECL
The average record length of the file. This is the *actual* average record length, *not* the value specified on the DEFINE CLUSTER command.

 # Records
The number of records in the data set. This column shows an asterisk (*) if data interrogation ended at the threshold specified by LIMIT=*m*.

Open The VSE/VSAM return and reason code, if an OPEN error occurred. It would be normal to see some errors in this column. For example, 08/6E would indicate that this cluster is empty, or 08/A8 would indicate that the cluster cannot be opened because it has been opened from another partition.

FDBK The VSE/VSAM feedback information, if a record management error occurred while reading the data set. Under certain circumstances, record management errors can be tolerated when processing SAM files in VSE/VSAM managed space. This is the case when, for example, the SAM files have been written using non-VSAM (that is, DTFSD) access.

Close The VSE/VSAM return and reason codes, if a CLOSE error occurred

(HU-)RBA
This column normally shows the hexadecimal number of bytes in the high-used-RBA of the data set. However, if a record management error occurred, it shows the current RBA for which the error indicated in the FDBK column was posted.

The IKQCPRED return code is the highest return code encountered during processing. If, for example, one or more data sets could not be examined because they were empty, the return code would be 4.

Chapter 6. Device Dependencies

This chapter discusses special functionality, restrictions, and exceptions applying to specific types of devices.

VSE/VSAM Support of Large DASD

VSE/VSAM supports DASD with a capacity exceeding 65535 (64K) tracks, referred to in this manual as *large DASD*. (Accordingly, DASD with a capacity of 64K tracks or less is referred to as *small DASD*.) This support applies up to a capacity of 10017 cylinders (150255 tracks), which corresponds to the capacity of an IBM 3390 Model 9.

If you try to define a VSE/VSAM catalog or space on a DASD volume that exceeds this limit of 10017 cylinders, you will receive the following message:

```
IDC0055I  VOLUME SPACE EXCEEDS MAXIMUM VSAM CAPABILITY. MAXIMUM WILL BE USED.
```

Large DASD implementation does not change the mapping of free and used tracks of the space map in the catalog. However, using large DASD the number of space map segments and catalog records used will increase. One catalog record can hold one segment of the space map, which describes 3520 tracks. For small DASD, the maximum number of space map segments is 19 (this means that 19 catalog records are required to map 65535 tracks on one disk. For the IBM 3390 Model 9, which has 10017 cylinders and 150255 tracks, the catalog will map the tracks of this disk device type within 43 catalog records.

Making Use of the Support

The following IDCAMS DEFINE commands can be used for defining space allocation:

```
DEFINE MASTERCATALOG
DEFINE USERCATALOG
DEFINE SPACE
DEFINE CLUSTER UNIQUE
DEFINE ALTERNATEINDEX UNIQUE
```

The above commands internally check the disk capacity. They use either:

- Up to 65535 tracks of the disk (the support before large DASD) when:
  - the disk does not have more than 65535 tracks, or
  - the current catalog owns VSAM space on that disk that was defined before VSE/ESA 2.6.
- Up to 10017 cylinders of the disk (the support introduced with large DASD) when:
  - the disk has more than 65535 tracks, and
  - the current catalog does *not* own VSAM space on that disk that was defined before VSE/ESA 2.6.

For an IBM 3390-9 or other large DASD, this means that either:

- VSAM space was already allocated on this disk for the current catalog from a previous VSE/ESA release. The disk will therefore not have large DASD support, and will only be supported as a “64K track disk”.

© Copyright IBM Corp. 1979, 2005

- The VSAM space was not used by VSE/VSAM before VSE/ESA 2.6. The disk will therefore have large DASD support, and a new flag bit in the Catalog Volume Record will indicate a “Large DASD”.

In addition, for an IBM 3390-9 or other large DASD:

- Where possible, allocations other than CYLINDERS (for example, TRACKS or RECORDS) will be translated internally to multiples of CYLINDERS. Or, if track allocation is required by VSE/VSAM for Large DASD, allocations of tracks or records (for example) will be translated internally to multiples of tracks-per-cylinder. For example, a VSE/VSAM cluster with a primary (and secondary) allocation of one track will have a primary (and secondary) allocation of one cylinder. Direct allocation in CYLINDERS is recommended.
- Track boundaries of extents will be rounded to cylinder boundaries.
- If a catalog resides on a large DASD, the minimum allocation for the catalog is 5 cylinders (4 cylinders for the data component plus 1 cylinder for the index component).

Migrating to Large DASD Using IDCAMS Backup/ Restore

Most files that have been defined using a CI size of 512 *cannot* be migrated using IDCAMS Backup /Restore to a large DASD device (this is due to internal restrictions). Backup/Restore is intended to transfer files with high performance, and is based on the CI Size.

Any file defined with IMBED option cannot be restored to or defined on a large DASD.

If you want to use files that have been defined with a CI size of 512 on a large DASD, you must follow these general steps:

1. Restore the files that have a CI size of 512, to a previously supported small DASD type. You use the IDCAMS RESTORE command to do this.
2. Export the files from the previously supported small DASD type using the IDCAMS EXPORT command.
3. Import the files to a large DASD using the IDCAMS IMPORT command.

For further details about using the above IDCAMS commands, refer to *VSE/VSAM Commands*, SC33-8245.

Performance Considerations (KSDS Only)

The performance of KSDS access may change if the data control area size (data CA) changes. One index control interval (index CI) controls one data control area. The larger the data control area together with a large index control interval, the better the keyed access performance. This is because less index I/O is required for keyed-direct and keyed-sequential access. VSE/VSAM calculates the control area size from the smaller of the primary or secondary allocation. The minimum is one track, and the maximum is one cylinder (15 tracks).

Note: A control area size of one cylinder is recommended.

Where possible, a VSE/VSAM KSDS on a large DASD will have a control area size of one cylinder. Primary and secondary allocations are rounded up to cylinder multiples and cylinder boundaries, even if they have been defined as TRACKS or RECORDS. To also get a control area size of one cylinder for long keys (up to 255 bytes), VSE/VSAM calculates the minimum data control interval size (CI size) of a

KSDS and increases it where required. The following key lengths require the following minimum control interval sizes:

Table 4. Minimum CI Sizes Depending on Key Length

Key Length in Bytes	Minimum CI Size
7 - 35	1024
36 - 55	2048
> 55	4096

BUFFERSPACE Parameter

The BUFFERSPACE parameter could force a smaller data control area size, and must have a size that is at least two data control intervals plus one index control interval. It is recommended not to use this parameter with DEFINE CLUSTER. Large DASD support ensures that the BUFFERSPACE parameter will not reduce the CA size.

Increased Size of the Catalog Index

As a result of large DASD support, the index primary allocation of a catalog on *all* DASD (large or small) will be at least 4% of the primary catalog data allocation. The catalog index secondary allocation will have the same size as the index primary allocation.

Restrictions for VSE/VSAM Support of Large DASD

The following restrictions apply when using large DASD with VSE/VSAM:

- No support for imbedded indices:
  - The definition of imbedded indices for catalogs, alternate indices, and clusters is not supported. If the keyword IMBED is used in existing IDCAMS DEFINE jobs, it will be ignored or rejected with an error message.
- The catalog default is NOIMBED:
  - For large DASD support, the default value for DEFINE MASTERCATALOG and DEFINE USERCATALOG is NOIMBED.
  - Newly defined catalogs (MASTERCATALOG and USERCATALOG) will never have an imbedded index.
- No CRA on Large DASD:
  - There is no support for the CRA (Catalog Recovery Area) keyword RECOVERABLE on large DASD.
  - Existing recoverable catalogs and CRAs on small DASD will be accepted and supported.
  - New CRAs can be defined on small DASD, but no CRA can be defined on a large DASD.
  - Recoverable catalogs cannot reside on a large DASD and cannot own space on a large DASD.

New or Changed Fields in LISTCAT Output

Large DASD support is reflected in LISTCAT output as follows:

- The field DEVTYPE has the prefix “BIG-” for a large DASD (for example, `DEVTYPE-----BIG-3390`). The prefix “BIG-” is used either in the Volume Group (DATA, INDEX) or in the Volume Entry.
- The field TRACKS in the Volume Group (DATA, INDEX) is replaced by CYLINDERS for a large DASD.

- The field SPACE-MAP in the Volume Entry is replaced by CYL-SPC-MAP, which indicates cylinder mapping instead of track mapping for a large DASD.

Support for FBA Disk Devices (FBA and SCSI)

z/VSE 3.1 is designed to allow IBM eServer zSeries servers to attach industry-standard Small Computer System Interface (SCSI) disk devices via zSeries Fibre Channel Protocol (FCP) channels.

User-written programs use VSE's existing Fixed Block Architecture (FBA) support (512 byte blocks) to access SCSI disks. User programs cannot use SCSI commands directly. z/VSE 3.1 is designed to support SCSI disk volume sizes from 8 MB to 24 GB. Because z/VSE itself uses the first 4 MB for internal purposes, the available user space is equal to the defined size of the disk minus 4 MB. z/VSE 3.1 limits VSE/VSAM to the first 16 GB of any SCSI volume.

Special migration considerations apply:

- It is not possible to use the Fast Service Upgrade (FSU) process to move from a VSE/ESA 2.6 or 2.7 system to a z/VSE 3.1 system with SCSI system residence disks.
- Not every cluster can be migrated using Backup/Restore. In some cases, Export/Import must be used.

Similar restrictions apply to FBA to SCSI migration.

z/VSE SCSI-FCP disk support complements SCSI support in z/VM Version 5 and Linux for zSeries. The individual z/VSE maximum SCSI volume size limits do not apply to z/VM minidisks backed by SCSI disks. When operating as a guest under z/VM (using SCSI disks not directly attached to z/VSE), z/VM presents SCSI disks as 9336-20 FBA disks. In this case, z/VSE sees them as FBA, not SCSI disks.

The maximum size of a z/VSE 3.1 FBA volume is 2 GB. Of course, multiple 2 GB minidisks can be assigned within the limits of a single physical SCSI disk volume controlled by z/VM. For SCSI disks directly attached to z/VSE under z/VM, the normal z/VSE limits described above apply.

Technical Considerations

VSE/VSAM extends the existing FBA logic to support SCSI disks. VSAM implements a SCSI disk as a generic FBA device and uses its own "virtual characteristics" for mapping and building channel programs for optimized VSAM performance and space utilization.

Except as noted, all commands, parameters, and requirements for FBA devices are valid for SCSI as well.

Several FBA configurations are supported. The generic FBA model is used to:

- simulate an FBA device in virtual storage; for example, the user can defined in CP:

  ```
  CP DEF VFB-512 AS 152 BLK 100000
  ```

  This virtual disk will be presented to the user under VSE as an FBA disk after the initialization with ICKDSF:

```
volume 152
AR 0015 CUU  CODE DEV.-TYP   VOLID  USAGE   SHARED    STATUS    CAPACITY
AR 0015 152  90   9336-10   FBA001  UNUSED                         99960 BLK
AR 0015 1I40I  READY
```

The upper limit for VM to address the storage is 2 GB, which agrees with the supported size of VSE/VSAM.

- access a VM minidisk (as part of a real SCSI device). VSAM can address only 2 GB. (z/VM 5.1 allows defining VM FBA minidisks with a larger size, but VSE/VSAM can only handle a 2 GB FBA disk in this case.)

  The nature of this 2 GB limit can be explained as follows: VSE/VSAM supports so-called Generic FBA Devices with a virtual FBA disk device characteristic of 64 FBA blocks per track and 15 tracks per cylinder, that is: 960 FBA blocks per cylinder = 491,520 bytes per cylinder.

  A space map in the catalog maps each track of a particular disk device with 1 bit (0 = track used, 1 = track free). Additional catalog fields and control blocks map the number of tracks and the start/end track of data spaces and data set extents in 2-byte-fields, which limits the maximum capacity of one DASD device to X'FFFF' = 65,535 tracks (64K - 1).

  Therefore, the current maximum FBA disk capacity is 65,535 tracks * 64 FBA blocks = 491,520 FBA blocks = 2,147,450,880 bytes = 2 gigabytes.
- directly access the SCSI device as an FBA-SCSI device (via FCP). The limit for an FBA-SCSI device is 24 GB, and VSAM can use up to 16 GB on this device by using a different device model (different min-CA and max-CA) as shown in the following table:

Table 5. Minimum and Maximum CA for Generic FBA Devices

FBA Device	Blocks per Minimum CA	Minimum CA per Maximum CA	Blocks per Maximum CA
Generic Virtual FBA	64	15	960
Generic FBA as VM Minidisk	64	15	960
FBA-SCSI	512	60	30720

The new device model for SCSI significantly improves the performance (due to fewer CI/CA splits, for example) but requires the system programmers to review, re-calculate, and possibly adapt the space definitions of the JCLs (for example, the minimum size for a catalog (6 min-CAs) and the size for space sub-allocation are different).

Restrictions

The following notes and restrictions apply to VSAM structures on SCSI disks:

- The minimum CA (min CA) is 512 blocks, which implies that the minimum file size for VSAM on such a device is 512 blocks. Space specifications will be rounded up to a multiple of 512 blocks (for example, 10000000 blocks will effectively become 10000384).

  Note: Due to different rounding values (different min CA values on FBA and SCSI), it is not guaranteed that the same JCLs will run on generic FBA and SCSI devices.
- The absolute minimum space specified for primary allocation is 2561 blocks, which is rounded to 3072.

- If the ORIGIN option is used during cluster definition, the minimum specified must be 3072, because rounding will not be performed in this case.
- The maximum CA (max CA) on a SCSI disk is 30720 (60*512) blocks, i.e., the min CA per max CA is 60.
- Any cluster defined in blocks with a key length >38 requires a minimum CISIZE of 1024.
- If no key length is specified, the default will be used, which in most cases is 64. In this case, any cluster definition with a CISIZE of 512 (smallest possible value under VSAM) will be rejected by VSAM with a corresponding error code
- Migration to any cluster defined on a SCSI device must be done using REPRO or EXPORT/IMPORT. The use of IDCAMS BACKUP/RESTORE is not recommended for long-term recovery or data migration and is not supported.
- VSAM data can be transferred to SCSI using IDCAMS BACKUP/RESTORE. For certain files that cannot be restored with IDCAMS RESTORE because of file definition restrictions for SCSI, IDCAMS REPRO should be used. This includes, but is not limited to, all files defined with the SPANNED option and some files defined with very small allocations or CI sizes.
- The parameters IMBED, REPLICATE, and RECOVERABLE are no longer supported and are either ignored or rejected with an error message.

  Migration of older clusters defined with any of these options should be performed using IDCAMS REPRO.
- The entire SCSI device can be made available to VSAM if SPACE is defined with option DEDICATE. Otherwise, up to X'FFFFFF' (16,777,215) blocks can be specifed (this is the same restriction as for the current RECORDS parameter).
- The hardware architecture of large DASD and SCSI devices imposes minimum allocation requirements in VSAM device support (1 cylinder, 512 block minimum CA size).

VSAM detects and reports the SCSI disk as device type 'FBA' on LISTCAT output:

```
CHARACTERISTICS
  BLKS/MIN-CA----------512      DEVTYPE---------FBA
  BLKS/MAX-CA--------30720      VOLUME-TIMESTAMP:
```

IUI dialogs for FILE AND CATALOG MANAGEMENT under RESOURCE DEFINITION assist the user in generating the JCLs for IDCAMS jobs to support SCSI devices.

Virtual Tapes

Local virtual tapes are implemented as standard VSAM ESDS files. One restriction, however, is that they must not be compressed. Information on using virtual tapes is provided in *z/VSE Planning*.

Chapter 7. Optimizing the Performance of VSE/VSAM

This chapter ...

Explains the following VSE/VSAM **options** that affect **performance**:

- "Data Space Classification"
- "Control Area (CA) Size" on page 85
- "Control Interval (CI) Size" on page 88
- "I/O Buffer Space (Using Non-Shared Resources)" on page 94
- "I/O Buffer Space (Using Local Shared Resources)" on page 99
- "Multiple Volume Support" on page 102
- "Space Allocation" on page 108
- "Data Protection and Integrity Options" on page 110
- "Distributed Free Space" on page 111
- "Index Options" on page 120

For an outline on **file statistics** that are available to you for evaluating possible performance improvements, refer to "Performance Measurement" on page 121.

Most of the options are specified in the IDCAMS command DEFINE when creating a file, and in the VSE/VSAM macros ACB and GENCB when a processing program prepares to open a file.

Because of the great number of variables, not everything presented in this chapter is true for all installations and under all conditions.

Number of Files Defined in a Catalog

The number of files defined in a catalog can have a direct impact on the performance of most VSE/VSAM activities. Generally, it is recommended that files on a single volume be defined in a unique user catalog.

A large number of files in a single catalog (for example, a thousand files) can significantly increase the run time for most IDCAMS functions. This includes DEFINE, DELETE, and LISTCAT functions. It also impacts open and close performance.

The exact number of files at which the impact on performance becomes noticeable depends on several factors (for example, DASD access speed and file name pattern). As the number of files in a single catalog increases, you should carefully monitor the performance of the indicated IDCAMS functions.

Data Space Classification

To direct the suballocation of data space to VSE/VSAM objects, you can assign a *class value* to VSE/VSAM data space; it allows you to optimize performance. Specify the value in the CLASS(value) parameter of the command DEFINE SPACE, DEFINE MASTERCATALOG, or DEFINE USERCATALOG. You can specify a value from 0 to 7.

© Copyright IBM Corp. 1979, 2005

Performance: Data Space Class

After you have assigned a value to a data space, you can request that it will be available for suballocation to an alternate index or cluster (or their components). Make the request in the USECLASS parameter of the command DEFINE CLUSTER, DEFINE ALTERNATEINDEX, or IMPORT.

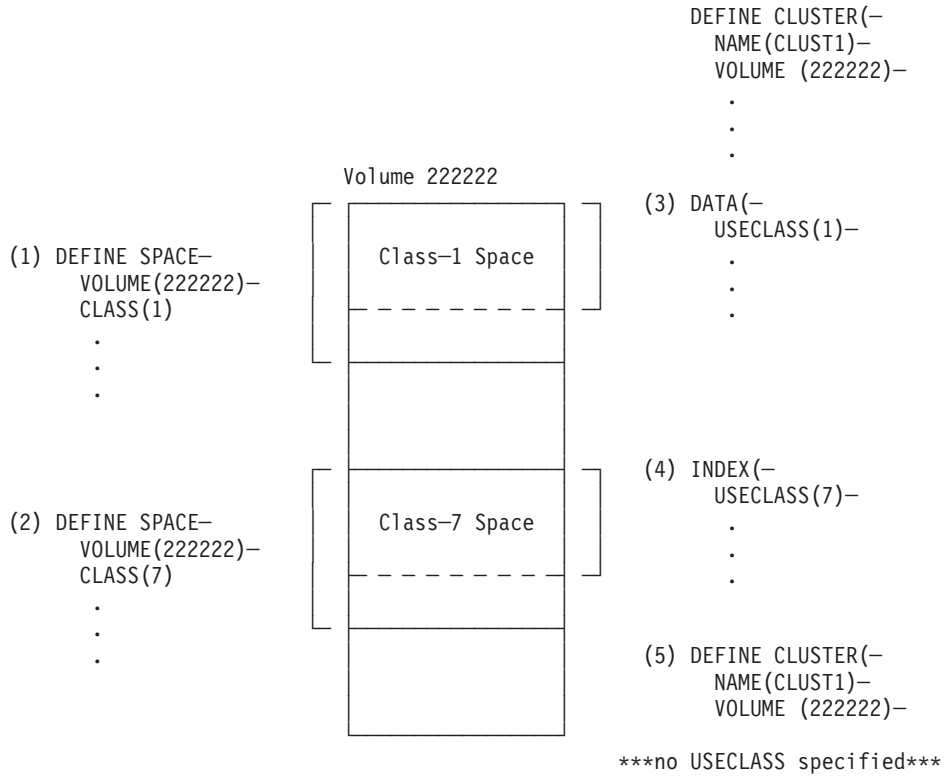
The class values and their meaning are:

0 General use and default. (Data spaces defined under MVS/VSAM are treated as class-0.)

1 High performance (specifically suggested for fixed-head areas).

2-7 User-defined classes (for example, data space in the middle of a volume).

Figure 25 illustrates the classification of data space and the use of classified data space.




```
Explanation:

  (1) Class—1 data space defined
  (2) Class—7 data space defined
  (3) Class—1 data space suballocated to the data component of CLUST1
  (4) Class—7 data space suballocated to the index component of CLUST1
  (5) This DEFINE command fails, because the default class (0)
      is not available on volume 222222
```

Figure 25. Classification of Data Space

The definition of VSE/VSAM catalogs involves the implicit allocation of data space and the suballocation of some (or all) of that data space to the catalog itself. Because of this, you need only specify the CLASS parameter if you want to assign a catalog's data space to a certain performance class. You do not have the option of

specifying the USECLASS parameter. The catalog is automatically suballocated from the same data space and the same performance class.

You can request a new class through the USECLASS parameter in the IMPORT command when an object is implicitly defined through this command.

The following restriction applies:

- Classes other than 0 are not permitted for unique objects.

For the DEFINE command, you must specify USECLASS concurrently (at the same level) with the space parameters (TRACKS, BLOCKS, and so on). For example, if you specify USECLASS in DEFINE CLUSTER at the data level, you must also specify CYLINDERS, TRACKS, BLOCKS, or RECORDS at the data level. If you do not do so, the USECLASS specification will be ineffective. The following are the three possible combinations of levels at which space may be specified for DEFINE CLUSTER or DEFINE ALTERNATEINDEX:

(a) Cluster level only or alternate index level only
(b) Data component level only
(c) Data component and index component levels

Therefore, these are also the levels that are effective for USECLASS.

In case **(a)**, the USECLASS specified (or defaulted to) is also applied to the data and index components.

In case **(b)**, the USECLASS specified, defaulted to, or modeled for the data component level is also applied to the index component level. This permits you to apply the same class of data space to both components while leaving the calculation of the index allocation to VSE/VSAM.

In case **(c)**, the data and index components may be assigned (or modeled or defaulted) to a separate or to the same class of data space, depending on the values chosen.

For information on *assigning* classes of data space, refer to the USECLASS parameter in the index of the manual *VSE/VSAM Commands*, for example, for the "DEFINE ALTERNATEINDEX" command.

Control Area (CA) Size

Minimum and Maximum CA Sizes

The terms *minimum control area size (min CA)* and *maximum control area size (max CA)* are device independent terms. For CKD devices, the term:

- *minimum CA* relates to *track size*
- *maximum CA* relates to *cylinder size*

For FBA devices, however, the terms *tracks* and *cylinders* (as used for CKD) are not meaningful, because FBA devices store data on *fixed-size blocks* (where the blocks are not associated with tracks or cylinders).

The terms of minimum CA and maximum CA, however, are common to both CKD and FBA devices.

For applicable values for the various IBM CKD and FBA devices, refer to Table 6 on page 87 and Table 7 on page 87.

For CKD devices, the CA allocation limits depend on the:

- Index CI size you indicate, if the BUFFERSPACE parameter prevents an increase of the index CI size.
- Primary or secondary allocation.

Note: If VSE/VSAM runs in an environment where simulated devices are used , VSE/VSAM depends on the characteristics provided by the simulating system. These device characteristics may be different from those of the simulated device type. Therefore, VSE/VSAM may use unexpected device characteristics.

Performance Implications

In the case of a key-sequenced file, the size of a CA can affect the size of the CI of the index component. If there is not enough room for index entries in the sequence set record, VSE/VSAM increases the CI size to accommodate more entries.

CA size has significant performance implications. When a whole number of CAs occupies a maximum CA (cylinder), performance is better than when CAs cross maximum CA (cylinder) boundaries. If you allocate space in a DEFINE command using the CYLINDERS parameter, or if a CKD file is defined as *unique* (that is, the file is the only one in its data space), IDCAMS sets the CA size to one maximum CA (cylinder). If a CA is smaller than a maximum CA (cylinder), its size will be an integral multiple of minimum CAs (tracks), and it can cross maximum CA (cylinder) boundaries. However, a CA can never cross the extent boundaries of a file; that is, an extent of a file is made up of a whole number of CAs.

Aside from specifying space in terms of maximum CAs (cylinders) or defining a CKD file as unique, you do not have a direct way of specifying that a whole number of CAs will occupy a maximum CA (cylinder). But, you can provide values in the DEFINE command that will influence the CA size as computed by IDCAMS.

IDCAMS checks the smaller of the primary and secondary space values against the maximum CA (cylinder) size of the specified device. If the smaller space quantity is less than or equal to the device's maximum CA (cylinder) size, the size of the CA is set equal to the smaller space quantity. If the smaller space quantity is greater than the device's maximum CA (cylinder) size, the CA size is set equal to the maximum CA (cylinder) size.

You specify space in number of tracks, cylinders, blocks, or records; the system then preformats space in CAs (except for DEFINE CLUSTER/AIX SPEED). By calculating the size of a CA as it does, IDCAMS is able to meet your primary and secondary space requirements without over committing space for this file.

An index record must be large enough to address all of the CIs in a CA. The more CIs an index record addresses, the fewer reads for index records are required for sequential access. Generally, the greater the size of the CA, the better the performance and space utilization.

Disk Storage Sizes

Table 6 on page 87 lists values for cylinders, tracks, and other capacities for IBM CKD devices.

Table 6. Disk Storage Sizes for IBM CKD Devices

IBM CKD Device	Cylinders (Max CA) per Volume	Tracks (Min CA) per Cylinder	Bytes per Track (1)	Maximum Total Capacity (Bytes)
3375	959	12	19,456 -33,280	382,986,240
3380 ADJ	885	15	23,552 - 45,056	570,931,200
3380 E	1770	15	23,552 - 45,056	1,141,862,400
3380 K	2655	15	23,552 - 45,056	1,712,793,600
3390-1	1113	15	25,088 - 55,296	846,236,160
3390-2	2226	15	25,088 - 55,296	1,692,472,320
3390-3	3339	15	25,088 - 55,296	2,538,708,480
3390-1 (2)	1113	15	23,552 - 45,056	718,018,560
3390-2 (2)	2226	15	23,552 - 45,056	1,436,037,120
3390-3 (2)	3339	15	23,552 - 45,056	2,154,055,680
3390-9 (3)	10017	15	56,664	8,514,049,320

Notes:

1. Depending on the physical block size (see Figure 27 on page 91)
2. When in 3380 track compatibility mode.
3. Large DASD

VSE/VSAM treats the IBM 3995 Model 151 Optical Library Dataserver as an IBM 3390 Model 2 direct access storage device.

An Enterprise Storage Server (ESS) is reflected as a 3390-3 or 3390-9 depending on the number of cylinders available.

Table 7 lists values for minimum and maximum CA, and 512-byte blocks for IBM FBA devices.

Table 7. Disk Storage Sizes for IBM FBA (and SCSI) Devices

IBM FBA Device	Max CA per Volume	Min CA per Max CA	Blocks* per Min CA	Blocks* per Max CA	Total Blocks
0671	See (1)	8	63	504	See (1)
3370-1	750	12	62	744	558,000
3370-1	958				712,752
9332-1 (2)	1,233	4	73	292	360,036
9332-2 (3)	1,900	4	73		554,817
9335	1,890	6	71	426	805,140
Other FBA (4)	See (1)	15	64	960	See (1)
SCSI (5)	See (1)	60	512	30720	See (1)

Notes:

1. Configuration or device dependent.
2. Models 200, 400, 402.
3. Models 300, 600, 602.
4. For example, IBM 9336 and virtual disk.
5. Appears as device type FBA.

(*) 1 block = 512 bytes.

Control Interval (CI) Size

How to Specify

You can let IDCAMS select the size of a CI for a data or index component, or you can specify CI size in the DEFINE command. CI size should be specified at both DATA and INDEX levels. If the CI size is specified at the CLUSTER or ALTERNATEINDEX level, this size applies to the data component and also to the index component.

The CI size you specify is checked for being within *acceptable limits*. IDCAMS tries to modify an unacceptable value. If it cannot, the DEFINE fails. If you specify a CI size that is not a proper multiple, IDCAMS increases it to the next multiple. For example, 2050 is increased to 2560.

Data CI and Block Sizes

The limits (mentioned above) depend on the maximum (nonspanned) or average (spanned) *record size* that you specify in the RECORDSIZE parameter of the DEFINE command.

Note that a CI is always a multiple of the physical block size.

Physical Block Size for Data Component

A physical block (or physical record) is any multiple of:

- 512 bytes up to 8,192 bytes
- 2048 bytes from 8,193 bytes to 30,720 bytes

Figure 26 on page 89 shows how VSE/VSAM computes physical block size, using DEFINE attributes and device type. The following explains the numbers shown in the figure:

(1) Maximum record size can be specified as 1 through 32761; the default is 4089 bytes.

(2) The CONTROLINTERVAL size of the data component can be specified as 512 through 32768; the default is:

2048 bytes if RECORDSIZE is specified

4096 bytes if RECORDSIZE is not specified

(3) The control area size chosen by VSE/VSAM is never larger than one max CA (cylinder).

(4) BUFFERSPACE(size) must provide enough space to accommodate:

two control intervals, and

one index control interval if the file is key-sequenced.

This is also the default. If you specify less than the default, the command is terminated.

(5) The physical block size chosen by VSE/VSAM depends on the device type that is being used, and on the size of the control interval. The physical block size chosen by VSE/VSAM is:

For CKD devices:

- 512 bytes up to 8,192 bytes
- 2048 bytes from 8,193 bytes to 30,720 bytes

For FBA devices: always 512 bytes.

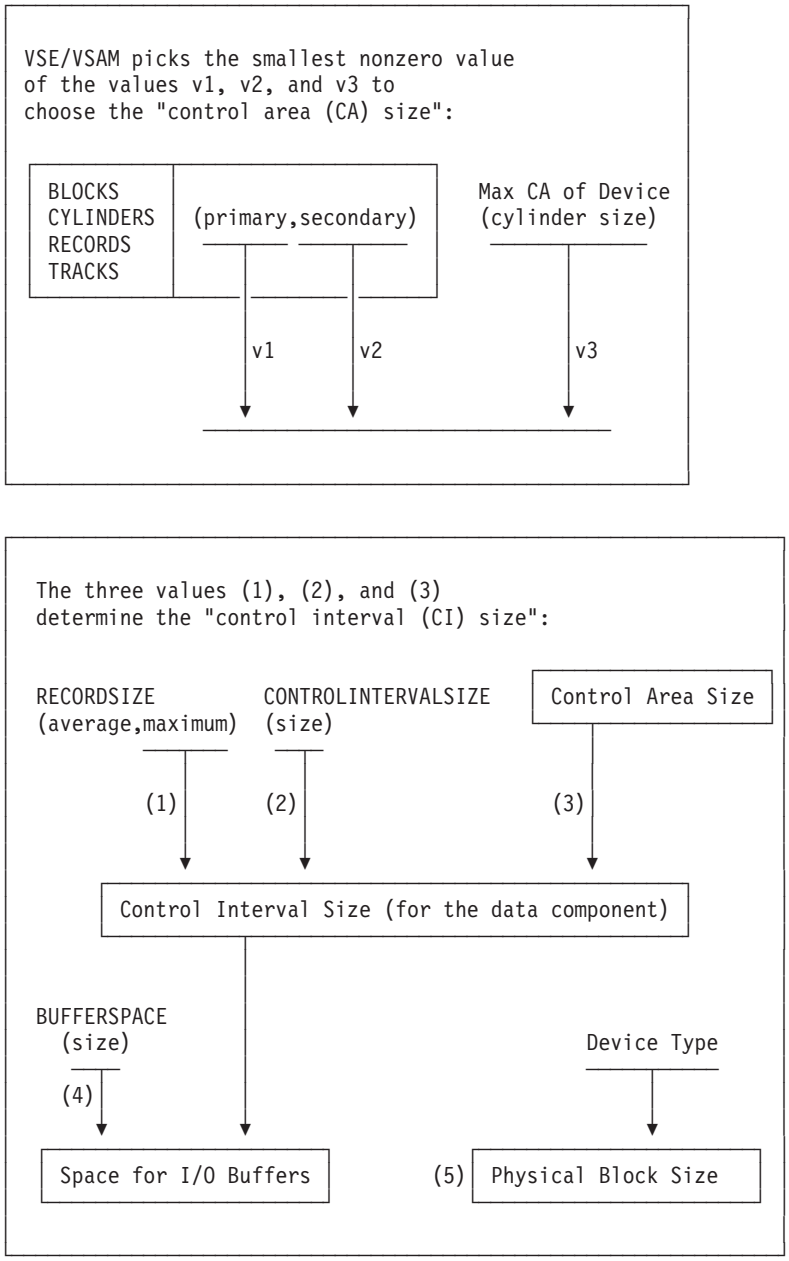


Figure 26. How VSE/VSAM Computes Physical Block Size

CI Size in a Data Component

The size of a CI in the data component can be any multiple of 512, up to 32,768. If it is over 8,192 bytes, it must be a multiple of 2048.

For nonspanned records, the CI must be at least seven bytes larger than the largest record in the data component.

For spanned records, the CI must be at least ten bytes larger than the average record in the data component.

Average and largest record are specified in the RECORDSIZE parameter.

CI size affects space utilization because of the way VSE/VSAM chooses physical block sizes on CKD devices. (There are no similar considerations for FBA devices.) For a given CI size, VSE/VSAM chooses the physical block size that results in the most efficient use of track capacity.

Note: A file with a data physical block size or index CI size other than .5, 1, 2, or 4KB cannot be directly processed by MVS. (File portability between VSE/VSAM and MVS via EXPORT/IMPORT is not impacted by data physical block size, but it does require an MVS-compatible CI size.)

Figure 27 on page 91 shows the physical block size that VSE/VSAM uses for a data CI, and the number of KB (kilobyte) of user data that can be accommodated on the track (the values depend on the specified CI size and the device that is used). For example, given a CI size of 6KB on a 3380, VSE/VSAM chooses a physical block size of 6KB that results in 42KB (plus overhead) of data on a 43008-byte track.

VSE/VSAM treats the IBM 3995 Model 151 Optical Library Dataserver as an IBM 3390 Model 2 direct access storage device.

CI Size	Physical Block Size (in KB)				Track Space Used (in KB)			
	3375	3380	3390	9345	3375	3380	3390	9345
0.5	0.5	0.5	0.5	0.5	20	23	24.5	20.5
1	1	1	1	1	25	31	33	28
1.5	1.5	1.5	1.5	1.5	27	34.5	39	31.5
2	2	2	2	2	28	36	42	34
2.5	2.5	2.5	2.5	2.5	30	37.5	42.5	35
3	3	3	3	3	30	39	45	36
3.5	3.5	3.5	3.5	3.5	31.5	38.5	45.5	38.5
4	4	4	4	4	32	40	48	40
4.5	4.5	4.5	4.5	4.5	31.5	40.5	45	36
5	5	5	5	5	30	40	45	40
5.5	5.5	5.5	5.5	5.5	27.5	38.5	49.5	38.5
6	6	6	6	6	30	42	48	36
6.5	6.5	6.5	6.5	6.5	32.5	39	45.5	39
7	3.5	7	7	7	31.5	42	49	42
7.5	7.5	7.5	7.5	7.5	30	37.5	45	37.5
8	8	8	8	8	32	40	48	40
10	5	10	10	10	30	40	50	40
12	4	6	12	4	32	42	48	40
14	3.5	14	7	14	31.5	42	49	42
16	16	8	16	8	32	40	48	40
18	4.5	6	18	18	31.5	42	54	36
20	4	20	10	20	32	40	50	40
22	2	22	5.5	22	28	44	49.5	44
24	8	6	24	8	32	42	48	40
26	6.5	6.5	26	6.5	32.5	39	52	39
28	4	14	7	14	32	42	49	42
30	7.5	6	10	10	30	42	50	40
32	16	8	16	8	32	40	48	40

Figure 27. Relationship of CI Size to Physical Block Size for Data Component

Performance Considerations

For performance improvement, consider the following rules.

- The larger the data CI, the better the *sequential* performance. EXPORT and IMPORT are sequential applications.
- As the size of your *nonspanned data records* increases, you may need larger data CIs.
- As data and index CI size increases and record size remains unchanged, more buffer space is required in storage for every CI.
- Free space will probably be used more efficiently as data CI size increases relative to data record size, especially with variable-length records.

  Free space in a *nonspanned data CI* is not used if there is not enough free space for a complete data record. In any event, free space in the last CI of a spanned record is never used for any other record, even if there is room enough to hold a complete data record.
- Direct processing is less sensitive to data CI size. But smaller data CIs generally improve performance.
- When you process input that already has been sorted, on the other hand, large data CIs may be better.
- If you have a choice between a large index CI or a large data CI for direct processing, choose the combination that yields the smallest buffer space value. This combination needs the least active storage and the least data transfer time.

CI Size in an Index Component

The CI size in the index component can be any multiple of 512, up to 8,192 bytes. Generally a 512-byte index CI is adequate if:

- The number of data CIs per CA is small,
- The full key size is not too large, and
- The key compresses well (usually when the data CI is 4KB or greater).

IDCAMS might adjust your specifications. To find the values actually set in a defined file, you can issue the IDCAMS LISTCAT command or, while your program is executed, the SHOWCB macro.

Considerations

Generally, you should specify the smallest index CI size that still is adequate.

You may want to specify the smallest value, that is 512. To find out if a 512-byte index CI size is adequate, do the following experiment:

- Use your chosen data CI size and a 512-byte index CI.
- Do not allow free space.
- Load enough records to equal one CA.
- At the end of the run, perform a LISTCAT.

  If there is only one level of index, a 512-byte index CI is large enough. For n CAs, there should be two levels of index with the number of index CIs equal to n + 1.

A smaller data CI may require a large index CI. The sequence set index CI contains pointers to the data CIs in a CA. If the data CI is made smaller (when the CA stays the same size), there will be more data CIs per CA, and therefore more entries in the sequence set. As an example, assume a one cylinder CA size on an IBM 3380.

Using 4096-byte data CIs, one CA can contain 150 data CIs. If the data CI size were changed to 1024 bytes, the CA could contain 465 data CIs. The sequence set would now require 465 pointers instead of 150.

What IDCAMS Calculates and Adjusts

For a key-sequenced file, after CI size has been set, IDCAMS determines the number of bytes to be reserved for free space, if any. For example, if the CI size is 4096, and the percentage of free space in a CI is twenty, 820 bytes are reserved (4096 x 20% = 820.)

If you do not specify a size for:
- Data CIs, IDCAMS uses 2048, if possible.
- Index CIs, IDCAMS uses 512, if possible.

To determine a suitable index CI size, IDCAMS uses the following formula:

```
(DCI x AES) + (2 x √DCI) + 31

where:

   DCI = number of data CIs per CA

   AES = average entry size:
```

If Key Length is	Then AES is
> 64	28
30–64	3 + (key length/3)
10–29	13
0–9	3 + key length

If the result of the calculation is an odd value, VSE/VSAM rounds it to the next higher even value.

After IDCAMS determines the number of CIs in a CA (see “Control Area (CA) Size” on page 85), it estimates whether one index CI is large enough to handle all of the data CIs in a CA. If the index CI is not large enough, its size is increased, if possible. If not possible, the number of CIs in a CA is decreased. This calculation may result in IDCAMS overriding the specified index CI size. For example, for a file without an index, if CI size space is not specified and the maximum record size is specified to be 200 bytes, IDCAMS sets the data CI size to 2048 bytes. For a key-sequenced file, IDCAMS additionally sets the index CI size to 512 bytes.

If spanning is not specified and the maximum data record size specified in RECORDSIZE is 2500 bytes, and 2500 is also specified for the data CI size, the system adjusts the 2500-byte CI size to the next higher multiple of 512: 2560.

Key Compression

The following information relates to the KEYRANGES parameter/subparameter of the IDCAMS commands DEFINE and IMPORT.

VSE/VSAM increases the number of entries that an index record can hold by key compression. Compression makes an index smaller by reducing the size of the keys in the index entries. VSE/VSAM eliminates from the front and back of a key those characters that are not needed to distinguish it from the adjacent keys. For example, the keys in the sequence 1110, 1230, 1450 would compress to 11, 23, 45 respectively.

Front compression works best when the keys of the last records of every CI run in a series (for example, 100, 101, 102, 106). When several high keys have the same leading characters, those characters can be compressed.

Rear compression works best when adjacent keys have large differences at the back of the key.

If keys compress poorly, more room is required in the index CI to store the compressed key. The index CI may be too small for the data. If it is too small, more CAs are needed. When VSE/VSAM has no more room to insert compressed keys from the data CIs into the index CI, it continues to load data into the next CA, using its associated sequence set CI. The previous CA contain fewer "filled" data CIs than if the index CI had been adequate.

Poor key compression can occur under the following conditions:

- The key is comprised of multiple fields.
- Changes occur in the front of the key and the back of the key, but not in the middle.
- If the number of keys in a group is less than the number of keys in a data CI, the high key in every data CI does not repeat the high-order characters. Therefore, front compression is almost non-existent.
- If the last field of the key is long and very dense, poor rear compression results.

Single field keys *do* compress well. Larger keys (20 - 30 bytes) can compress to 8 or 9 bytes (including control information). Smaller keys (5 - 15 bytes) can compress to 3 - 5 bytes (including control information).

Example of a Key that Compresses Poorly

```
NNN0000000000SS

where: NNN -- changes every 4 or 5 records; there are more
               than 4 or 5 records per data CI.
       0000000000 -- changes rarely.
       SS -- changes in every record.
```

The key would compress well if:

- NNN changed every 20 - 25 records;
- SS is seldom changed;
- SS were located next to NNN (NNNSS0000000000) and changed frequently; or
- The entire key were one field and the bytes changed randomly.

I/O Buffer Space (Using Non-Shared Resources)

VSE/VSAM transmits the contents of a CI to a buffer in main or virtual storage. Therefore, the CI size affects the use and size of I/O buffers, and the amount of storage space for I/O buffers.

If you do not specify buffer space, VSE/VSAM allocates buffer space for two data CIs and (if the file is indexed) one index CI. You may not specify less space, but to optimize performance, you may want to provide additional buffer space.

If you specify a buffer space that is *not* large enough to contain:
Two data CIs, and
One index CI for KSDS and for VRDS,

the DEFINE command terminates.

Considerations

Sequential Processing

Increasing the space to hold three or more data CIs generally improves performance due to I/O command chaining. More than four or five data buffers may cause excessive paging.

If there is an index component, the buffer space must be large enough to hold an index CI also.

Direct Processing

Any remaining buffer space beyond that required for two data CIs is used for index CIs. To optimize performance, specify enough buffer space to accommodate one index CI for every level of index. If the index CI size or the number of index levels is not known, specify 2KB of buffer space for the index (default BUFFERSPACE, which rounds to a 2KB boundary, may in some cases accomplish this for you), and check the result with LISTCAT output. Make adjustments with ALTER, if necessary.

Buffer Specification

You can specify buffer space through the:
- IDCAMS command DEFINE,
- ACB macro, or
- // DLBL statement.

The buffer space entry in the catalog was either specified or defaulted to when the cluster was defined or modified with the ALTER command.

Specifying through DEFINE Command

Using DEFINE, you can specify the BUFFERSPACE parameter at the cluster or data level, but not both. The default buffer space allocation is two data buffers and one index buffer (key-sequenced data sets only). For ESDS and RRDS, the default is two data buffers.

Specifying through ACB Macro

Programming Interface Information

You can specify buffer space values or cause a default buffer space through the ACB macro:

```
ACB   .
      .
      .
      BUFSP=n
      BUFNI=n
      BUFND=n
```

To use the ACB buffer space, the value selected must be larger than the catalog entry buffer space. The use of ACB parameters is explained under “Buffer Allocation” on page 96.

End of Programming Interface Information

Specifying through // DLBL

At run time, you may require more than the buffer space specified in the catalog or ACB. The minimum requirements for run time buffers are as follows (default STRNO=1):

```
Data buffers  = ACB STRNO + 1
Index buffers = ACB STRNO
```

If STRNO = 2 (that is, you require concurrent file positioning), the minimum buffer space required for output is three data CIs and two index CIs.

Example: Specifying Buffer Space: You can specify buffer space through the use of the // DLBL statement:

```
// DLBL filename,'file-ID',,VSAM,BUFSP=size
```

To be effective, the value specified for the DLBL buffer space must be larger than the catalog entry buffer space.

VSE/VSAM rounds the buffer space value (obtained from the DLBL, ACB, or DEFINE) so that it is a multiple of either the index CI size or the data CI size, whichever is smaller.

If the amount of buffer space specified is greater than the minimum required, VSE/VSAM uses the remainder for additional index buffers (direct processing) or additional data buffers (sequential or skip sequential processing).

Example: Specifying Number of Buffers: You can specify the number of buffers through the use of the // DLBL statement:

```
// DLBL filename,'file-ID',,VSAM,BUFND=m,BUFNI=n
```

Note that BUFND and BUFNI represent the *total* number of buffers, independent of the number of strings. That is, if the value for BUFND, respectively BUFNI, is lower than the required minimum, the default values are used.

Buffer Allocation

The following explains how VSE/VSAM allocates buffer space according to ACB specification. The following ACB parameters relate to buffer allocation:

```
ACB  MACRF=(IN|OUT,SEQ|DIR|SKP)
     STRNO=n
     BUFSP=n
     BUFND=n
     BUFNI=n
```

Minimum Buffer Allocation

Data Buffers: If you specify:

```
MACRF=(...,IN,...)
```

then, the number of data buffers for:
ESDS and RRDS is the greater of BUFND or STRNO.
KSDS is the greater of BUFND or STRNO + 1.

If you specify:

```
MACRF=(...,OUT,...)
```

then, the number of data buffers is the greater of BUFND or STRNO + 1.

Index Buffers: If the number of index buffers is the greater of BUFNI or STRNO, then OPEN calculates the remainder as follows:

```
Remainder = BUFSP - ((NDB*DCI) + (NIB*ICI))

   where: NDB = number of data buffers
          DCI = size of a data CI
          NIB = number of index buffers
          ICI = size of an index CI
```

If the remainder ≤ 0, then OPEN allocates the number of data buffers and index buffers and increases BUFSP to hold them.
If the remainder > 0, and to calculate additional buffers, refer to "If Remainder > 0," below.

Note that you get no indication if the BUFSP used for the minimum allocation is greater than that specified in DEFINE, DLBL, or ACB.

If Remainder > 0:

1. MACRF=(...,SEQ,OUT,...)

   VSE/VSAM allocates data buffers until there is a remainder that is less than the data CI size; then it allocates more index buffers. (This is only possible when the index CI size is less than the data CI size. If the index CI size is larger, see item 2 below.)

   Example:

   ```
   BUFSP=13824
   data CI size=4096
   index CI size=512
   STRNO=1
   MACRF=(...,SEQ,OUT,...)
   ```

Allocation			Cumulative Totals
Minimum	= 2 data buffers	8192	
	1 index buffer	512	8704
Additional	= 1 data buffer	4096	12800
	2 index buffers	1024 *	13824
* Resulting from MACRF specification.			

2. MACRF=(...,DIR,OUT,...)

   VSE/VSAM allocates more index buffers until there is a remainder that is less than the size of one index CI; then it allocates more data buffers. (This is possible only when the data CI size is less than the index CI size.)

   Example:

   ```
   BUFSP=13824
   data CI size=4096
   index CI size=512
   STRNO=1
   MACRF=(...,DIR,OUT,...)
   ```

```
Allocation                                  Cumulative
                                              Totals

Minimum     = 2 data buffers    8192
              1 index buffer     512          8704
Additional  = 10 index buffers  5120 *       13824

* Resulting from MACRF specification..
```

3. MACRF=(...,SEQ,DIR,OUT,...)

   VSE/VSAM increases the number of index buffers to twice STRNO. (If this is not possible, VSE/VSAM uses the procedure described in item 2 above.) If there is still a remainder, VSE/VSAM uses the procedure described in item 1 above to allocate the remainder.

   Example:

```
BUFSP=13824
data CI size=4096
index CI size=512
STRNO=1
MACRF=(...,SEQ,DIR,OUT,...)
```

```
Allocation                                  Cumulative
                                              Totals

Minimum     = 2 data buffers    8192
              1 index buffer     512          8704
Additional* = 1 index buffer     512          9216
            = 1 data buffer     4096 *       13312
            = 1 index buffer     512 *       13824

* Resulting from MACRF specification.
```

   Later modifications of RPLs do not change buffer allocations.

Buffer Allocation for a Path

Path Entry for Alternate Index (AIX): If the path entry is not a member of the upgrade set, buffers are allocated in the same manner as for a normal KSDS. Your ACB is used for the path entry.

If the path entry is a member of the upgrade set, then buffers are allocated as for a normal KSDS, but minimum allocations are increased by one for both the number of data buffers and the number of index buffers. Your ACB is used for the path entry.

Buffer Allocation for Path Entry when the Base Cluster is a KSDS: Buffers are allocated in the same manner as for a normal KSDS with the following ACB specifications:

```
BUFND=0
BUFNI=0
STRNO=number of strings specified in the ACB
```

You can influence buffer allocation only through the BUFFERSPACE parameter of DEFINE CLUSTER or through DLBL BUFSP= ,BUFND= ,BUFNI=.

If you open the path for *input* only, the base cluster uses MACRF=(...,DIR,IN,...). If you open the path for *output*, the base cluster uses MACRF=(...,DIR,OUT,...).

Buffer Allocation for a Base Cluster of an Alternate Index: You can influence buffer allocation through the path DLBL BUFND=, BUFNI=. If the base cluster is a KSDS, the minimum index buffer allocation is one buffer per index level per string.

Buffer Allocation for an Upgrade Set: The buffer allocation is always two data buffers and one index buffer. You cannot influence buffer allocation for the upgrade set.

Miscellaneous Notes on Buffer Allocation (NSR)

- Data and index buffers are acquired and allocated only at OPEN time. Buffer space is freed at CLOSE time.
- Buffer space is aligned on page boundaries. Data buffers are allocated first, then the index buffers.
- Writing a buffer does not free buffer space. The CI is still in storage, so if you again reference that CI, VSE/VSAM does not reread the CI. Because VSE/VSAM checks to see if the CI is in storage, processing directly in a limited key range may increase throughput if extra data buffers are provided.
- The POINT macro does not cause read ahead because its purpose is to position for subsequent sequential retrieval. It fills only one data buffer.
- When processing *directly*, VSE/VSAM reads only one data CI. It does not reread data or index CIs if they reside in storage, except when SHAREOPTIONS(4) is specified. VSE/VSAM will immediately write a data buffer if PUT (UPD,DIR) or PUT (NUP,DIR) was issued. VSE/VSAM will write immediately for a sequential PUT if PUT (SEQ) follows GET (DIR) for the same RBA.
- Although VSE/VSAM does not read index buffers ahead, the effect is similar. Index buffers are loaded when referenced. If multiple index buffers are provided, index CIs are not reread because there is room for the CIs in storage. VSE/VSAM reuses buffers on a least-recently-used basis.
- For SHAREOPTIONS(4) processing, VSE/VSAM usually reads data and sequence-set CIs on every request. Exceptions are:
  - Consecutive retrievals, not for update, from the same CI do not cause a reread in sequential or skip-sequential mode.
  - Consecutive inserts or retrievals for update, in sequential or skip-sequential mode, do not cause rereads, unless the SHAREOPTIONS(4) lock has been held for a period longer than about 0.5 seconds. (The SHAREOPTIONS(4) lock is for a CA.)

  High-level index CIs are not reread unless they are out of date.

  Read-ahead is not done under SHAREOPTIONS(4); therefore extra data buffers are of no benefit.

I/O Buffer Space (Using Local Shared Resources)

Using the Shared Resources facility of VSE/VSAM, you can manage I/O buffers. This includes:

- Deferring write operations for direct PUT requests.
- Correlating deferred requests by transaction ID.
- Writing out buffers whose writing has been deferred.

For more information, refer to “Sharing Resources Among Files and Displaying Catalog Information” on page 191.

Miscellaneous Notes on Buffer Allocation (LSR)

- Read-ahead is not done under LSR. Therefore, extra buffers are of no advantage.

- Writing a buffer does not free buffer space. The CI is still in storage, so if you again reference that CI, VSE/VSAM does not reread the CI. Because VSE/VSAM checks to see if the CI is in storage, processing directly in a limited key range may increase throughput if extra data buffers are provided.
- When processing *directly*, VSE/VSAM reads only one data CI. It does not reread data or index CIs if they reside in storage, except when SHAREOPTIONS(4) is specified. VSE/VSAM will immediately write a data buffer if PUT (UPD,DIR) or PUT (NUP,DIR) was issued. VSE/VSAM will write immediately for a sequential PUT if PUT (SEQ) follows GET (DIR) for the same RBA.
- Although VSE/VSAM does not read index buffers ahead, the effect is similar. Index buffers are loaded when referenced. If multiple index buffers are provided, index CIs are not reread because there is room for the CIs in storage. VSE/VSAM reuses buffers on a least-recently-used basis.
- For SHAREOPTIONS(4) processing, VSE/VSAM usually reads data and sequence-set CIs on every request. Exceptions are:
  - Consecutive retrievals, not for update, from the same CI do not cause a reread in sequential or skip-sequential mode.
  - Consecutive inserts or retrievals for update, in sequential or skip-sequential mode, do not cause rereads, unless the SHAREOPTIONS(4) lock has been held for a period longer than about 0.5 seconds. (The SHAREOPTIONS(4) lock is for a CA.)

  High-level index CIs are not reread unless they are out of date.

LSR Buffer Hashing

Large VSAM LSR buffer pools can improve response time and reduce I/O operations. However, searching the pool to find the right buffer takes time. Benefits were often reduced due to the increased CPU time needed to search large buffer pools. To overcome this reduction in performance, VSAM buffer hashing has been implemented, in which a VSAM hashing algorithm allows direct access to the required buffer. Using VSAM buffer hashing, you can take advantage of very large buffer pools without the disadvantage of additional processor load. VSAM buffer hashing is a function introduced with VSE/ESA 2.5. This buffer management technique provides the following improvements over the existing sequential buffer management:

- The time required to perform buffer searches is reduced, since the need to do sequential searches through the buffer pool is removed. The search technique uses a hashing algorithm. Using this hashing algorithm, the path length of the search is significantly shortened. The I/O rate is therefore reduced.
- The path length does not depend upon the number of buffers (therefore the search time is independent of the buffer pool size).

How Does Buffer Hashing Work?

Using VSAM buffer hashing, you can take advantage of using very large buffer pools, without the disadvantage of additional processor load.

VSAM Buffer Hashing uses a:

- Hash Table – A table in main storage in which each table entry is used as a pointer to a BCB (Buffer Control Block). A BCB contains the address of the buffer (for data or index), and information about the buffer itself. There is one BCB for each buffer in the LSR buffer pool.
- Synonym – When using a hashing technique, synonyms may occur when two or more entities hash to the same anchor point. In VSAM Buffer Hashing,

synonyms are chained together in the BCB. However, the possibility that synonyms occur is very small, and the chain is usually very short.

- Hash Algorithm, which is calculated as follows:

  X = remainder of (RBA/2 + DSID1/2 + DSID2/2) / DIM

  where:

  X The remainder of the above calculation, and is used as the index to the hash table.

  RBA The Relative Byte Address, used by VSAM to identify a certain buffer in the LSR buffer pool.

  DSID1 and DSID2
  The Data Set Identifiers (DSIs), which are unique identifications of a certain opened VSAM data set component, either a data or an index component.

  DIM The number of entries in the Hash Table. DIM = (2N-1).

  N The number of buffers in the subpool.

Here is a ″simple″ example:

″Simple″ in this case means that the values of this example were simplified to decimal values (not hexadecimal) to give a better understanding of the technique.

1. Let us assume that we have an LSR pool with 10 buffers. The Hash Table will have (2 * 10 -1) = 19 entries. Therefore:

   DIM = 19

2. A VSAM GET operation reads a data record from a certain VSAM data set with the internal data set identifications DSD1 and DSD2 into a data buffer. Therefore:

   DSID1 = 220, DSID2 = 32

   The BCB pointing to that data buffer is at storage location ’640000’. The RBA (Relative Byte Address) of the VSAM data buffer is 800. Therefore:

   RBA = 800

3. The hash algorithm `X = remainder of (RBA/2 + DSID1/2 + DSID2/2) / DIM` therefore calculates the following index for the hash table:

   (800/2 + 220/2 + 32/2) /19 = 27, **remainder = 13 = X**

   ″**13**″ will be used as index into the hash table.

4. The BCB pointer **’640000’** will be stored in the **13th position** of the hash table.

5. Whenever another request is searching for a data buffer with RBA **800** from this certain dataset, the hash algorithm can calculate easily the index of **13** into the hash table and use the BCB at address **’640000’** and its related data buffer without a long pool search. This hashing technique also works, of course, with very large buffer pools (for example, 32767 buffers).

Preventing Deadlock in Buffer Contention

Contention for VSE/VSAM data (the contents of a CI) can lead to deadlocks, in which a processing program is stopped because its request for data cannot be satisfied.

Your processing program gets exclusive control of a buffer (CI) whenever you issue a GET for update (RPL option OPTCD=UPD) to retrieve a record from that buffer.

You are responsible for preventing a deadlock by releasing as soon as possible the buffer for which another request may be waiting. Two requests, for example, A and B, may engage in four different contests:

1. A wants exclusive control, but B has exclusive control (OPTCD=UPD). VSE/VSAM refuses A's request. A must either do without the data or retry its request.
2. A wants exclusive control, but B has read-only access to the data (OPTCD=NUP). VSE/VSAM gives A a separate copy of the data.
3. A wants read-only access to the data (NUP), but B has exclusive control. VSE/VSAM refuses A's request. A must either do without the data or retry its request.
4. A has read-only access to the data, and B has read-only access. VSE/VSAM gives A a separate copy of the data.

VSE/VSAM's action in a contest for data rests on the assumptions that, if a processing program has exclusive control of the data (OPTCD=UPD), it will (or at least might) update or delete it and that, if a processing program is updating or deleting the data, it has exclusive control of it.

In contests 1 and 3, B is responsible for giving up exclusive control of a CI by way of an ENDREQ or a request for access to a different CI. (The RPL that defines the ENDREQ or request is the one that was used to acquire exclusive control in the first place.)

Multiple Volume Support

Key Ranges

The records of a key-sequenced file, including alternate indexes, can be grouped on volumes according to key ranges. A payroll file, for example, could have employee records beginning with A, B, C, and D on one volume, with E, F, G, H, and I on a second volume, and so on. Every portion of a multivolume file can be on a separate volume. Every key range of a file, as well as the end of the file, is preformatted. Multiple volume support is affected by the following DEFINE parameters: VOLUMES, ORDERED|UNORDERED, CYLINDERS|RECORDS|TRACKS|BLOCKS, and KEYRANGES.

The first allocation made on every volume is always the primary allocation. To place the index and data on separate volumes, specify the VOLUMES parameter for both data and index components.

Your CLASS specification in the DEFINE command can affect suballocation. For further information, see "Data Space Classification" on page 83.

Space Allocation

Space Allocation without Key Range Specified

Primary space is acquired from the first volume at define time. If VSE/VSAM needs more space during loading or processing of the file, and if secondary allocation was specified, VSE/VSAM uses the secondary extents on the first volume. When VSE/VSAM has acquired all the secondary space it can on the first volume and still needs more space, then primary space from the second volume is acquired, even if no secondary allocation was specified. If more space is needed, secondary space is acquired on the second volume.

Space Allocation with Key Range Specified

Primary space is acquired from every volume at define time. Every key range is assigned to a volume. There is a primary allocation for every key range. If there are fewer volumes than key ranges, the extra key ranges are grouped together on the last volume. If there are more volumes than the number of key ranges, the excess volumes become overflow volumes. A key range is associated with the primary allocation volume and can extend to any overflow volumes.

A key range is extended first by acquiring secondary extents on its volume of primary allocation, next by acquiring primary allocation on the first overflow volume, then secondary extents on the first overflow volume. Primary allocation is then acquired on the second overflow volume, followed by acquiring secondary extents on the second overflow volume. If there is not enough room on an overflow volume to acquire primary space for that key range, VSE/VSAM does not acquire any secondary space for that key range. VSE/VSAM just skips that overflow volume and goes to the next overflow volume to try to obtain primary space.

VSE/VSAM searches for space on volumes in the order they were specified in the VOLUMES parameter. This does not mean that the volumes are allocated or suballocated in that order. Allocation depends on whether ORDERED or UNORDERED was specified.

Unordered Space Allocation

If no Key Range was Specified: UNORDERED means VSE/VSAM must find a primary allocation (or the DEFINE command will fail), but not necessarily on the first volume listed in the VOLUMES parameter. If there is no room for a primary allocation on the first volume, successive volumes are checked for primary space.

If Key Range was Specified: UNORDERED means that VSE/VSAM must find room for a primary allocation for every key range, but not necessarily the first key range on the first volume, the second key range on the second volume, and so on.

Ordered Space Allocation

ORDERED means VSE/VSAM must suballocate space on the volumes in the order in which the volumes are listed in the VOLUMES parameter.

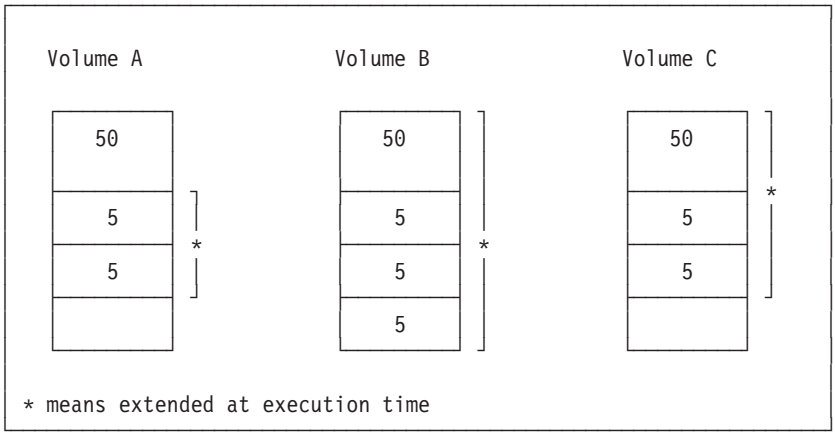
If secondary allocation is specified, space for a component can be expanded to include a maximum of 123 extents. Every primary and every secondary allocation can be made up of up to five non-contiguous areas (extents).

Examples: Allocation of Space on Multiple Volumes

The following examples show various combinations of ORDERED and UNORDERED space allocation, VOLUMES, and primary versus secondary allocations.

Example 1

```
VOLUMES(A B C)
ORDERED
CYLINDERS(50 5)
SUBALLOCATION
```



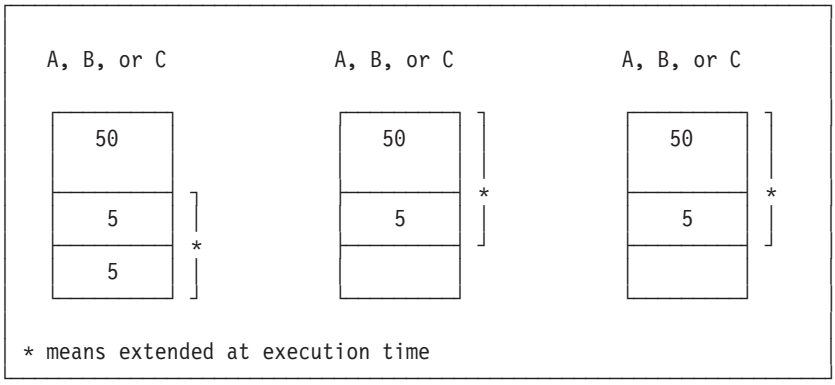
Volume A is the primary volume; volumes B and C are overflow volumes. Fifty cylinders of primary space must be available on volume A, or the DEFINE command will fail.

If the file is extended, a 5-cylinder secondary allocation is made on volume A, providing volume A has enough available VSE/VSAM space of the required class. Otherwise, an allocation of 50 cylinders (primary amount) is made on volume B. If volume B does not have enough data space for this allocation, the request for extension is rejected.

If volume B has 50 cylinders for allocation (primary amount) and the file needs to be extended further, secondary allocations are made from volume B. Volume B must have enough space available of the required class. Otherwise, a 50-cylinder allocation is made on volume C.

Example 2

```
VOLUMES(A B C)
UNORDERED
CYLINDERS(50 5)
SUBALLOCATION
```



Fifty cylinders of primary allocation must be made on one volume. It may be volume A, B, or C. If it is not possible to allocate all 50 cylinders a single volume, the DEFINE fails.

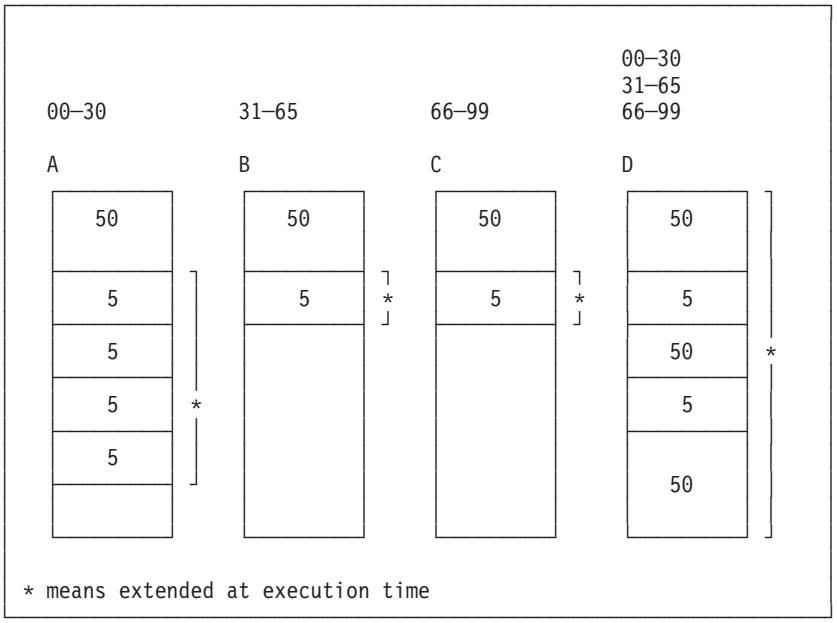
Volumes are searched in the order they are specified. If both A and B have 50 cylinders available, allocation is made on A because it was specified first.

When the file is extended, VSE/VSAM attempts to make the 5-cylinder secondary allocations on the same volume the primary allocation was made on. This continues until all data space of the required class is used.

To further extend the file, VSE/VSAM searches the volumes for space in the same order specified for primary allocation. If VSE/VSAM cannot acquire the primary amount of space (50 cylinders), an error code is issued.

Example 3

```
VOLUMES(A B C)
KEYRANGES((00 30) (31 65) (66 99))
ORDERED
CYLINDERS(50 5)
SUBALLOCATION
```

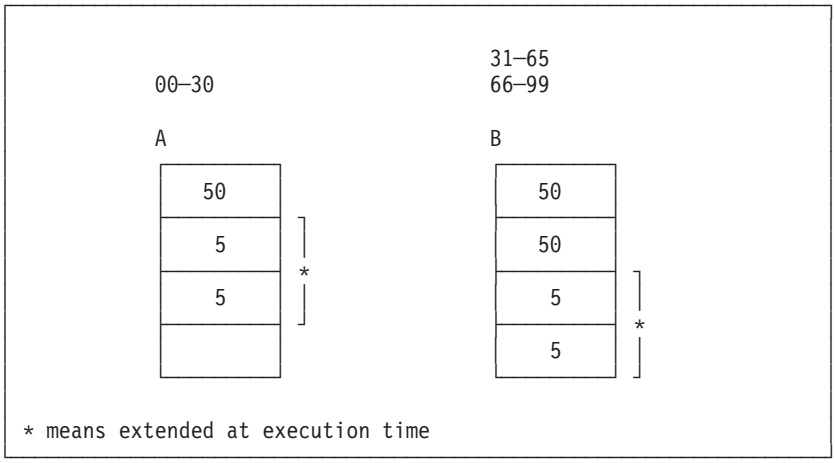


A primary allocation of 50 cylinders is made for every key range. The first key range is on volume A, the second on volume B, the third on volume C. If 50 cylinders cannot be allocated on every volume, the DEFINE fails. The 5-cylinder secondary allocations are made as needed.

A key range can be extended only on the volume it occupies or on an overflow volume. If volume D were added to the VOLUMES list, all key ranges would be extended on volume D if the volume initially assigned to the key range became full: first a primary allocation amount of 50 cylinders for a key range on volume D, then secondary allocations of 5 cylinders.

Example 4

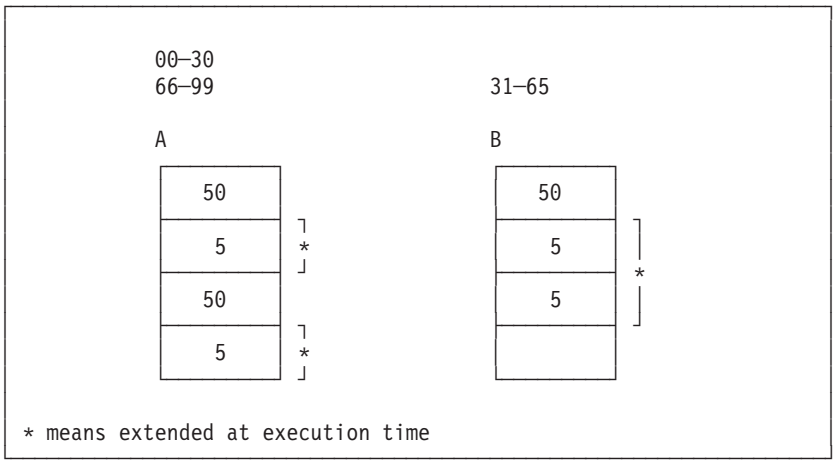
```
VOLUMES(A B)
KEYRANGES((00 30) (31 65) (66 99))
ORDERED
CYLINDERS(50 5)
SUBALLOCATION
```



If only volumes A and B are specified, the first key range is allocated on volume A, and the second and third key ranges are allocated on volume B. Volume A has one 50-cylinder primary allocation, and volume B has two 50-cylinder primary allocations. This can occur only for a file with the SUBALLOCATION attribute specified. If both UNIQUE and KEYRANGES are specified, every key range must reside on a separate volume.

Example 5

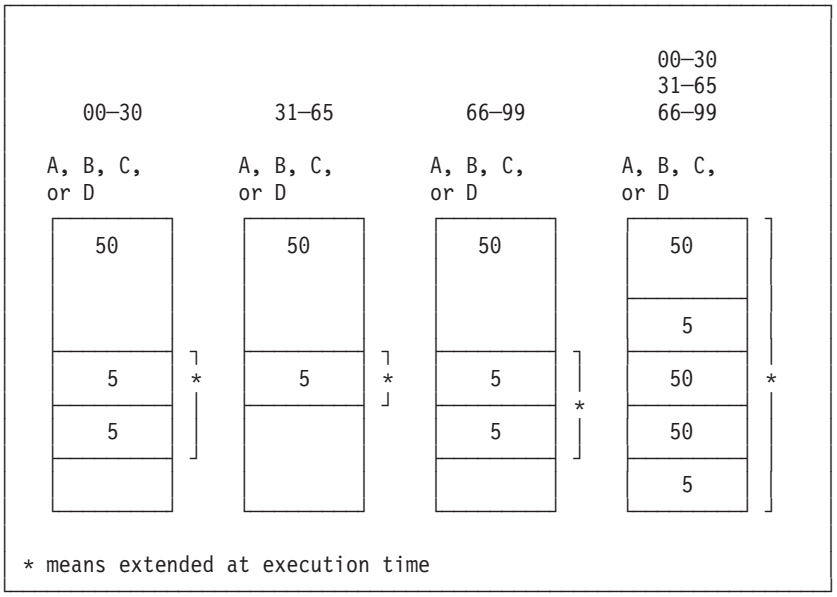
```
VOLUMES(A B A)
KEYRANGES((00 30) (31 65) (66 99))
ORDERED
CYLINDERS(50 5)
SUBALLOCATION
```



A primary allocation of 50 cylinders is made for every key range. The second key range is on volume B; the first and third key ranges are on volume A. This can occur only for a file with the SUBALLOCATION attributed specified. If both UNIQUE and KEYRANGES are specified, every key range must reside on a separate volume.

Example 6

```
VOLUMES(A B C)
KEYRANGES((00 30) (31 65) (66 99))
UNORDERED
CYLINDERS(50 5)
SUBALLOCATION
```



A primary allocation of 50 cylinders is made for every key range. VSE/VSAM attempts to put one key range on every volume. If volume A does not have 50 cylinders available, the first key range is put on volume B, and the second and third key ranges are put on volume C. If neither A nor B has 50 cylinders, all three key ranges are placed on volume C.

VSE/VSAM first extends a key range on the volume it is on before trying to extend it on any overflow volume. If volume D were added to the VOLUMES list, every key range would be extended on volume D, if no more spaces were available on the volume of its primary allocation.

If volume D were listed in the VOLUMES parameter, it would not necessarily be an overflow volume. If 50 cylinders of primary allocation were available on A, B, and C, then D would be an overflow volume. If volume A does not have 50 cylinders available, but B, C, and D have 50 cylinders each, the first key range is put on volume B, the second on volume C, and the third on volume D. Volume A becomes the overflow volume.

An Exercise

Assume that you have a 600-cylinder file that you want to have reside on two volumes: 400 cylinders on volume A, and 200 cylinders on volume B. How would you specify this allocation requirement in the DEFINE command?

Do **not** specify:

```
VOL(A B)
CYL(600)
```

This request would be rejected because the amount of primary space to be allocated on every volume is greater than that available on one volume.

Do **not** specify:

```
VOL(A B)
CYL(400,200)
```

This request would obtain 400 cylinders of primary allocation on volume A and 400 cylinders of primary allocation on volume B.

Do specify:

```
VOL(A B)
CYL(200,200)
```

This request obtains:
200 cylinders primary allocation on volume A,
200 cylinders secondary allocation on volume A, and
200 cylinders primary allocation on volume B.

The mounting requirements with multiple volumes are simple. All volumes must be mounted (except with sequential KSDS, ESDS, and RRDS). A primary allocation amount will be acquired on *every* volume.

Space Allocation

Possible Options

The CYLINDERS|RECORDS|TRACKS|BLOCKS parameters of the DEFINE command determine how VSE/VSAM allocates space. You may specify allocation at the CLUSTER/AIX level, DATA level, DATA and INDEX levels, and CLUSTER/AIX and DATA levels. Considerations in choosing allocation parameters are:

- If you specify allocation at the CLUSTER/AIX level only, the amount needed for the index is subtracted from the specified amount. The remainder of the specified amount is assigned to data.
- If you specify allocation 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.
- If you specify allocation 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.
- If you specify *secondary* allocation at the DATA level, secondary allocation must be specified at the INDEX level unless you specify allocation at the CLUSTER level.
- A CA can never cross an extent boundary. A cluster extent consists of a whole number of CAs.
- A CA is never larger than one cylinder (CKD) or one maximum CA (FBA). Optimum performance is obtained when an integral number of CAs occupy a cylinder (or maximum CA).
- IDCAMS checks the smaller of primary and secondary space allocation values against the specified device's cylinder (or maximum CA for FBA devices) size. If the smaller quantity is greater than the device's cylinder (or maximum CA) size, the CA is set equal to the cylinder (or maximum CA) size. If the smaller quantity is less than or equal to the device's cylinder (or maximum CA) size, the

size of the CA is set equal to the smaller space quantity. For FBA, this value is then rounded up to a multiple of minimum CA size.

For example:

```
CYL(5 10)     results in a 1-cylinder CA
TRK(100 3)    results in a 3-track CA
REC(2000 5)   results in a 1-track CA (assuming 10 records
              per track - minimum CA is 1 track)
TRK(3 100)    results in a 3-track CA
```

For a device with 64 blocks per minimum CA and 960 blocks per maximum CA:

```
BLK(1100 1000)  results in a 960-block CA
BLK(900 400)    results in a 448-block CA
BLK(100 40)     results in a  64-block CA
```

For CKD to force IDCAMS to select cylinder CAs, specify CYLINDERS or UNIQUE. When defining through the RECORDS|TRACKS parameters, specify the smaller of primary or secondary allocation as a value of at least one cylinder.

- If you specify secondary allocation, space for a component can be expanded to a maximum of 123 extents (if there is sufficient data space) with a limit of 16 extents per volume if REUSE is specified.
- A UNIQUE file can have a maximum of 16 extents per volume, but it can not be extended; no secondary allocations are permitted for UNIQUE files.
- A spanned record cannot be longer than a CA minus the control information (10 bytes per CI). Do not specify large spanned records with small primary or secondary allocation.
- VSE/VSAM acquires space in increments of CAs. For example, if the allocation amount is 20 tracks and the device is an IBM 3380, the CA size is one cylinder. Two cylinders of space (two CAs) are allocated, because a 3380 has 15 tracks per cylinder.
- LISTCAT gives information in increments of CA size. If you specify either TRACKS or RECORDS and the allocation is less than one cylinder, LISTCAT reflects the allocation as TRACKS. If the specification results in a one-cylinder CA, LISTCAT reflects the allocation as CYLINDERS. If you specify BLOCKS, the allocation is given in multiples of blocks.

NOALLOCATION

NOALLOCATION allows you to define a file into a catalog without suballocating any space to it. This parameter can be useful in two ways:

- Creating default models. (For a discussion of default models, see “Using an Object as a Model” on page 57.)
- Creating *dynamic* files for which space is not actually suballocated until the file is opened.

Dynamic Files

Formerly, files that were used for brief periods of time (for example, work files) occupied disk space from the time they were defined until they were deleted. If they were required again, they had to be redefined.

Using the DEFINE CLUSTER command with NOALLOCATION and REUSE parameters makes it possible to define a file for which no space is suballocated until the file is to be opened; this file is called a *dynamic* file. The catalog entry for a dynamic file contains only the allocation *size* specified at define. Information about the suballocated space is added to the catalog when the file is opened.

When you try to delete a dynamic file, VSE/VSAM determines if space is currently allocated to it. If it is, VSE/VSAM deletes it as if it were a normal VSE/VSAM cluster. If space is not allocated, only the catalog entry of the file is removed.

Dynamic files may be entry-sequenced (including SAM ESDS supported by the *VSE/VSAM Space Management for SAM Function*), key-sequenced, or relative-record files.

Dynamic File Restrictions: The following restrictions apply to dynamic files:

- A path (but not an alternate index) may be built over a dynamic file, except for a SAM ESDS file.
- A dynamic file that does not have space allocated to it cannot be printed (PRINT), copied (REPRO), or exported through EXPORT. EXPORT only supports non-empty dynamic files
- A default model cannot be opened. If you specify NOALLOCATION, you must also specify REUSE if you plan to open the file.
- Normally, parameters such as CYLINDERS, TRACKS, and USECLASS control space allocation. However, for noallocation models (other than reusable files), these attributes are recorded only for modeling purposes.
- If you specify the VOLUMES parameter when you define a file as NOALLOCATION, VSE/VSAM records those volumes in the catalog as candidate volumes.
- The NOALLOCATION attribute exists in the catalog entry, but it cannot be implicitly modeled. It can be explicitly modeled (MODEL parameter of DEFINE).
- You cannot specify NOALLOCATION on the ALTER command.
- You cannot ALTER REMOVEVOLUMES for the last existing volume on the candidate list for NOALLOCATION files.

Data Protection and Integrity Options

When considering performance, you must also consider the data protection and integrity options you are using. VSE/VSAM performance is affected by the following:

- Share options.

  For more information, see “Protecting Shared Data” on page 129.
- Write check.

  If you specify WRITECHECK in the DEFINE command, it means you wish to have your records checked as they are written. After a record is written, it is then read without data transfer to test for a data check condition. If applicable, VSE/VSAM uses the bypass cache option when writing to write check files. If NOWRITECHECK is specified (and this is the default), a record is written but no checking occurs. That is, you will get better performance with the NOWRITECHECK option.
- Recovery versus Speed.

  The RECOVERY and SPEED parameters in the DEFINE command control the preformatting of CAs before records are inserted. RECOVERY|SPEED applies only to initial loading. Specifying RECOVERY means that space allocated to the data component is preformatted. Specifying SPEED means that space will *not* be preformatted.

  Specifying SPEED gives you better performance, whereas specifying RECOVERY enables you to recover from certain system failures.

  Consider the following:

- If you specify SPEED in a file's DEFINE command, and a system failure occurs, the file must be deleted, redefined, and reloaded. RECOVERY is only useful if you have a recovery procedure that allows you to resume loading the file after a system failure. RECOVERY formats every CA before loading records into it. It allows you to find the software end-of-file if an abnormal termination occurs during initial creation. After the initial creation of the file, RECOVERY is always in effect.
- RECOVERY works in conjunction with the IDCAMS VERIFY command. If a system failure occurs before a file is closed (CLOSE or TCLOSE), VERIFY can prevent your having to reload the file by updating the catalog with the current high RBA. This ensures that your data will not be overwritten inadvertently at a later time, and that you may continue the load at the point of interruption (load-extend). If the SPEED option was in effect while the file was loaded, VERIFY cannot help because no preformatting was done and no high RBA exists until the file is closed.

Distributed Free Space

Free space can occur in files as a result of:

- your FREESPACE specifications in the commands DEFINE ALTERNATEINDEX and DEFINE CLUSTER, or
- CI/CA splits.

For more information and examples on CI and CA splits that result from record inserts during direct and sequential processing, refer to "CI/CA Splits" on page 114.

You can specify free space only for key-sequenced data sets (KSDSs), variable-length relative-record data sets (VRDSs), or alternate indexes. The CI free space should be as large as the design insertion level. Determine the free space required by estimating the percentage of additions to be made between file reorganizations. Also, consider the size of your records. If there are to be no additions, or if records will not be lengthened, there is no need for free space.

Loading a File

Specifying Free Space

You specify free space for both the CI and the CA as a percentage of the total space for the respective unit. For example:

```
FREESPACE (20 10)
```

indicates that:
20% of every CI is to be initially empty, and
10% of every CA is to be initially empty.

If you specify the minimum CA free space of 1%, free space for one CI in every CA will be provided. The system default for free space is (0 0).

Altering Free Space

You can change the free space after the file is loaded. To take full advantage of mass insertion, specify FREESPACE(0 0) in the ALTER command after the file has been loaded.

Considerations for Loading a File

- If additions occur only in a specific part of the file:

Load those parts that will have no additions with a free space specification of (0 0). Alter the free space to (n n) to load those parts of the file that will receive the additions.

If SPEED is specified, it is in effect for loading the initial portion only. Any subsequent portions are loaded with RECOVERY, regardless of the DEFINE specification.

- If additions occur throughout the file, but are unevenly distributed:

  Specify a small amount of free space when you define the file. Then, increase the percentage after loading the file. As new CIs and CAs are required, they are created with the increased free space specification.

  Additional splits (after the first split) in the part of the file with the most growth will be minimized. CIs that have little or no growth will contain only a small amount of unneeded free space.
- If there are few additions to the file:

  Consider a free space specification of (0 0) for loading the file and subsequent processing.

  When records are added, new CAs are created to provide room for additional insertions. In this case, unused free space is not provided.
- For direct insertions:

  Make the CI free space larger than the CA free space, unless the frequency of insertions is very low. In that case, zero CI free space and average CA free space might be indicated.
- For sequential processing:

  The greater the free space specification, the more disk space is required.

  For *sequential* processing, more I/O operations (with more system overhead) are required to process the same number of records. A bad combination of CI-size/record-size/free-space can cause poor sequential performance if much of the free space is unusable.

Performance with Too Much or Too Little Free Space

Too much free space could increase the number of index levels, which could increase run time for *direct* processing.

Too little free space can cause an excess of (time-consuming) CI/CA splits:

- For *sequential* processing - and after a split occurred - extra time is required because the records are not in physical sequence.
- For *direct* processing, CA splits can increase seek time.

Another factor is the additional VSE/VSAM overhead to do the split. (If insertions are truly random, ideally all CAs would split at approximately the same time.) It is recommended to monitor CA splits and to reorganize the file when the splits become prevalent. To monitor CA splits, use LISTCAT or the ACB JRNAD exit.

Where VSE/VSAM Places the Records

Records are loaded or mass inserted at the end of a CI until the free space threshold is passed. The free space threshold is the point at which free space becomes less than the amount specified in the DEFINE command.

VSE/VSAM ensures that at least one record (or one segment of a spanned record) is placed into a CI. Also, if the CA free space specified in the DEFINE command is not zero but is less than one CI, the result is one free CI in the CA.

Specifying Free Space in a CI and CA

You specify free space in a CI as a percentage of bytes in the CI. Generally you should specify a value equal to at least one record, because VSE/VSAM does not round up the free space to hold a whole number of records.

The amount of *unused space* in the CI, however, may be more than the *free space* you requested. For example:

> If you specify (33 0) free space, you are in effect telling VSE/VSAM to put as many records as possible into 67% of the CI. If the CI can contain four logical records, the first two records will fit into 50% of the CI. This leaves 17% unallocated space. The unallocated space is added to the 33% free space, for a total of 50% free space available for allocation. In this case, where the amount of unused space is greater than the amount of requested free space, the value you requested is stored in the catalog.

For this same CI, if you specify (25 0) free space, the CI would contain three logical records and 25% free space. If (20 0) free space is specified, the result is three logical records and 25% free space. If (80 0) free space is specified, the result is one logical record and 75% free space. Specifying *free space in a CA* is somewhat different. If you specify any value greater than zero, VSE/VSAM will round up the value so that you get at least 1 free CI per CA. As in CI free space allocation, however, you may get additional space due to the combination of requested free space plus unallocated space.

Notes:

1. Remember that a CI contains logical records, free space, and control information (CIDF and RDF). A 4KB CI cannot contain four 1KB logical records. A 4KB CI with (25 0) free space specified will contain at least 1KB of free space; only two 1KB fixed length logical records could be loaded into the CI. Only one more 1KB logical record could be added before a CI and/or CA split would be required.
2. If you specify FREESPACE(100 100), the CIs and CAs are not left empty. VSE/VSAM writes one record per CI and one CI per CA; the rest is free space.
3. If ten CIs fit into one CA and (0 5) free space is specified, the CA will have one free CI.

Reclaiming Space

You can use the ERASE macro to delete records. The space that was occupied by the deleted record is returned to the free space.

If a CI is emptied by ERASE, it can be reclaimed later as a free CI if it is needed.

Note: Space that becomes free within a CI because of records deleted or shortened may remain unused even though the space is available. This situation occurs when new records to be added to the file do not have key-field values that match the range of the freed area within the CI. For example:

> A record with key-field value 250 cannot be inserted between records with key-field values 22 and 70.

Depending on the amount of unusable space, you may want to reorganize the file (using EXPORT and IMPORT) to make the available free space useable.

The same problem can exist on a CA level. If all records in all CIs (in one CA) are deleted, the CA is not reused unless space is required in its key range. To reclaim unused CAs, use BACKUP and RESTORE to reorganize the file.

CI/CA Splits

The following explains the rules for CI and CA splits.

Sequential Processing

CI Split: If the insert is in the middle of the CI, the records with higher keys are moved to the free CI. The insert and the records with lower keys remain in the old CI. If the insert is at the logical end of the CI, the inserted record goes to the free CI.

CA Split: If the insert is *not* in the last logical CI, all CIs after the split CI are moved to the new CA. If the insert is within the last logical CI, that CI is moved to the new CA. If the insert is at the end of the last logical CI, the inserted record is placed into the new CA.

Direct Processing

CI Split: Half the records (those with the higher keys) in the CI are moved into the new CI. The new record is inserted (in key sequence) into the CI to which it belongs.

CA Split: Half the CIs (those with the higher keys) are moved to the new CA. Insertion then occurs through regular CI split processing, using the newly-created free space CIs.

Updates can cause CI/CA splits when:

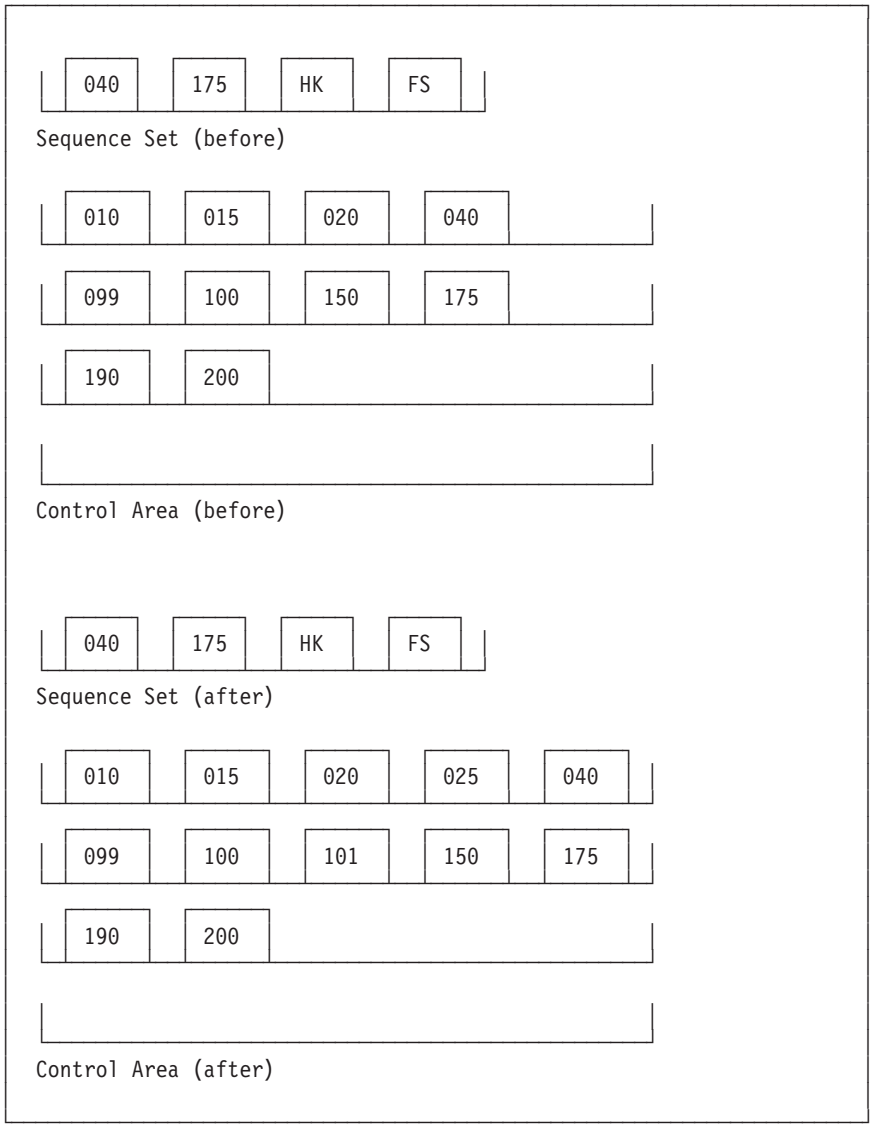
- The record length is increased, and there is not enough free space in the CI, or
- The record length is decreased and additional RDFs are required. If the space required for the RDFs is more than the amount by which the record is shortened, and there is insufficient free space, the CI must be split.

Examples: CI/CA Splits

In the examples:
HK means High Key
FS means Free Space

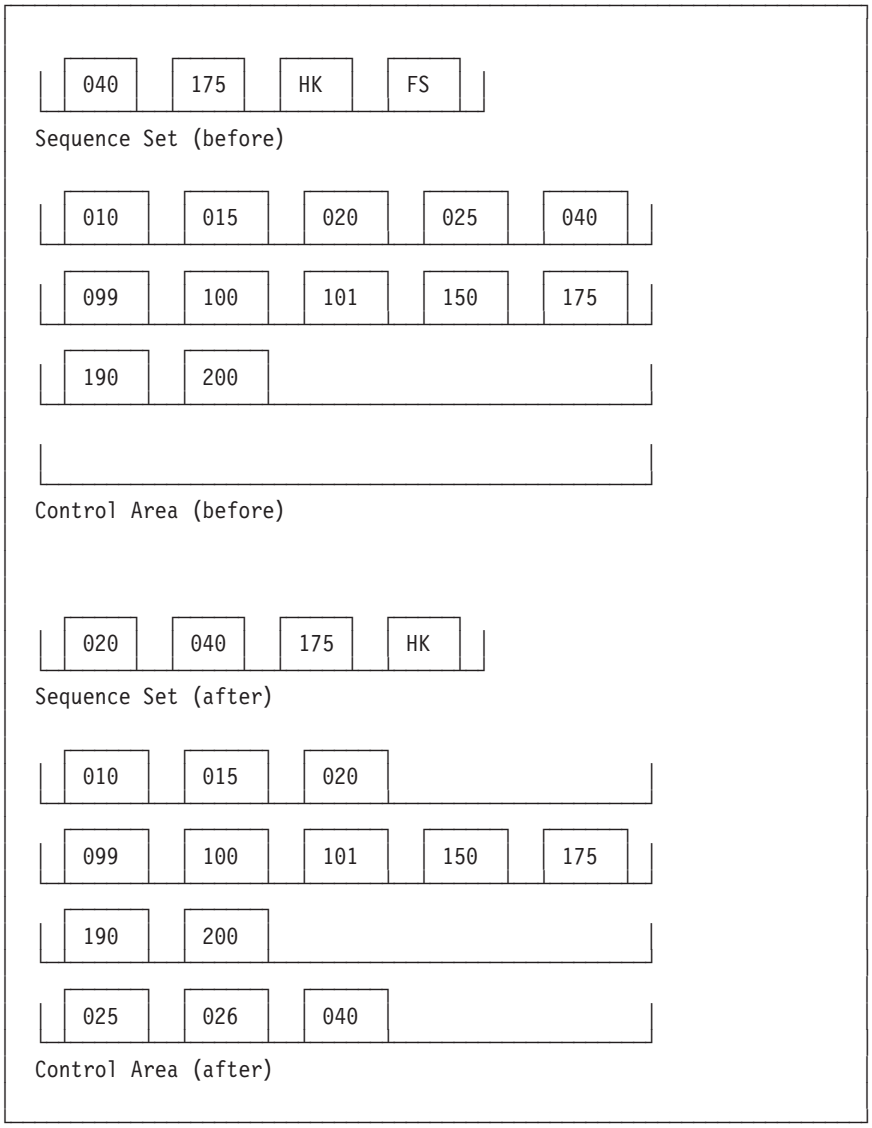
Example 1

shows the CA after direct and sequential insertion of records 025 and 101.



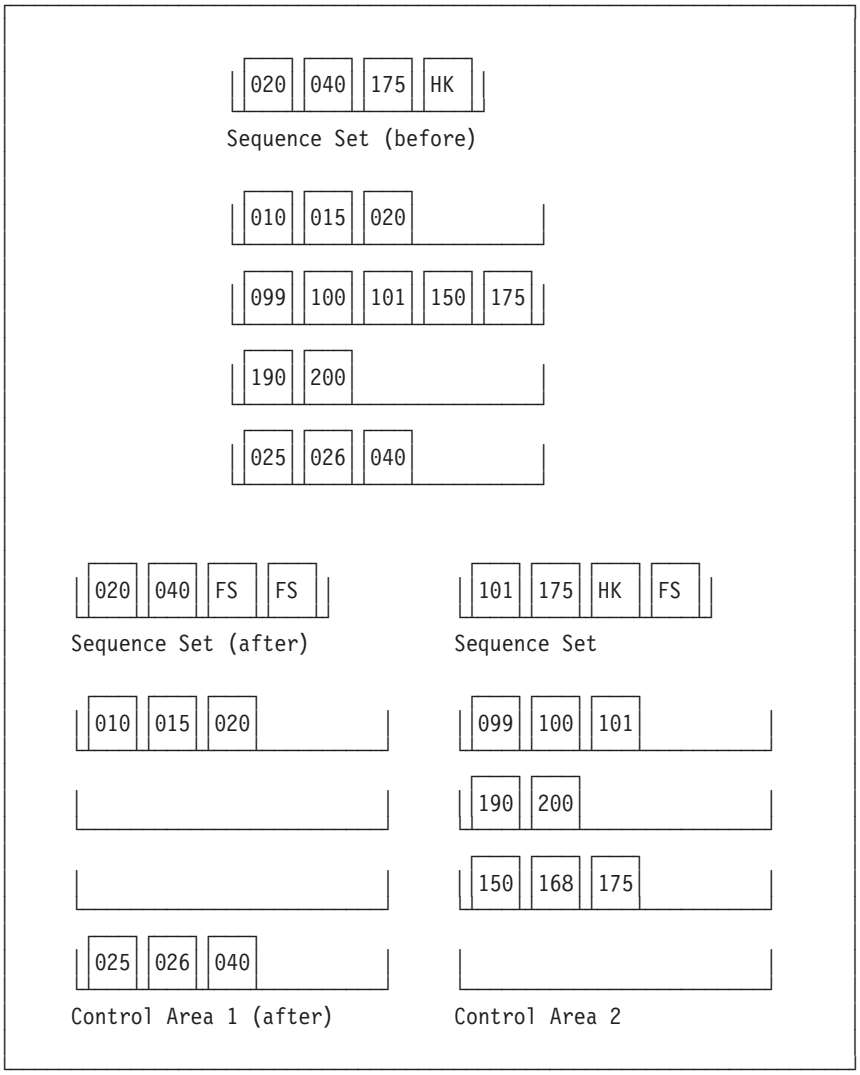
Example 2

shows the CA after direct insertion of record 026, causing a CI split.



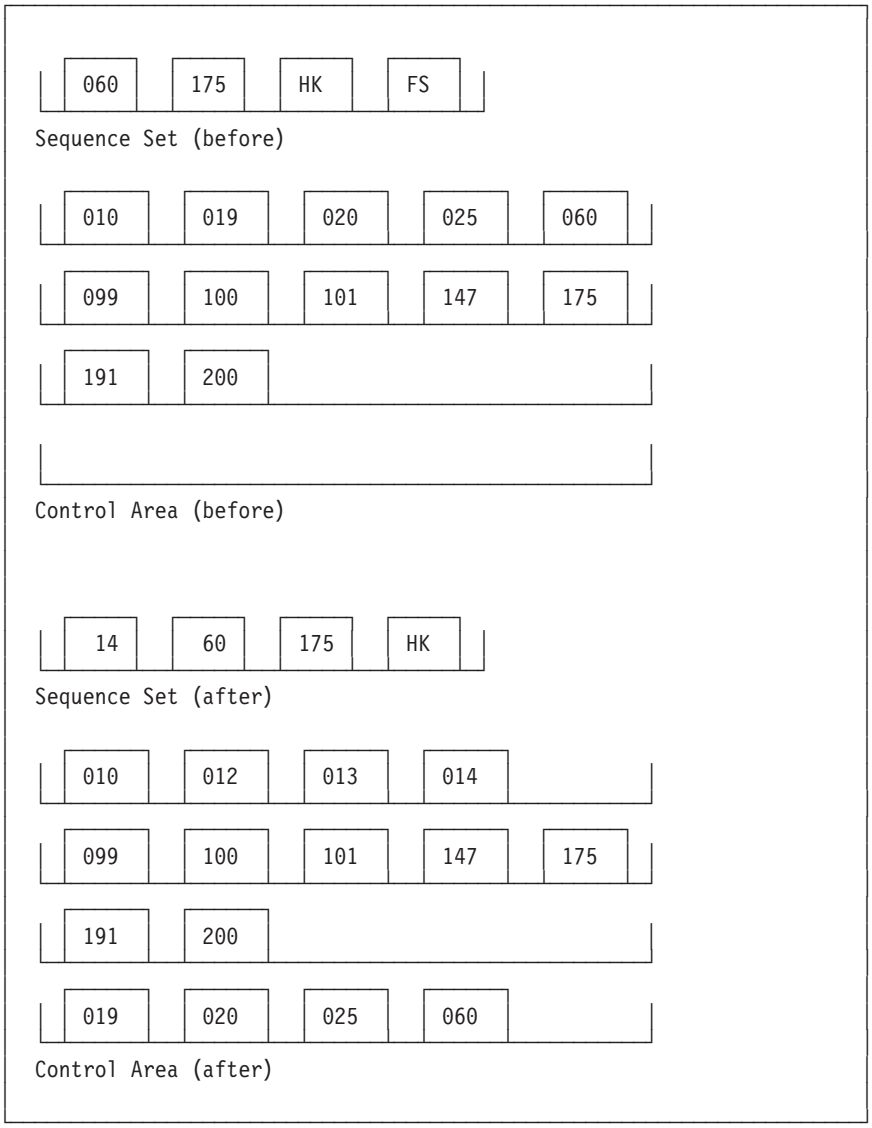
Example 3

shows a CA split and CI split caused by the direct insertion of record 168.



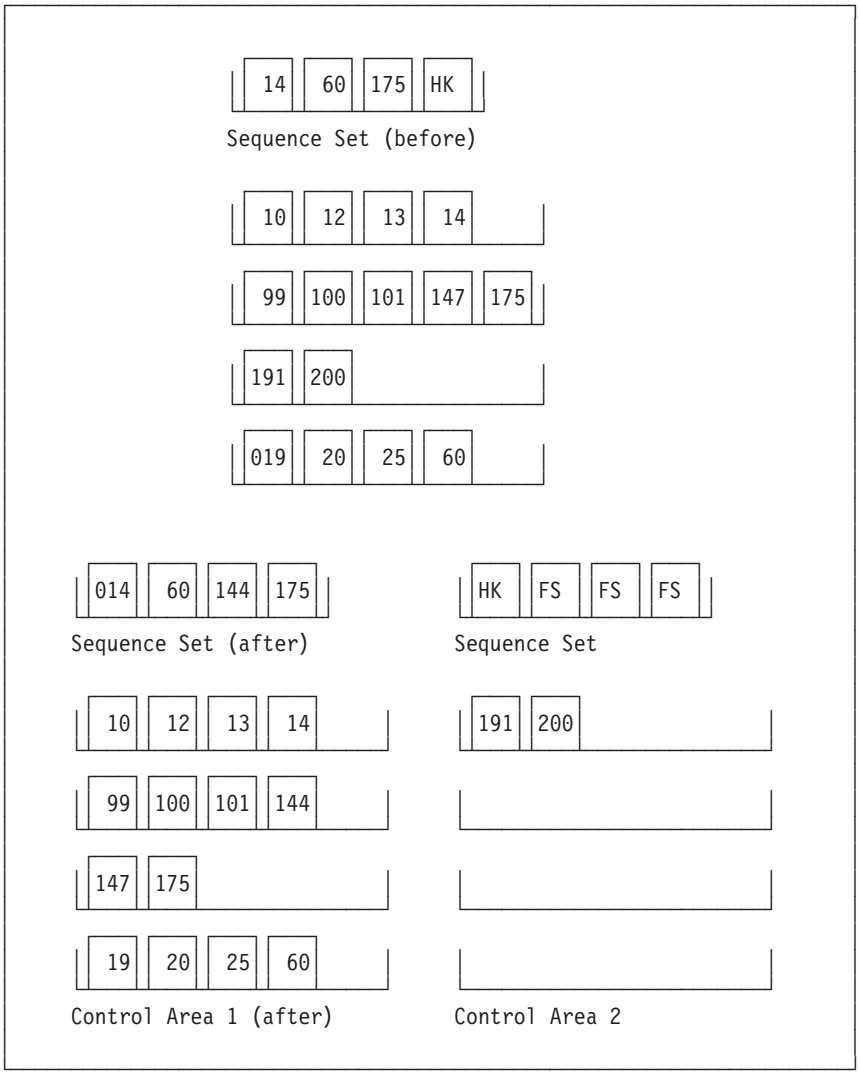
Example 4

shows the CA after sequential insertion of records 12, 13, and 14. Record 12 causes a new CI split. Note that the key associated with the old CI is one number less than the low key in the new CI. This permits mass insertion to take advantage of the newly-created free space.



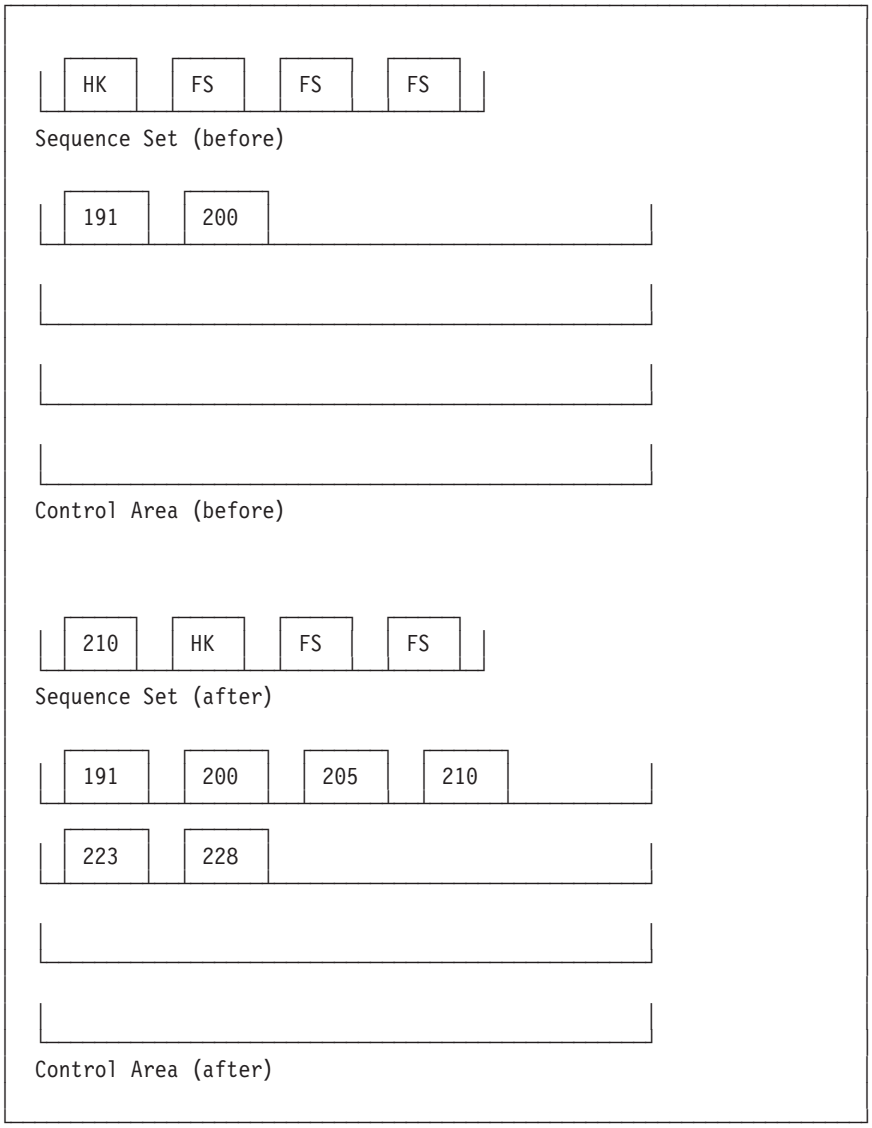
Example 5

shows a CA split and CI split caused by the sequential insertion of record 144. Note that the key associated with the old CI is one less than the low key in the new CI. This permits mass insertion into the newly-created free space.



Example 6

shows a CA after a sequential insertion of records 205, 210, 223, and 228, during load extend processing. Note that the free space is preserved.



Index Options

The following options influence performance and storage requirements for the use of the index of a key-sequenced file or alternate index:

- Number of index records in virtual storage
- Index and data on separate volumes
- Sequence set records adjacent to CAs
- Replication of index records

Number of Index Records in Virtual Storage

For keyed access, VSE/VSAM needs to examine the index of a file. Performance improves if a large number of index records can be held in virtual storage.

Before the processing program begins to process the file, it must specify, either explicitly or by default, the amount of space VSE/VSAM can use to buffer index records. Enough space for one index record is the minimum. However, when the space is large enough for only one or two index records, an index record may be continually deleted from virtual storage to make room for another and then retrieved again later when it is required anew. Ample space in which to buffer

index records can improve performance by preventing this situation, provided that the buffer allocation does not cause excessive paging by z/VSE. Remember that VSE/VSAM searches only the sequence set for sequential access but every index level for direct access.

You can ensure that an acceptable number of index records is in virtual storage by specifying enough space for I/O buffers for index records through one of the following parameters when you open the file:
- BUFFERSPACE parameter of the DEFINE CLUSTER command for a file
- BUFNI and BUFSP parameters of the ACB macro
- BUFNI and BUFSP parameter of the DLBL statement

VSE/VSAM keeps as many index set records in virtual storage as the space will hold. Whenever an index record must be retrieved to locate a record, VSE/VSAM makes room for it by deleting another index record from the space.

Index and Data on Separate Volumes

When you define a key-sequenced file or alternate index, you can place the entire index on a separate volume from the data, either on the same or on a different type of storage device. The data and the index of a cluster (file) are defined separately; the volume that is to contain the data and the volume that is to contain the index are specified in the VOLUMES parameter of the DEFINE command.

Using different volumes enables VSE/VSAM to gain access to an index and to data at the same time. Additionally, the smaller amount of space required for an index makes it economical to use a faster storage device for it than for the data.

Consideration for ECKD Devices

For extended count key data (ECKD) devices (such as the IBM 3390), special considerations apply. Especially in conjunction with cached devices, performance will usually be best if the index is as compact as possible.

Key Ranges

The records of a key-sequenced file and an alternate index can be grouped on different volumes according to key ranges. For example, a payroll file could have employee records beginning with A, B, C, and D on one volume, with E, F, G, H, and I on a second volume, and so on. You can then process the records of every volume, or you can process as many volumes together as your program(s) requires. For more information and examples, refer to “Multiple Volume Support” on page 102.

Performance Measurement

VSE/VSAM keeps statistical information about a file in its catalog record. Some statistics, such as number of extents in a file, number of records retrieved, added, deleted, and updated, and number of CI splits, can help you decide when to take action to improve performance. Appropriate actions could be, for example, reorganizing a file or altering the type of processing.

You can list the entire catalog record, the statistics, and the parameters selected when the file was defined, by using the LISTCAT command. For an explanation of the output produced by the LISTCAT command, refer to the manual *VSE/VSAM Commands* under “Interpreting LISTCAT Output.” You can use the SHOWCB and

TESTCB macros in a processing program to display or test one or more file statistics and parameters. These statistics and parameters include:

- CI size
- Percentage of free CIs per CA
- Number of bytes of available space at the end of the file
- Length and displacement of the key
- Maximum record length
- Number of buffers
- Usage of LSR buffer pools
- Number of records See Note 1, below.
- Password
- A time stamp that indicates if either the data or the index has been processed separately
- Number of levels in the index
- Number of extents
- Number of records retrieved, inserted, deleted, and updated See Note 1, below.
- Number of CI splits in the data and in the sequence set of the index
- Number of EXCPs that VSE/VSAM has issued for access to a file

Notes:

1. VSE/VSAM does not update these statistics when a file is processed in control interval access (that is, MACRF=CNV is specified in the ACB macro).
2. When a cluster or alternate index is exported, that is, named in an EXPORT command, the statistics are reset in the exported catalog record due to the redefinition of the imported cluster or alternate index in another catalog.
3. SAM ESDS statistics are not updated when the file is accessed via DTF.

Displaying Statistics About Buffer Pools

You can use the SHOWCB macro to obtain statistics about the use of buffer pools. These statistics help you to determine how to improve both a previous definition of a resource pool and a 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 (subpool). 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 the resource pool.

For information on the use of SHOWCB, refer to “The SHOWCB Macro” on page 251.

For buffer pool statistics, the keywords described below are specified in the FIELDS operand. 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 I/O operations to bring data into a buffer.
NUIW	The number of nonuser—initiated writes. Applies only for DFR. Writes that VSE/VSAM was forced to do because no buffers were available for reading.
STRMAX	The maximum number of place holders currently active for the resource pool (for the whole resource pool).
UIW	The number of user—initiated writes. For DFR only.

Chapter 8. Data Protection and Data Recovery

This Chapter ...

- Explains which VSE/VSAM options and z/VSE utilities you can use to protect your data. **Data protection** includes:
  - **data security**, the safety of data from theft or intentional destruction.
  - **data integrity**, the safety of data from accidental loss or destruction.
- Includes procedures to **analyze problems** with files, catalogs, and volumes, and shows how to **recover** from error conditions.

For an overview of available **tools**, refer to "Tools for Data Integrity, Backup, and Recovery" on page 139.

Data Protection

VSE/VSAM provides options to protect files against unauthorized use and loss of data. These options allow you to specify passwords and the use of a *user security-verification routine* (USVR), and allow you to control file sharing and data set name sharing. Using IDCAMS commands, you specify the options when you define a file or catalog.

Passwords to Authorize Access

Password Levels

You can optionally define passwords for clusters, alternate indexes, components (data and index), paths, and catalogs. To gain access to password-protected objects, a program or the operator must provide the appropriate password. Password levels differ for various degrees of security. These levels are (from low to high):

- *Read access* (READPW parameter). This is the read-only password, which allows you to retrieve data records and catalog entries, but not to add, update, or delete them, nor to see password information in a catalog entry.
- *Update access* (UPDATEPW parameter). This password authorizes you to retrieve, update, add, or delete records in a file. Specifying a catalog's update password authorizes you to define files in it.
- *CI access* (CONTROLPW parameter). This password authorizes you to retrieve and store the contents of an entire CI (rather than a logical record).
- *Full access* (MASTERPW parameter). This is the master password, which allows you to perform all operations (retrieving, updating, adding, and deleting) on a file and on the catalog entry or any index associated with it. Using this password to gain access to a catalog entry allows you to delete an entire file and to alter any catalog information (including passwords) about data, index, or catalog. The master password allows all operations and bypasses any additional verification checking by a user security-verification routine (USVR).

Every higher-level password allows all operations permitted by lower levels. Any level may be null (not specified), but if a low-level password is specified, the master level password must also exist. The DEFINE and ALTER commands accomplish this by propagating the value of the highest password specified to all the higher password levels. For example, if you specify only a read-level password, that password becomes the update, control-interval, and master level passwords as

© Copyright IBM Corp. 1979, 2005

well. If you specify a read password and a control-interval password, the control-interval password becomes the master level password as well. However, the update level password is not affected (it remains null).

Password Submission

A password, if required, is normally supplied by the processing program in a field pointed to by the ACB or through IDCAMS parameters. If neither of these are supplied, the password must be supplied by the operator. VSE/VSAM prompts the operator for every entry password.

Two options can be specified in the DEFINE command for use when the operator supplies a password: the ATTEMPTS option and the CODE option.

- The ATTEMPTS option specifies how many times, 0 through 7, the operator can attempt to supply the correct password. If 0 is specified, passwords cannot be supplied by the operator. If ATTEMPTS is not specified in the DEFINE command, the default (2) allows the operator to attempt to supply the password twice.
- The CODE option specifies a one-to-eight character name, other than the name (file-ID) of the file, to which the operator responds with a password. This *prompting code* helps keep data secure by not allowing the operator to know both the name of the file and its password. If the CODE option is not specified, the name of the job and the name (file-ID) of the file are supplied to the operator.

If the processing program omits the password or supplies the wrong password, and the operator cannot supply the correct password in the allowed number of attempts, OPEN is terminated. An error code is set in the ACB indicating that the file cannot be opened because the correct password was not supplied.

Password Relationships

Catalogs may have passwords. If you define passwords for any files in a catalog, you *must also* define passwords for the catalog so that the file passwords can take effect. If you do not define passwords for the catalog, no password checking takes place during operations on the file's catalog entry. For some operations (for example, listing all of a catalog's entries with their passwords, or deleting catalog entries), the password of the catalog may be used instead of the password of the file's catalog entry. Thus, if the master catalog is protected, its 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.

Password Checking

VSE/VSAM does password checking only for files that are password-protected. Operations on a catalog may be authorized by the catalog's appropriate password or, in some cases, by the appropriate password of the file whose definition in the catalog is operated on. For example:

- If you want to delete a protected file from a password-protected catalog, you must specify the catalog's or file's master password.
- If you want to alter a file definition in a password-protected catalog, and if the file is password-protected also, you must specify the catalog's or the file's master password.
- If you want to list a file's catalog definition in a password-protected catalog, and if the file is password-protected also, you must specify the catalog's or the file's read (or higher) password. If you want to list the passwords themselves, you must provide the master password.

- If you want to list a file's catalog definition in a password-protected catalog, and if the file is *not* password-protected, you do not have to specify a password.

Because a user catalog defines itself, it may be password-protected without the master catalog being password-protected. To delete an empty user catalog, you must give its master password, whether the master catalog is password-protected or not.

Passwords and IDCAMS Operations

Some IDCAMS operations may involve more than one password authorization. For example, importing a file involves defining the file and loading records into it. If the catalog into which the file is imported is password-protected, its update (or higher) password is required for the definition; if the file is password-protected, its update (or higher) password is required for the load. In these cases, the master password of the catalog satisfies both requirements.

Every VSE/VSAM file is represented in a catalog by two or more entries: a cluster entry and a data entry or, if the file is a key-sequenced file, a cluster entry, a data entry, and an index entry. Of the entries, the *cluster entry* is the controlling entry. Each entry can have its own set of four passwords; the passwords you assign need have no relationship to one another. One reason for this separate password-protection is to prevent access to the index of a key-sequenced file (because an index can be opened independently of the cluster). For example, if you password-protect a cluster but do not password-protect the cluster's data component, another user could issue LISTCAT to determine the name of your cluster's data component, then open the data component and access records in it even if the cluster itself is password-protected.

The following protection considerations and precautions should be observed when using IDCAMS commands that refer to a catalog without using the files defined by the catalog:

- To gain access to passwords in a catalog (for example to list or change passwords), you must specify the master password of either the entry or the catalog. If both the password of the entry and the password of the catalog are supplied, the password of the catalog is used. Similarly, a master password must be specified with the DEFINE command if you want to model the entry's passwords (with the MODEL parameter).
- To delete a protected file entry from a catalog requires the master password of the entry or the master password of the catalog containing the entry. To delete a non-empty VSE/VSAM data space, the master password of the catalog is required, if the catalog is password-protected; to delete an empty VSE/VSAM data space, the update password of the catalog is sufficient. When a catalog entry is created (with a DEFINE command), the catalog's update or higher-level password is required.
- You can list catalog entries that are password-protected by specifying the read passwords of the entries or the catalog's read password. You can list unprotected entries without specifying the catalog's read password. If you wish to list the passwords associated with a catalog entry, you must specify either the master password of the entry or the master password of the catalog.
- If the proper password is not specified with an IDCAMS command, a password prompt occurs. Unless you have specified the CODE parameter on either the DEFINE or ALTER command, the prompt includes the *file-ID* of the file; if you specify CODE, the prompt includes the code name you specified.

  In some circumstances, more than one prompt occurs. For example, when an ALTER or DELETE request is processed, the catalog must be referred to twice,

once to locate the information, and again to perform the requested function. Again, incorrect password specification when you want to list catalog entries may cause numerous prompts. To avoid unnecessary prompts, specify the catalog's master password, which allows access to all entries contained in that catalog.

An unprotected file can be deleted without a password, even if the catalog is protected. This is important during IMPORT and RESTORE processing, because the old, unprotected version of the file is deleted, but a new version is not defined.

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

The following protection considerations and precautions should be observed when using commands that cause access to a VSE/VSAM file:

- To access a VSE/VSAM file by using its cluster name instead of a data or index name, you must specify the proper level of password for the cluster. The proper level password for the cluster is required even if the data or index passwords are null (that is, no password was assigned).
- To access a VSE/VSAM file by using its data or index name instead of its cluster name, you must specify the proper level data or index password. If cluster passwords are defined, however, the master password of the cluster may be specified instead of the proper data or index password.
- If a cluster is not password-protected, you can access the file by using the cluster name without specifying passwords. This is true even if the data and index entries of the cluster have passwords defined. This allows unrestricted access to the VSE/VSAM file as a whole, but protects against unauthorized modification of the data or index as separate components.
- An update password is required at OPEN for MACRF=IN files when DLBL DISP or ACB CLOSDSP is: DELETE or DATE.

User Security-Verification Routine

Programming Interface Information

If you specify password-protection when you define a file or catalog, you can also supply your own routine to double-check the authority of a processing program to access the file. To use this routine, specify the name of *your* user security-verification routine (USVR) in the AUTHORIZATION parameter of the DEFINE or ALTER command.

The verification routine must be a phase residing in the library (LIBDEF=). VSE/VSAM transfers control to the verification routine only after the program that tries to open the file gives a correct password other than the master password. (The verification routine is always bypassed whenever a correct master password is specified.) The authorization option can also include a maximum of 255 bytes of information which will be passed to the authorization routine when it is called. The verification routine receives control in AMODE24. Therefore, the routine must reside below the 16MB line of address space. When the authorization routine gets control from VSE/VSAM, the registers are set as shown in Table 8 on page 129.

Table 8. Register Settings on Passing Control to Authorization Routine

Register	Content
0	Unpredictable.
1	Address of a parameter list with the following format: 44 bytes: File-ID. 8 bytes: Prompting code (specified by the CODE parameter), or zero. 8 bytes: Owner-ID (specified by the OWNER parameter), or zero. 8 bytes: Password that has already been verified. 2 bytes: Length of the authorization string (next field). Up to 255 bytes: Authorization string (specified in the AUTHORIZATION parameter) of DEFINE or ALTER.
2-12	Unpredictable.
13	Address of save area. Note: This address must not be destroyed by the USVR.
14	Return address to VSE/VSAM.
15	Entry point to verification routine.

The USVR should *not* issue any VSE/VSAM macros because they will cause VSE/VSAM to loop. The USVR should return control to the program for any VSE/VSAM requests.

When the authorization routine returns to VSE/VSAM, register 15 should be set to zero if the processing program is authorized to access the file or catalog. If register 15 is not zero, VSE/VSAM does not allow the processing program to open the file.

End of Programming Interface Information

Protecting Shared Data

Files can be shared among partitions, among tasks in a partition, or among z/VSE systems. File sharing is controlled by the use of the SHAREOPTIONS parameter in the DEFINE command, and the type of processing (input or output) for which the file was opened.

For sharing among systems, you must establish the DASD sharing environment through the correct system generation and IPL commands. You are also responsible for ensuring that the volume containing the file is mounted on a *shared device*.

In determining the level of sharing you intend to allow, you must evaluate the consequences of a loss of *read integrity* (reading the correct information) to the processing program, and a loss of *write integrity* (writing the correct information) to the file owner.

The degree of sharing to be allowed for the file is specified, when the file is defined, in the SHAREOPTIONS parameter of the DEFINE command. The SHAREOPTIONS parameter can be changed by the ALTER command (if the file is not concurrently open for another program). A file cannot be deleted or reset if it is currently open for another program, regardless of the share option specified.

During the initial load of a file (and regardless of the share option values specified), VSE/VSAM treats the share option specification as if it were *share option 1* (see below). After the file is loaded and successfully closed, VSE/VSAM uses the specified share option value.

One of the following file share options can be specified, where every open ACB counts as one request:

- Share option 1: The file may be opened by any number of requests for input processing (retrieve records), or it can be opened by one request for output processing (update or add records). This option ensures full (read and write) integrity. Every open ACB counts as one request.
- Share option 2: The file may be opened by more than one request for input processing and, at the same time, it may be opened by one request for output processing. This option ensures write integrity but, because the file might be modified while records are retrieved from it, read integrity must be ensured individually by every user.
- Share option 3: The file can be opened by any number of requests (ACBs) for both input and output processing. VSE/VSAM does nothing to ensure either the integrity of information written in the file or the integrity of the data retrieved from the file. VSE/VSAM does ensure, however, that an open file is not deleted or reset.
- Share option 4: A key-sequenced or relative-record file can be opened by any number of requests (ACBs) for both input and output processing by users in the first system requesting output to the file. Once a file has been opened for output by one system, VSE/VSAM accepts only open for input requests from another system.

  VSE/VSAM ensures write integrity by using the z/VSE LOCK facility. Read integrity is ensured by VSE/VSAM only when records are retrieved for update. If records are *not* retrieved for update, VSE/VSAM may miss or skip some of the records in CIs that are updated concurrently by more than one program. In this case, read integrity is not ensured, because every program might retrieve a different copy of the CI. If one task makes multiple GET/PUT requests (through two or more ACBs) to the same file, VSE/VSAM cannot resolve the integrity conflict and issues an error code. The requestor must resolve the conflict and retry the request.

Note: If you specify share option 4 for an ESDS file, VSE/VSAM treats the specification as if it were share option 2.

If a file cannot be shared for the type of processing you specify, your request to open a file is denied.

If a file is fully sharable (share options 3 and 4), more than one request can open it at the same time to update or add records. If the file is not sharable, only one request at a time can open it to update or add records. With share options 2, 3, or 4, any number of requests can retrieve records from the file regardless of whether it is sharable or not. With share option 1, data retrieval is prevented by the OPEN macro if the file is already opened for output.

If an alternate index is defined with the UPGRADE attribute and share option 1 or 2, keep in mind the restrictions on the number of requests that can open it for input and/or output processing. For example, if you specify share option 2 for an alternate index that is a member of an upgrade set, you cannot open another update path over the base cluster, or the base cluster itself, for output. This is because share option 2 does not allow a file to be opened twice for output.

Cross-Systems Sharing

VSE/VSAM allows the sharing of catalogs and files across z/VSE systems. To this end, the catalogs and files must reside on *shared devices* that have been defined to the supervisor at IPL.

You do not specifically invoke cross-system sharing when opening catalogs and files, because:
- Catalogs are automatically shared if they reside on shared devices.
- Files are automatically shared if they are owned by a shared catalog.

To ensure data protection, the degree of *file sharing* that is to be allowed can be specified in the SHAREOPTIONS parameter of the IDCAMS commands: ALTER, DEFINE CLUSTER, and DEFINE ALTERNATEINDEX. The following explains various options and their results.

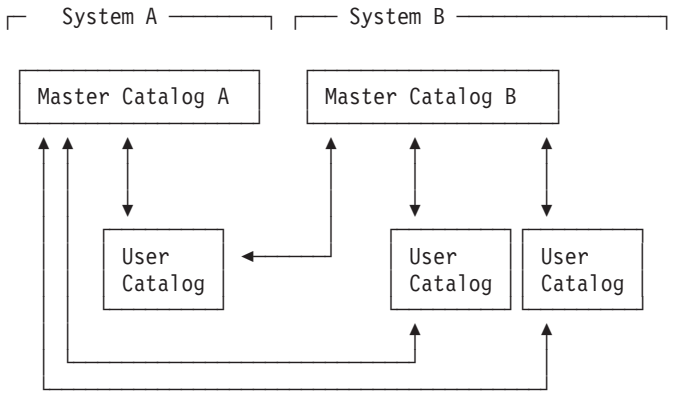
Specifying SHAREOPTIONS(4): This specification provides:
- *Input OPEN function* for all the systems that participate in cross-system sharing, and
- *Output OPEN function* only for the first system that requests it.

If you ignore this restriction, VSE/VSAM issues an OPEN error message with the error code 168 (X'A8'). The error code means that the file is already open for output from another processor, and that only one processor may write output to the file at a time.

Specifying SHAREOPTIONS(4 4): This specification provides OPEN functions for input *and* output processing for all the systems that participate in cross-system sharing. It provides full read and write integrity for a file that is accessed from:
- Different partitions of a particular CPU, or
- Different CPUs.

Note: With SHAREOPTIONS(4 4) specified, the lock file activity (with regard to z/VSE DASD sharing) increases. This may have an effect on performance.

Defining Shared User Catalogs: You may wish to have a non-shared master catalog on every system, and shared user catalog(s) that connect to every master catalog as illustrated in the following diagram:



To do this, define the user catalog under one master catalog, then IMPORT CONNECT the user catalog to another master catalog. The shared (user) catalog(s) must contain entries for all shared files.

Data Integrity

To protect your VSE/VSAM data from accidental loss or destruction, you can use the IDCAMS commands and command options listed below, and you can use the following IBM utility programs:

- VSE/Fast Copy

  The use of this utility in a VSE/VSAM environment requires special considerations, because both the volume VTOC and the catalog contain space mapping information about the volume which has to be synchronized to insure accessibility and to avoid damage to data.

  With VSE/VSAM, data can be flexibly distributed among many DASD volumes (minidisks) of different device types and capacity. However, some rules need to be followed:

  – A catalog resides on a single volume.
  – Only one catalog can exist per volume.
  – A catalog may own space on any number of DASDs of different device types, sizes, and architectures.
  – Several catalogs can own space on the same volume, but then the recovery may become quite complex.
  – Each component of a cluster must reside on the same DASD type. The DASDs can have different sizes.
- VSE/VSAM VTOC Utility (**IKQVDU**)

For brief explanations on when to use which command, option, or utility, refer to Table 9 on page 140. The figure also shows where to find more detailed explanations and procedures.

IDCAMS Commands and Command Options for Data Integrity

BACKUP/RESTORE commands
DEFINE CLUSTER allocation option (See Note)
DEFINE CLUSTER RECOVERY|SPEED option
DEFINE CLUSTER DATA WRITECHECK option
DEFINE CLUSTER WRITECHECK option
DEFINE USERCATALOG command (See Note)
DEFINE SPACE command (See Note)
DELETE SPACE FORCE option
EXPORT/IMPORT commands
LISTCAT command
REPRO command
SNAP command
VERIFY command

Note: Though not specifically designed for the purpose of data integrity, the commands and options DEFINE SPACE, DEFINE CLUSTER, and DEFINE USERCATALOG can be used for that purpose as explained below.

Using the DEFINE SPACE Command

The DEFINE SPACE command and its DEDICATE parameter can be used to easily dedicate an entire volume to VSE/VSAM by defining a space that occupies the whole volume. Other volumes can be used exclusively for nonVSAM files. This allows recovery on a volume basis to be strictly VSAM or nonVSAM. If the volumes are mixed, two different approaches are needed for integrity. For example, a copy of the data on tape is needed to back up the nonVSAM data, but several

exports may be all that is necessary for VSE/VSAM files. Both a COPY run with VSE/Fast Copy and the EXPORT command are necessary on the mixed volumes. If the volumes are segregated, only one of the integrity measures is necessary.

Using the DEFINE CLUSTER Allocation Subparameter

Secondary allocation that occurs after the last catalog backup results in new catalog records that are not available to the backup catalog. The allocation subparameter of the DEFINE CLUSTER command can be used to improve file integrity and reduce this exposure by eliminating or minimizing secondary allocation. An entry-sequenced file is extended only by adding new CAs to the end of the file. Thus, the effect of addition is predictable and the problem is eased. If it is impractical to allocate enough primary space to accommodate additions, the secondary allocation quantity should be large enough so that extension is infrequent.

When secondary allocation is done, a new back up of the catalog or file (or both) can be made. By monitoring the file statistics in the catalog, either by way of a LISTCAT command or by way of a SHOWCB macro against an open ACB (to inspect the number of bytes of available space), you can predict when secondary allocation will occur. You can determine if a secondary allocation took place with a SHOWCB or TESTCB for the RPL feedback information after every PUT request.

For a key-sequenced file the problem is much more complicated. If existing records are not lengthened and all additions are made to the logical end of the file, the situation is similar to that of an entry-sequenced file, except that the index must also be checked. The other patterns of insert and update activity are limitless. Some of them are specific and dictate specific back up strategies, but discussion here assumes a random distribution of activity against the file.

There are reasons, other than recovery, to design a key-sequenced file to minimize extensions. A control-area split takes a relatively long time. For many online systems this can be a serious disruption. A characteristic of key-sequenced files is that, assuming a random insert pattern, all CAs tend to split at roughly the same time. Because every split results in two CAs created from the original one, the file's physical size doubles in a short period of time.

For these reasons it is advisable to design free-space percentages to minimize the probability of a split for a given insert level, rather than to allow extra primary allocation for expansion. The file should be reloaded (reorganized) when its insert level approaches the design point. For further information, see "Distributed Free Space" on page 111.

Using the DEFINE USERCATALOG Command

The DEFINE USERCATALOG command can be used to create many user catalogs (as many as one per volume) and reduce the number of files per catalog. If a catalog becomes unusable and has, for example, only ten files cataloged in it, access to only those ten files has to be recovered.

Note that once a catalog has been destroyed, the data it controls can no longer be accessed. Thus, if a system contains only one (master) catalog and that catalog is destroyed, the resources of the whole system are lost and must be restored by the use of backup copies.

Catalogs with *only nonVSAM entries* can be backed up with VSE/Fast Copy. After the volume is restored, only those jobs that updated the files since the backup was made would have to be rerun.

When several user catalogs are involved, only the resources controlled by the destroyed catalog are affected, and it can be rebuilt while processing on other data continues. Because user catalogs (like the master catalog) are self-describing, you need only rebuild the master catalog and the resources directly connected to it. This applies even if the master catalog has been destroyed. No files in a user catalog connected to that master catalog can be accessed until the user catalog is again connected to a master catalog.

Protecting VSE/VSAM Files and Volumes

You must plan in advance how much and what kind of protection you need. You need to consider questions such as:

- Does it take less time, effort, or expense to recreate lost data than it does to maintain backup copies?
- Should I segregate VSAM and nonVSAM files and make maximum use of recoverability, or is it sufficient to use VSE/Fast Copy plus the file update reruns necessary to make the file current?

The following explanations should help you answer such questions.

Backup Considerations

In choosing methods of back up and recovery, you need to consider the physical matters of accomplishing the work, and the need for back up, operational characteristics, and security and integrity of the backup medium.

- *Necessity for Back Up*: If the file can be recreated from the original input or from records or journals you kept, perhaps there is no need for back up. Considering the time required for regular backup procedures and the relative infrequency of recovery, many files may fit into this category.
- *Operational Factors*: You should consider frequency of back up and possible frequency of recovery, time required for back up and recovery, and the ease or difficulty of the backup and recovery technique used.
- *Frequency Factors*: In deciding for the best method for back up and recovery, you have to find a good balance between the *frequency* of, and the *time required* for making back ups and recoveries. You may find some methods are considerably easier to use than others but may require more time to accomplish. Thus, a method that might be suitable for one file because of its relative infrequency of backing up might be unacceptable for another file that must be backed up frequently.
- *Time Required Factor*: The time required for back up and recovery may be a deciding factor in the choice of method, particularly for real-time systems where recovery must be accomplished quickly. A method that takes longer may have other characteristics that are more desirable. Time required for recovery may also necessitate that a backup technique be used that takes longer.
- *Ease of Use*: The alternatives for back up and recovery vary widely in relative ease of use. Complicated methods that are difficult to use may cause errors, which makes recovery much more time consuming than estimated. If recovery is infrequent, a difficult method may require more time to reason out than another method would require to do the actual recovery.
- *Physical Security and Integrity*: Security and integrity of the backup medium are often neglected. Measures used while data is on the system are of no use for a

backup copy that is stored elsewhere. Security and integrity factors may also need to be reviewed as the nature of data changes in an installation.

Relationship of Catalog Entries to VSE/VSAM Files and Volumes

The VSE/VSAM catalog contains information essential to accessing and controlling its files and volumes. Note the following:

- All VSE/VSAM files must be cataloged. Because the physical and logical description of a file is contained in its catalog entries, VSE/VSAM requires up-to-date catalog entries to be able to access files.
- For multivolume files, the same catalog must own space on all current and candidate volumes.
- Logical and physical mapping information is contained in the catalog entries. For files defined in nonunique VSE/VSAM data spaces, the catalog contains the only record of the physical extents allocated to the file. For unique files, entries in the VTOC also contain a record of physical extents. In both cases, only the catalog contains the logical-to-physical mapping information (the relationship of the RBA ranges of the file to the physical extents).

All other types of data access must use catalog information.

Creating Backup Copies of VSE/VSAM Files

Several methods of back up and recovery can be used for VSE/VSAM files. Usually, it is not possible to use only one method for all files in an installation. You should consider individual files or groups of files, and then determine the most suitable method for each.

- Use the BACKUP command to create a copy of the file on tape or disk. The command backs up empty objects, including catalog entries.

  Note: The file you are backing up must be available for an INPUT OPEN. The OPEN *might* fail if the file is currently opened for input or output by another partition or system. Because the OPEN might not *always* fail, it is strongly recommended that the file which is being backed up should not be opened for output by any other partition or system. Otherwise, the resulting backup copy might not represent the actual state of the original file.

  Use the RESTORE command to create - from the backup copy - an object that is equivalent to the original one. You can also use the RESTORE command to move the files to a different disk device type, or to increase the allocation size of a file.

  You can back up (or restore) all the objects owned by one catalog (or contained on the backup file) with a single command. Generic names let you include or exclude subsets of objects from the backup or restore operation.

  Note that the format of the file produced by BACKUP is different from the format produced by EXPORT. Therefore:

  – RESTORE cannot process files created by EXPORT, and

  – IMPORT cannot process files created by BACKUP.

  Recommendation: Because of their performance advantage, BACKUP and RESTORE should be used for regular back up of files, with restoration as necessary. EXPORT and IMPORT should be used for migration between VSE/VSAM and MVS, and for reorganization on record-level or CI-level. For

optimum performance a COMPRESSED file is stored by BACKUP. A compressed file can only be restored on a system with support for VSE/VSAM data compression (that is VSE/VSAM Version 2 or later).

- Use the EXPORT command to create an unloaded, portable copy of the file. The operation is simple. There are options that offer protection, and most catalog information is exported along with the data, easing the problem of redefinition. You can prevent the exported file from being updated until the IMPORT command reestablishes its accessibility. A COMPRESSED file is backed up by EXPORT in an uncompressed format, hence the IMPORT can be done by any system supporting the IMPORT command. IMPORT defines the file as a NOCOMPRESS file, unless the target file is a pre-defined, empty file with the COMPRESS attribute.

  For more information and examples, refer to the manual *VSE/VSAM Commands* under "Using EXPORT/IMPORT for Transporting or Backing Up Files."
- Use the REPRO command to create either a SAM file, or a duplicate VSE/VSAM file for back up. The advantage in using REPRO (instead of EXPORT) is the accessibility of the backup copy. A DEFINE command is required before reloading, but this is a relatively minor inconvenience, particularly if the original DEFINE statements can be used. A COMPRESSED file is copied by REPRO in an uncompressed format.

  For more information and examples, refer to the manual *VSE/VSAM Commands* under "Loading Records into a File."
- User-written programs for back up are usually appropriate when the data has some characteristic that does not allow you to take advantage of a generalized backup method. Files for which not all records have to be saved for back up might fit into this category. Also, keyed sequential files which have to be processed sequentially on a regular basis could be backed up by creating a sequential file as a by-product.

You must keep in mind that any backup procedure that does not involve an image copy of the file (for example, the BACKUP, EXPORT, and REPRO commands do not provide an image copy of the file) will result in data reorganization and the re-creation of the index for a key-sequenced file. Therefore, any absolute references by way of RBA may become invalid.

Creating Backup Copies of Volumes

You can use VSE/Fast Copy to create a backup copy of an entire volume and to restore that copy on a volume. However, the use of this utility in a VSE/VSAM environment requires special considerations, because both the volume VTOC and the catalog contain space mapping information about the volume that has to be synchronized to ensure accessibility and to avoid damage to data. Therefore, it is generally recommend that every volume should have its own user catalog. This can make the problem of synchronizing data and catalog information much simpler.

For details on how to use "VSE/Fast Copy", refer to the manual *z/VSE System Utilities*.

For information on how to solve problems relating to catalogs and volumes, refer to "Procedures for VSE/VSAM Recovery" on page 143.

Protecting VSE/VSAM Catalogs

Because of the importance of the VSE/VSAM catalog, you should consider to back up catalogs as well as files and volumes. If all of the files owned by a catalog are backed up individually, it is possible to recover from destruction of the catalog by carrying out recovery procedures for every file. The probability of losing an entire catalog is very low. However, to speed recovery or minimize exposure in the case of catalog damage or destruction, three backup methods are available:

- Use the REPRO command to create a backup copy of either a master catalog or user catalog, and to reestablish that backup copy as a catalog. Use the command to unload the catalog to a VSAM or nonVSAM file.

  This set of functions is referred to as catalog *unload* and *reload*. The REPRO command requires no special parameters to perform the function. The unload function is triggered when the REPRO source is a catalog. The reload function is triggered when the REPRO target is a catalog. When a new catalog is defined, an unloaded catalog file may be reloaded into the newly defined catalog, or the unloaded catalog can be reloaded into a version or the original catalog.

  Before using *REPRO unload/reload* as the method for catalog recovery, refer to the manual *VSE/VSAM Commands* under "Using REPRO for Catalog Backup and File Reorganization."
- Use VSE/Fast Copy to back up the entire catalog volume.

The following explains how to proceed if integrity problems occur with catalogs, files, or volumes.

Creating Backup Copies of Catalogs

You should protect catalogs by backup procedures against:
- Loss of data.
- Unusable catalog.

Protect Against Loss of Data, and Recover

The only way to safeguard against loss of data is to have a copy of the data in another form or place. The usual method of doing this is to use the BACKUP command of the VSE/VSAM Backup/Restore Function to copy the volume to tape or to another disk volume.

If you have lost a file and if you do have a backup copy, use the RESTORE or IMPORT command to copy the volume to disk. Then reprocess any updates made since the backup copy was made. If you do *not* have a backup copy of the file, you must recreate the file by redefining, loading, and updating the file.

Before you restore a volume, consider the following:

- If the information on the restored volume is downlevel, (your original volume has been updated since the back up was made), you must apply the updates to the restored level of the volume to bring it to the level of the original volume.
- If the volume is not the catalog volume, you have information about the volume in the catalog that may not match what is actually on the volume. It would be helpful to have a LISTCAT listing of the catalog at the time you created the backup copy to compare with a present listing. The data spaces and file extents may be different if any file updates have been made since the back up was made. See "Volume is Inaccessible" on page 148 for a complete recovery procedure.
- If the volume is the catalog volume, all of the volumes owned by the catalog may have file and data space extents that do not match the catalog information.

Again, LISTCAT listings of the backup copy and the original catalog can be of help. Every volume must be handled as if the volume was just restored. See "Volume is Inaccessible" on page 148.

Protect Against Unusable Catalog, and Recover

If there is *no* loss of data, but the catalog is partially or totally unusable, you can use (depending on prevailing conditions) one or the other of the following methods:

- If REPRO has been used to periodically copy your catalog, perform the following steps:
  1. BACKUP or EXPORT those files that have been updated (and that can still be accessed through the catalog) since the catalog copy was made.
  2. Reload the catalog using the catalog REPRO copy.
  3. RESTORE or IMPORT the files copied in step 1.
  4. If there are files that were updated but could not be copied by BACKUP or EXPORT, recreate the files from back level copies by reprocessing updates.

  If you do not have a REPRO unload copy, you have to restore a volume backup as explained under "Volume is Inaccessible" on page 148.
- If you do *not* have a copy of the catalog, perform the following steps:
  1. BACKUP or EXPORT those files that have been updated (and that can still be accessed through the catalog) since the last backup.
  2. Delete the data spaces from the volume.
  3. Redefine the catalog and data spaces.
  4. RESTORE or IMPORT the files copied in step 1.
  5. RESTORE or IMPORT the files not copied in step 1 from the backup tape.
  6. If there are files that could not be processed in step 1, recreate them from back level copies by reprocessing updates.

Rebuilding a Catalog

You may have to rebuild your catalog if it gets damaged. Use the following procedure:

1. Run a LISTCAT command to determine which files own space on the volume. Assuming that you want to save the contents of these files, determine if an acceptable back level copy exists of each. If not, save the contents of these files by running either BACKUP or REPRO.

   The BACKUP command is preferable, because it automatically saves any alternate indexes built over the cluster backed up. (If REPRO is used, you must rebuild these alternate indexes at restoration.) If there is catalog damage, it may not be possible to recover all files.
2. Issue a DELETE command for every object in the catalog. (alternate indexes and paths associated with the file are automatically deleted.)
3. Issue a DELETE SPACE command for all volumes owned by the catalog.
4. Delete the catalog itself.
5. Define the catalog.
6. Issue a DEFINE SPACE command for any volumes on which the catalog will own space.
7. Define a compression control data set.
8. If any files (and associated alternate indexes or paths) were deleted in step 1, reintroduce them into the catalog in one of the following ways:

- If you used BACKUP in step 1, use the RESTORE command to define and restore objects saved in step 1.
- If you used REPRO in step 1, DEFINE every object that was deleted in step 2. Then use REPRO to restore the objects saved in step 1. Also DEFINE any alternate indexes or paths deleted in step 2. Recreate any associated alternate indexes using the BLDINDEX command.

Guide to VSE/VSAM Recovery

About Data Organization and Recovery

Questions and considerations on *VSE/VSAM recovery* have a very close relationship to how you organize the VSE/VSAM data so that it can be recovered more easily, and on how you provide for backup data. For information on these topics, refer to:

"Data Integrity" on page 132
"Protecting VSE/VSAM Files and Volumes" on page 134
"Protecting VSE/VSAM Catalogs" on page 137

If the time required for recovery is the governing factor, follow the preparation and recovery steps explained under "Quick Recovery" on page 150.

About the Recovery Process

VSE/VSAM recovery is the process of regaining access to lost VSE/VSAM data. To regain access to lost data, you can use a combination of functions from VSE/VSAM, IDCAMS, and z/VSE system utilities.

Levels of Recovery

The types of VSE/VSAM data recovery, in terms of the currency of the recovered data, are: *current* and *downlevel*.

The *current* type of data recovery operation restores addressability and access to the most recent version of the data. Operations that recover current data are generally used to correct problems such as read and write errors associated with the data itself or with the data description.

The *downlevel* type of data recovery operation restores addressability and access to a version of the data other than the most recent. Operations that recover downlevel data are generally used to correct logical problems such as a programming error or faulty transactions. This is the most common type of recovery, probably because of the types of problems encountered and the level of data available for recovery. An example of a downlevel recovery is the restore of a volume.

Note: Some of the utilities (listed in Table 9 on page 140) can only recover data that currently is not *downlevel*. Further processing is necessary to make the file, volume, or catalog current.

Tools for Data Integrity, Backup, and Recovery

Table 9 lists the integrity options, backup programs and commands, and recovery tools. In the figure, the column headings that are not self-explanatory have the following meaning:

TOOL TYPE indicates where the tool is supported: in VSE/VSAM, in IDCAMS, or in z/VSE. For more information on *current* and *downlevel*, refer to "Levels of Recovery" on page 139.

FILE TYPE indicates what is recovered:
FILE means VSE/VSAM file.
VOL means volume.
CAT means VSE/VSAM catalog.

TOOL CLASS indicates the command or program class:
INT is any tool that is a VSE/VSAM integrity option.
BKP is a backup command or program other than recovery, and other than recovery-type tools.

(X) in the last column means that you have to refer to the manual *VSE/VSAM Commands*. There, you find the discussion under the quoted title.

Table 9. Tools for Integrity, Backup, and Recovery

Tool Name	Tool Type	File Type	Tool Class	Application	Where Discussed
VSE/Fast Copy	z/VSE Utility	VOL	BKP	Use the *VSE/Fast Copy* system utility to create a backup copy of an entire volume and to restore that copy on a volume. The use of this utility in a VSE/VSAM environment requires special considerations, because both the volume VTOC and the catalog contain space mapping information about the volume which has to be synchronized to insure accessibility and to avoid damage to data.	• "VSE/Fast Copy" in the manual *z/VSE System Utilities*.
BACKUP/ RESTORE	IDCAMS	CAT or FILE	BKP	Use BACKUP and RESTORE for high-speed data recovery operations, or to move files to a different disk device type, or to change the allocation size of the file.	• **(X)** "Using BACKUP and RESTORE" in the *VSE/VSAM Commands*. • "Creating Backup Copies of VSE/VSAM Files" on page 135, and "Creating Backup Copies of Catalogs" on page 137.
EXPORT/ IMPORT	IDCAMS	FILE	BKP	Use the EXPORT command to create backup copies of data and associated catalog entries. The catalog entries can be reestablished in the catalog from which they were extracted or into a different catalog using IMPORT command. The data file is reestablished by IMPORT without redefining it.	• **(X)** "Using EXPORT/IMPORT for Transporting or Backing Up Files" in the *VSE/VSAM Commands*. • "Creating Backup Copies of VSE/VSAM Files" on page 135, and "Creating Backup Copies of Catalogs" on page 137.

Table 9. Tools for Integrity, Backup, and Recovery (continued)

Tool Name	Tool Type	File Type	Tool Class	Application	Where Discussed
LISTCAT	IDCAMS	FILE, VOL, CAT	REC	Use the LISTCAT command to list the contents of a catalog after a recovery operation. Visually compare this list with a copy of the LISTCAT list most recently done before the recovery. For a description of the out-of-synchronization condition you may find, see "Catalog Entry Mismatches" on page 147.	• **(X)** "Listing Catalog Entries" in *VSE/VSAM Commands*. • "Catalog Entry Mismatches" on page 147.
REPRO	IDCAMS	CAT	BKP	The REPRO command is used to create a backup copy of catalog. The unloaded or backup copy can be reloaded into a newly defined catalog or a version of the original if the backed up catalog becomes unusable.	• **(X)** "Using REPRO for Catalog Backup and File Reorganization" in *VSE/VSAM Commands*. • "Creating Backup Copies of Catalogs" on page 137.
SNAP	IDCAMS	VOL	BKP	Use the SNAP command to copy entire ESS volumes so that a backup operation can be performed on the target volumes involved.	• **(X)** SNAP command in *VSE/VSAM Commands*.
VERIFY	IDCAMS	FILE	INT	Use the VERIFY command if you want to compare the file's catalog information with the EOF indicator in the file.	• **(X)** "Verifying a File's Accessibility" in the *VSE/VSAM Commands*.
VTOC Utility (IKQVDU)	VSE/VSAM Utility Program	VOL	BKP	Use the VSE/VSAM VTOC utility program IKQVDU to initialize a VSE/VSAM-owned volume when the owning catalog is not available. VSE/VSAM volume ownership can be given up and VSE/VSAM space can be returned to the VTOC as available space. All data in that space is lost. Caution. The owning catalog is not modified.	• "Maintaining VTOC and VOL1 Labels on Disk (IKQVDU)" on page 355.
DEFINE SPACE	IDCAMS	VOL, CAT	INT	Use the DEFINE SPACE command to dedicate use of volumes for VSE/VSAM files in order to segregate VSE/VSAM and non-VSE/VSAM recovery. You can dedicate a volume by defining a VSE/VSAM data space that occupies the whole volume, or by specifying the DEDICATE parameter.	• **(X)** "Defining a VSE/VSAM Data Space" in the *VSE/VSAM Commands*. • "Data Integrity" on page 132.

Table 9. Tools for Integrity, Backup, and Recovery (continued)

Tool Name	Tool Type	File Type	Tool Class	Application	Where Discussed
DEFINE USER-CATALOG	IDCAMS	VOL, CAT	INT	Use the DEFINE USERCATALOG command to maximize the use of user catalogs and to limit the use of the master catalog. Compare the effect of the loss of a catalog when 10 files are cataloged and 50 files are cataloged in every of two catalogs. The fewer the catalogs the greater the disruption of daily operations in the event of loss of a catalog.	• **(X)** "Defining Objects in a Catalog" in the *VSE/VSAM Commands*.
DEFINE option WRITE-CHECK	IDCAMS	FILE	INT	Use the optional WRITECHECK parameter of the DEFINE command to verify every write operation when writing data to auxiliary storage. (See the WRITECHECK parameter for an explanation.)	• **(X)** "Defining a VSE/VSAM File (Cluster)" in the *VSE/VSAM Commands*. • **(X)** "DEFINE CLUSTER" in the *VSE/VSAM Commands*.
DELETE SPACE FORCE	IDCAMS	VOL	INT	Use the DELETE SPACE FORCE command to remove information from both the VTOC and the catalog. When space is deleted by using FORCE option, the VTOC's VSE/VSAM volume ownership is given up (if no other catalogs own space on that volume), the catalog's VSE/VSAM space is returned to the VTOC, the space definition in the catalog for that volume is deleted, and VSE/VSAM files on that volume are marked as unusable in the catalog. If you want to redefine the files, you must first delete them.	• **(X)** "Altering Catalog Entries" in the *VSE/VSAM Commands*.

Table 9. Tools for Integrity, Backup, and Recovery (continued)

Tool Name	Tool Type	File Type	Tool Class	Application	Where Discussed
DEFINE CLUSTER RECOVERY SPEED	IDCAMS	FILE	INT	When you define a cluster, you can indicate that VSE/VSAM is to preformat every CA as records are loaded into the cluster (RECOVERY) or is not to preformat them in interest of performance (SPEED). As records are loaded into a preformatted area there is always a following end-of-file indicator that indicates how far loading has progressed. If an error occurs that prevents loading from continuing, you can readily identify the last successfully loaded record and resume loading from that point.	• **(X)** "Defining a VSE/VSAM File (Cluster)" in the *VSE/VSAM Commands*. • "Data Protection and Integrity Options" on page 110.
DEFINE CLUSTER ALLO-CATION	IDCAMS	FILE	INT	Minimize or eliminate secondary allocations for files to overcome the difficulty in catalog recovery stemming from secondary extents.	• "Using the DEFINE CLUSTER Allocation Subparameter" on page 133.

Procedures for VSE/VSAM Recovery

You can use the following procedures to analyze and to recover from the following conditions:
- "File is Not Properly Closed," below
- "File is Inaccessible" on page 145
- "Catalog is Unusable" on page 146
- "Volume is Inaccessible" on page 148

Because the two activities *backup* and *recovery* overlap, read also the explanations under:
"Creating Backup Copies of VSE/VSAM Files" on page 135
"Creating Backup Copies of Volumes" on page 136
"Creating Backup Copies of Catalogs" on page 137

Several of the following procedures use volume restore. If this is indicated, one or the other of the following must be true:
- The volume restored does not contain multivolume files.
- If a volume does contain a portion of a multivolume file, all volumes that contain portions of those multivolume files are treated as a single unit. That is, if a volume is required, the entire set is restored.

File is Not Properly Closed

Cause of Failure

VSE/VSAM files are not properly closed when they are opened for output and a system failure occurred, or automatic CLOSE was not activated. This condition is

reflected in the catalog and is communicated to the next program that does an OPEN of the file. There is a possibility that the failure occurred after the load or update of the file was complete. In this case, the file itself and the file's catalog entry are correct.

Error Conditions:

- Incorrect high RBA in catalog
- Incomplete write to direct access device
- Duplicate data

Overview

The warning "file not properly closed" may indicate an error in a VSE/VSAM file. This condition can generally be corrected by using the VERIFY command. If other errors are encountered or suspected, they can generally be corrected by using either the IMPORT command or the REPRO command.

Recovery for Incorrect High RBA in Catalog

This is the error most likely to occur. If you are running in RECOVERY mode, all you need to do is reopen the file, and the automatic VERIFY function of VSE/VSAM will correct the error and update the catalog with the correct high RBA. However, VSE/VSAM cannot correct the following:

- If an ESDS file opened for control interval (CI) access.
- If a SAM ESDS file is in non-CI format.
- If a SAM ESDS file is in CI format. VSE/VSAM cannot update the EOF indicator, because the file is always loaded and extended in SPEED mode.

Recovery for Incomplete Write to a Disk Device

The file must be restored from a backup copy. You can use either an exported or sequential backup copy created by the REPRO command.

Use the IMPORT command to put a previously exported copy into the catalog, or:

1. Delete the file that failed.
2. Redefine the file with the DEFINE command.
3. Load the new file with the sequential backup file by using the REPRO command.

The restored file is downlevel and all updates since the back up was made must be reapplied to make the file current.

Recovery for Duplicate Data in a Key-Sequenced File, Alternate Index, or Catalog

This can result from a failure during a CI or CA split. One of two possible situations can exist for a duplicate data error conditions, depending on the type of processing done.

For addressed or control interval (CI) processing, you correct the error condition by using the REPRO command to copy the current version of data to a temporary file and then copy it back into the original file. This gives you a reorganized file without duplicate data.

For keyed or sequential processing, VSE/VSAM automatically detects and corrects the duplicate data condition. (VSE/VSAM erases the original versions of the copied records.) Duplicate records caused by a failure during a CI split may cause an error if the file is processed by z/VSE.

File is Inaccessible

Cause of Failure

A VSE/VSAM file may become inaccessible due to damage to the file itself, damage to related information in the catalog, or both. Depending on the extent of damage and prior actions, it may be possible to gain access to either the current or a downlevel version of the data.

Error Conditions:

- The file cannot be opened
- The file is partially unreadable (but can be opened)
- The file is totally unreadable (but can be opened)
- The compression status of the file is CMP-UNDET

Overview

The inaccessibility of a VSE/VSAM file can be analyzed by using the LISTCAT command, and the extent of file damage can be determined. Based on the analysis, you can recover the data by using BACKUP/RESTORE, EXPORT/IMPORT, and REPRO.

Recovery for a File that Cannot Be Opened

The problem is probably due to catalog damage. Determine the extent of this damage. If the damage is relatively minor (that is, relatively few catalog file entries are affected):

1. Use the analysis tool LISTCAT to determine the extent of damage. This can be done by comparing a previous LISTCAT list with one of the damaged catalog.
2. For a *catalog*, if a back level copy of the file is available, you can RESTORE or IMPORT the file to gain access to the file.

Recovery for a File that is Partially Unreadable

The problem is either confined to the file itself, or to an entire physical extent of the file.

1. Use an analysis tool as outlined in "Recovery for a File that Cannot Be Opened" to determine if there is a mismatch in the number of extents. If the catalog indicates one or more extents than there are on the volume, it may be caused by a volume restored independent of the catalog.
2. For a catalog, you can import a previously exported copy. See "Recovery for a File that Cannot Be Opened" for use of these tools.
3. If there is no catalog mismatch, a backup copy of the file must be restored, using BACKUP/RESTORE, EXPORT/IMPORT, or REPRO.

Recovery for a File that is Completely Unreadable

Either the file has been destroyed, or the catalog and volume are not synchronized.

1. Analyze the catalog with LISTCAT to determine if the damage is in the file or in the catalog.
2. Regain access to data
   - If the damage is to the file or a catalog, use IMPORT or REPRO to restore the file. This gives you access to a downlevel copy of the data.
   - If the file has a CMP-UNDET compression status, the backup copy of the file must be restored, using BACKUP/RESTORE, EXPORT/IMPORT, or REPRO.

Catalog is Unusable

Cause of Failure

A catalog may become unusable because of physical damage to the catalog. Depending on the extent of the damage and prior actions, it may be possible to gain access to current level catalog entries or to downlevel catalog entries.

Error Conditions:

- Catalog can be opened, but many VSE/VSAM files cannot be opened.
- The catalog cannot be opened.
- The catalog volume is unusable.

Overview

An unusable catalog can be reestablished, provided certain backup procedures made possible by the system copy utility and the REPRO command are followed. This provides a downlevel version recovery when a file or volume is damaged or unusable.

Recovery for a Catalog that Can Be Opened, but Many VSE/VSAM Files Cannot Be Opened

A problem with the catalog probably exists. This can be determined by using an analysis tool. If I/O errors are encountered or mismatches are detected, some form of catalog recovery is required. If not, the problem is confined to the files themselves and the procedures given for "Recovery for a File that Cannot Be Opened" on page 145 can be used.

1. Use the analysis tool LISTCAT to determine if a problem exists in the catalog. This can be done by comparing a previous LISTCAT list with one of the suspect catalog.
2. If the problem is with the catalog, recovery depends on the availability of backup copies of the catalog, volumes, and files.

   Proceed as follows:

   a. Delete each VSE/VSAM file that cannot be opened.
   b. Redefine these files in the catalog, or use the IMPORT command to load backup copies created by the EXPORT command.
   c. If backup copies created by the EXPORT command are not available, load the files with backup REPRO copies, if available.

Recovery for a Catalog that Cannot Be Opened

You must have either:

- Backups of all data sets of this damaged catalog, or
- A copy of the whole volume.

For *catalogs*, proceed as follows:

1. As applicable:
   - Reload the backups of the data sets into a newly defined catalog.
   - Restore the copy of the whole volume.
2. Use LISTCAT listings of backup files and current files to determine if there are mismatches. If entry mismatches are detected, see "Catalog Entry Mismatches" on page 147.
3. For those files with other than an RBA or general mismatch, delete the file and reestablish with a backup copy of the file created by the IMPORT command or the REPRO command.

Recovery for a Catalog Volume that is Unusable

- See the procedure "Recovery for a Catalog that Cannot Be Opened" on page 146.

Catalog Entry Mismatches

Whenever a catalog is used out of synchronization with the volumes it owns, there is the possibility that the information in the catalog does not match the physical characteristics of the volumes or files that it describes. Catalog entry mismatches may indicate that the data is inaccessible, partially accessible or completely accessible.

The descriptions of catalog mismatches are meant to guide you through a comparison of two LISTCAT listings that you have produced:
One listing of a catalog taken *before* the catalog is restored.
One listing of a catalog taken *after* the catalog is restored.

If you notice a difference in the entries of the two listings:

1. For the description of LISTCAT keyword fields, consult the manual *VSE/VSAM Commands* in "Interpreting LISTCAT Output."
2. Determine what mismatch has occurred by following the descriptions given under "Determination of Mismatches" below.

This method is for the analysis of catalogs for out of synchronization conditions that may occur when the catalog is restored to a previous level.

Determination of Mismatches: By comparing the LISTCAT runs that were made when the catalog was backed up with those when the catalog is restored, critical changes can be detected.

Volume Mismatches:

Mismatched Space Map: This mismatch indicates that the catalog does not correctly reflect the tracks (Min CAs) on the volume occupied by its VSE/VSAM files. Files wholly contained in space correctly indicated as allocated can be accessed if their file descriptions in the catalog are correct.

Mismatched Data Space Group: This mismatch shows that the catalog does not correctly reflect the VSE/VSAM files that it owns on the volume. Files wholly contained within data spaces that are accurately described are accessible if their file descriptions in the catalog are correct.

Mismatched File Directory: This mismatch shows that the catalog does not correctly reflect the files it owns on the volume. Files known from the file directory are accessible if their descriptions are correct.

File Mismatches:

Mismatched Statistics: These mismatches do not affect accessing of a file.

Mismatched High RBA: This mismatch indicates that the catalog does not correctly reflect the end of data in a file. Correct this condition by reopening the file, which causes automatic VERIFY to reset the high RBA.

Mismatched Extents: This mismatch indicates that the file has acquired additional extents that are not reflected in the catalog. The data contained in the extents that

are correctly identified may be accessed. For a key-sequenced file it may be necessary to treat the data portion as an entry-sequenced file 0n order to access the data.

Mismatched Volume or Key Range: This mismatch indicates that the file:

- Was extended to a volume which is unknown to the catalog's file record, or
- On the volume has the same name as the catalog, but it is not the same file that is described in the catalog.

If the file was extended to a volume not known in its catalog record, the extents of the file on that volume are not accessible. The extents of the file on known volumes may be accessible.

Actions that Cause Catalog Mismatches: There are several actions that cause mismatches from a backup catalog. Some of these are overt actions such as the use of the DEFINE and DELETE commands to create files or data spaces. Others are automatic system actions, such as acquiring additional extents.

Define/Delete/Extend Data Space: Any of these actions cause the data space group set of fields for a data space to be invalid in a backup catalog.

Define/Delete Files: Either of these actions cause the file directory in the volume record and some of the file entries to be invalid in a backup catalog. The use of the EXPORT command may cause a deletion. The use of the IMPORT command always causes an entry definition.

Add/Remove Volumes: The ALTER ADDVOLUMES command is used to add a volume to a file as a candidate. The ALTER REMOVEVOLUMES command is used to remove a volume from a file as a candidate.

File Extension through Suballocation: Extension causes the volume space map in the backup catalog to be invalid as well as the entry for the file.

Minimization of Catalog Mismatches: The possible catalog mismatches described above, which cause files to be wholly or partially inaccessible, are all caused by the DEFINE, DELETE, and ALTER commands, or by the extension of VSE/VSAM files or data spaces. Because DEFINE, DELETE, and ALTER are always known to you, a backup copy of the catalog can be made every time one of these commands is used. Therefore, the only action that invalidates a backup catalog without you being aware is the extension of space. Thus, the minimization of space extension tends to minimize critical catalog changes. To prevent any VSE/VSAM object from extending, you can define VSE/VSAM objects with no secondary extent value. As long as a VSE/VSAM object does not extend, it remains totally accessible from a backup copy of the catalog.

Volume is Inaccessible

Cause of Failure

A given volume may become wholly or partially unusable because of physical damage to the volume, or because the catalog that owns the volume was restored to a state that is not synchronized with the volume.

If the problem is because of a catalog restore operation, the procedure outlined under "Catalog is Unusable" on page 146 can be used to correct the condition.

If the problem is because of physical damage to the volume, recovery depends on the availability of backup copies of the catalogs, volumes, and files.

Error Conditions

- Volume is totally unusable.
- Volume is partially unusable.

Overview

A given volume that is wholly or partially unusable can be reestablished if backup copies of the data are available. In certain cases, the current version of the data can be extracted from the unusable volume and reestablished in the system.

Recovery for Volume that is Totally Unusable

You can recover only if you have a volume backup and a catalog volume backup created at the same time (that is, at the same level), or if you have copies of the files created by the REPRO command or by the EXPORT command.

If you have backup volumes:

1. Restore the damaged volume(s).
2. Compare the LISTCAT listing from the backup level with the current LISTCAT listing for possible mismatches.
3. Use the EXPORT command or the REPRO command to move the downlevel copies recovered to temporary files.
4. Initialize the volume and reestablish nonVSAM files.
5. If there is a volume mismatch which requires the use of the EXPORT command or the REPRO command, use the DELETE command with the FORCE option to clean up the volume and to remove the volume entry from the catalog (file entries are marked unusable); then, define space on the volume.
6. Use the IMPORT command or the REPRO command to reestablish the files. These files are downlevel and any update applied after the backup was made has to be reapplied to make the file current.
7. If reestablished nonVSAM files are cataloged, delete and redefine the nonVSAM entries.

If you do not have volume backup, but you do have a file backup:

1. Initialize the volume and reestablish the nonVSAM files.
2. Use DELETE FORCE to clean up the volume of VSE/VSAM ownership and data spaces. This will also remove the volume entry from the catalog and mark file entries unusable.
3. Reestablish files
   - If copies created by the EXPORT command of VSE/VSAM files are available, use the IMPORT command to reestablish them.
   - If backup files created by the REPRO command exist, delete the unusable files and redefine them using the DEFINE command and then load the backup copies into the newly created files with the REPRO command.
   - If reestablished nonVSAM files were cataloged, delete and redefine the nonVSAM entries.

Recovery for Volume that is Partially Unusable

If the VSE/VSAM files are partially or totally unusable, but the nonVSAM files are accessible, use the above procedure. If the VSE/VSAM files are accessible, but the nonVSAM files are not, proceed as follows:

1. Recover the VSE/VSAM files on the volume using the EXPORT command.

2. Initialize the volume and reestablish the nonVSAM files.
3. Using the IMPORT command, reestablish the VSE/VSAM files.
4. If the reestablished nonVSAM files were cataloged, delete and redefine the nonVSAM entries.

Quick Recovery

There are some applications (such as online teleprocessing systems) which require that file recovery be done as quickly as possible. In this type of situation, normal VSE/VSAM recovery procedures may be too time consuming to be of much use. For quick recovery, you have to:

- Implement certain "restrictions".
- Ensure data integrity.
- Recover lost objects.

Procedure for Quick Recovery

1. Implement "restrictions":
   - Define all files so that they *cannot* acquire additional extents.
   - Allocate sufficient extents (overallocate).

   These definitions ensure that the backup catalog stays in synchronization with the files that the catalog controls.

   Explanation to "overallocate extents": The restriction that files cannot acquire additional extents does not mean that CA splits will not be allowed. As long as there is sufficient unused space in the current extent, CA splits can still occur.

   To provide sufficient space for CA splits, overallocate your extents. For example, an overallocation of 20 cylinders for any VSE/VSAM file allows that at least 20 CA splits may occur.

   Note: Overallocate for the index also, because at least one new index record will be created whenever a CA split occurs.
2. Create a backup of the catalog *whenever* any file is defined, deleted, or altered. To do so, use the REPRO command.
3. Create a backup of the compression control data set *whenever* a compressed file is defined, deleted, altered, or loaded. To do so, use the REPRO command.
4. Recover objects:

   If the catalog controlling the files is lost, do the following:
   a. REPRO the backup catalog into the existing catalog.
   b. Run VERIFY against all files controlled by the catalog.

   If a volume is lost, do the following:

   a. Restore the backup copy of the lost volume.
   b. If the volume is the catalog volume, REPRO the corresponding backup catalog into the existing catalog, and if applicable, REPRO the corresponding backup of the compression control data set into the existing CCDS.
   c. Run VERIFY against all files on this volume. (The files may also belong to other catalogs.)

      If the volume is a catalog volume, also run VERIFY against all files of this catalog (where the files may reside also on other volumes).
   d. Update restored files from journaled records.

Chapter 9. VSE/VSAM Support for SAM Files

This Chapter ...

Explains how you can prepare your existing **SAM files** and programs so as to take advantage of the functions provided by VSE/VSAM.

The chapter highlights the optional and required steps and definitions, and includes examples for loading and defining SAM ESDS files.

Overview

About SAM ESDS Files

A SAM file that can be processed within VSE/VSAM data space is called a *SAM ESDS file*.

You can move/convert your SAM files to:
- *SAM ESDS files*, or
- *VSE/VSAM ESDS files*.

Notes:

1. A SAM ESDS file is not identical to a VSE/VSAM ESDS file. For more information, refer to "Differences Between VSE/VSAM ESDS and SAM ESDS File Format" on page 179.
2. SAM ESDS files can only be created and used if the *VSE/VSAM Space Management for SAM Function* is effective. This is always true for the z/VSE environment.

SAM files that have been converted can take full advantage of the processing capabilities of VSE/VSAM. You can use SAM with VSE/VSAM for data and work files, and you can move data and programs from SAM control to VSE/VSAM control.

The *full* conversion of SAM files involves the three steps explained under "Levels of Migrating Data and Programs from SAM to VSE/VSAM Control" on page 155. You have the option to stop at any level of the conversion. Depending on the level that you implement, you can use specific VSE/VSAM functions and capabilities for processing the files.

To indicate a SAM ESDS file and provide SAM record format information, specify the RECORDFORMAT parameter in the IDCAMS DEFINE CLUSTER command for a NONINDEXED (ESDS file). During VSE/VSAM access of a SAM ESDS file, SAM records are processed according to your specifications in the RECORDFORMAT parameter. To the VSE/VSAM program, a SAM ESDS file appears as though it is in VSE/VSAM ESDS file format.

SAM ESDS files can be accessed by VSE/VSAM (ACB) access if the files are formatted with control intervals (CIs).

© Copyright IBM Corp. 1979, 2005

About the VSE/VSAM Space Management for SAM Function

VSE/VSAM offers the *VSE/VSAM Space Management for SAM Function*. The function allows you to:

- Define and process your SAM files within VSE/VSAM data space.
- Request quantities of disk space rather than absolute locations.

To define and process your SAM files in VSE/VSAM data space, you use a DTF, and SAM imperative macros (for example, OPEN, GET, PUT). You indicate your intention to use a SAM file in VSE/VSAM data space by opening a DTF that specifies a file name described in a VSE/VSAM DLBL statement. This tells OPEN that the file to be accessed is a SAM ESDS file. Then, managed-SAM OPEN retrieves file information from the VSE/VSAM catalog rather than from the VTOC.

Data is written in a format similar to a VSE/VSAM ESDS file. The control interval (CI) format is used; where the CI is the basic unit of information that is transmitted to or from a direct-access device. This format allows:

- VSE/VSAM access (through ACB) to SAM files in VSE/VSAM data space.
- Disk independence (for example, maximum DTF BLKSIZE is not limited to disk track size, but only to CI size minus 7).

You need not specify absolute extent limits for the file, because VSE/VSAM determines the location of the file.

Advantages in Using SAM ESDS Files

If you move unmanaged-SAM files into SAM ESDS files, you can take advantage of many of the functions available in VSE/VSAM, including IDCAMS commands. The following are the functions that are available after you complete the first step explained under "Levels of Migrating Data and Programs from SAM to VSE/VSAM Control" on page 155.

Dynamic Allocation

With VSE/VSAM managing your space, you can take advantage of VSE/VSAM's dynamic allocation. The allocation of file space is simpler because you do not have to specify extent limits. You need only request a quantity of space. This space is allocated when it is needed. If more space is subsequently needed, a secondary quantity is allocated.

VSE/VSAM's dynamic file capability allows you to define a file in the VSE/VSAM catalog without allocating space for it. Space is allocated at OPEN and deleted at CLOSE under control of the DLBL DISP parameter. This dynamic file capability applies to SAM ESDS files.

Simplified Job Control

This improvement is available to the SAM user through the *VSE/VSAM Space Management for SAM Function*. The information required by the system to check the location and characteristics of files is stored in the VSE/VSAM catalog. The need for DLBL statements is also removed for many of the IDCAMS commands. Because VSE/VSAM user catalogs are programmer logical units, they too are eligible for automatic assignment. Operator communications are also simplified because the operator may mount a requested volume on an available drive without the need to assign the drive.

Default Modeling

Default modeling allows you to select your own parameter defaults in place of the usual system defaults during explicit define. The ability to specify default parameters for the IDCAMS DEFINE CLUSTER command through default modeling is available for SAM ESDS files as well as for VSE/VSAM files.

Implicit File Definition

SAM ESDS files do not have to be explicitly defined (by way of the IDCAMS DEFINE command) prior to the time they are opened. An implicit define of a reusable SAM ESDS file occurs during managed-SAM OPEN if the file has not yet been explicitly defined.

Generally, when a file is implicitly defined, it may be implicitly deleted during managed-SAM CLOSE. This depends on the disposition parameters specified on the DTF or DLBL statement. For more information, refer to "Implicit Deletion of a SAM ESDS File" on page 173.

Device Independence

You do not have to be concerned with different track and cylinder sizes for various types of devices. (The DTF DEVICE and DEVADDR parameters are ignored by managed-SAM OPEN so that the file may reside on any disk device type.) Allocation sizes may be requested in terms of number of records and average record length rather than tracks and cylinders, which are device-dependent. This may be specified in the DEFINE CLUSTER command for explicitly defined files, or in the DLBL RECORDS and RECSIZE parameters for implicitly defined files. For implicitly defined files, a default secondary allocation size of twenty percent of the primary allocation size (rounded up) is assumed if none is specified.

The control interval (CI) is the basic unit of information that VSE/VSAM transmits to or from a direct-access device. Because the CI size has no relation to the track and cylinder size of a particular device, this makes the processing of files disk independent.

IDCAMS Commands

You do not need to use different utility programs to manipulate files. With the *VSE/VSAM Space Management for SAM Function* effective, you can use IDCAMS commands to print, copy, alter, delete, and move files from one system to another. For special considerations in using the commands, refer to "The IDCAMS Commands for a SAM ESDS File" on page 170.

Security and Integrity of Data

VSE/VSAM ensures the security and integrity of data through a combination of VSE/VSAM facilities:

- The share options support for SAM ESDS files.
- Password-protection to prevent unauthorized access.
- Automatic CLOSE facility for files not closed before the end of job.

Data Recovery

Data recovery is supported for a SAM ESDS file through various IDCAMS commands. As a basis for reconstruction (if the original file becomes inaccessible), you can use the EXPORT and IMPORT commands in the same way as for a

VSE/VSAM file. That is, you can create a portable copy of a SAM ESDS file by using the EXPORT command, and introduce the copy of the file into the system by using the IMPORT command

Additional Functions Available for Managed-SAM Access

In addition to the specific functions mentioned above, the following additional functions are available to facilitate processing of SAM ESDS files:

- Multiple extents and multiple volumes are supported, unless:
  - During definition of the file there was no secondary allocation size specified, or a single volume was specified, *or*
  - The program accessing the file does not support multiple extents (for example, DTFPH with EXCP access).
- A SAM ESDS file can be extended through the request in the DISP parameter of the DLBL statement.

Planning for Files

Note that all the functions described in this chapter apply also to work files, but you may find some of them as being inappropriate (for example, password-protection and data recovery).

Work Files

Automatic Space Management

Work files may need varying amounts of space for different jobs. For some jobs only a small quantity of space is needed. At other times, a great deal of space is needed.

VSE/VSAM provides automatic space management. That is, you do *not* need to:

- Ensure that enough space is available for jobs that require large quantities of space.
- Keep large amounts of space tied up. The space needed for work files can be smaller.

You can think in terms of the average size of space needed rather than the maximum size needed. This is because VSE/VSAM provides *dynamic secondary allocation*. You can make your primary allocation nearer to the average size of space needed; if more is needed, VSE/VSAM gets the necessary space by using the secondary allocation. In addition to dynamic secondary allocation, VSE/VSAM provides *dynamic primary allocation*. This allows you to define a file that does not need space until it is opened. (A file defined in this way is called a *dynamic file*.) When a dynamic file is opened, the needed space is provided by VSE/VSAM. The options available at OPEN and the disposition of the files at CLOSE depend on what you code in the DISP parameter of the DLBL statement, or what is specified in the DTF (for example: DELETFL=NO).

Partition/Processor Independence

VSE/VSAM provides for partition/processor independence through the *implicit define* or *explicit define* with the dynamic data set capability. This VSE/VSAM support eliminates the tasks of:

- Assigning different work files to different disk locations for every partition.
- Specifying those disk locations in your job control statements.

The file-ID is chosen according to the partition in which the job is running, and space is assigned as needed. Work-space can also be shared between processors. The same job can be processed in any partition of a number of different processors without conflict in the catalog. When the file is closed, it may be deleted or deallocated. The space the file occupied is reclaimed and made available for use by other files.

Disposition

The disposition of a reusable file (REUSE) can be controlled through the DLBL DISP parameter. A file can be allocated, reset, or implicitly defined at OPEN according to these specifications. Whatever you specify in the DLBL DISP parameter overrides whatever was specified in the DTF. Pertinent information from the disposition parameters and the DTF is saved for closing of the file. At that time, the file is kept, reset, deallocated, or deleted according to the disposition that is specified in the DTF and DLBL statement. When you do not specify the DISP parameter, a default is chosen according to the type of file opened or closed. The default disposition is the same as would occur for unmanaged-SAM files. For example, the default disposition for:

- DTFSD OUTPUT data file is DISP=(NEW,KEEP)
- DTFSD INPUT data file is DISP=(OLD,KEEP)
- DTFSD work file is DISP=(NEW,DELETE).
  If DELETFL=NO, then DISP=(NEW,KEEP).

For disposition parameter specifications and their results, see Figure 13 on page 32 and Figure 15 on page 35. Disposition processing for VSE/VSAM (ACB) access of a SAM ESDS file is the same as for a VSE/VSAM ESDS file.

Extending Existing SAM ESDS Files

With VSE/VSAM support you can also extend existing SAM ESDS files through the use of the DLBL DISP parameter. For example, to extend a SAM ESDS file during output processing using SAM access, DISP=OLD would position the file to end-of-file to allow for extension. Refer also to Figure 13 on page 32.

Space for extension of a file is allocated (if necessary) according to the secondary space allocation specified at definition time. SAM ESDS files are always extended in SPEED mode. See “Example 4: Define a Dynamic SAM ESDS File and Access” on page 177.

This support is not provided for SAM ESDS work file access.

Levels of Migrating Data and Programs from SAM to VSE/VSAM Control

Moving from SAM-control to VSE/VSAM-control consists of up to three steps:
Step 1: Move unmanaged-SAM data (files) into SAM ESDS files.
This allows managed-SAM access.

- **Step 2**: Change managed-SAM access programs to VSE/VSAM access programs.
  SAM ESDS files are accessible by VSE/VSAM.
- **Step 3**: Convert data (files) from SAM ESDS files to VSE/VSAM ESDS files.
  Files are accessible by VSE/VSAM only.

Depending on the VSE/VSAM functions you want to be able to use, you can simply implement only the first step, or you can complete the other steps as well.

Figure 28 illustrates the relationship of the migration levels and steps.

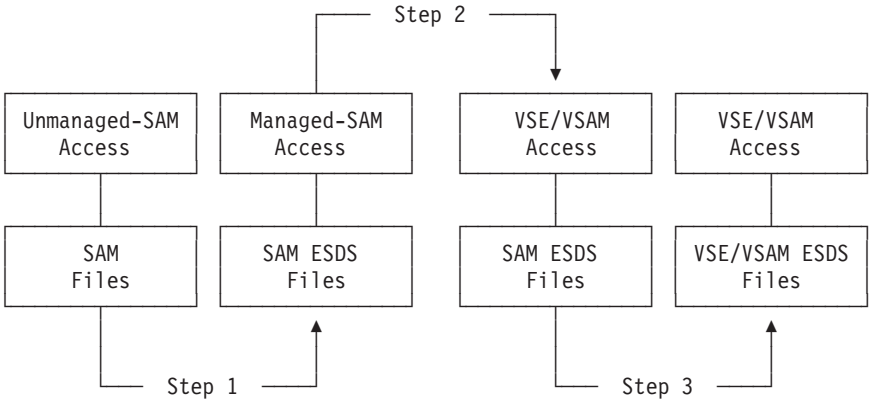


Figure 28. Migration from SAM Control to VSE/VSAM Control

Figure 29 lists the valid combinations of access modes and file types that can be used when the *VSE/VSAM Space Management for SAM Function* is effective.

File Type	Access Mode	
	VSE/VSAM Access	Managed-SAM Access
SAM ESDS	Valid	Valid
VSE/VSAM ESDS	Valid	Invalid

Figure 29. Valid Combinations of Access Methods and File Types

Functions Available at the Various Migration Levels

Step 1: Move SAM Files to SAM ESDS Files

After you completed this step, you get all the functions described under "Advantages in Using SAM ESDS Files" on page 152.

Before you can move your SAM files, you have to create SAM ESDS files as explained under "Creating a SAM ESDS File" on page 157.

Step 2: Change Managed-SAM Access Programs to VSE/VSAM Programs

After you completed this step, you get the following additional access functions (they are provided by VSE/VSAM access):

- VSE/VSAM provides a single ACB/RPL format and a single set of request macros for all file types. You can generate the ACB or the RPL by specifying the GENCB macro.
- The file can be accessed in a direct manner through access by RBAs.
- The file can be processed in a skip sequential manner or sequentially backwards.
- Access statistics are maintained; they can be displayed through LISTCAT.

- Multiple (CI) buffers may be used in support of VSE/VSAM's read-ahead capability.
- A password may be specified in the ACB so that the operator is not involved with passwords.
- The job is not canceled due to logical or physical errors as is done in SAM. Rather, a return code and error code are passed back to the user to allow diagnosis of the failure within the user's program.
- Multiple strings and chained RPL support are provided by VSE/VSAM.

Step 3: Convert SAM ESDS Files to VSE/VSAM ESDS Files

After you completed this step, you get the following additional capabilities:

- You can build alternate indexes or paths. It allows you other ways of gaining access to your files, thereby eliminating the need to keep multiple copies of the same information sorted differently for different applications.
- You can specify RECOVERY in the DEFINE CLUSTER statement. This parameter will help ensure data integrity by preformatting every CA before records are loaded into it. In case of load or extend failure, the IDCAMS VERIFY command can be used to recover data written, and your program may resume writing data from the last correctly-written data record.
- Records can be spanned (SPANNED) records, eliminating the need for very large CIs.
- You might be able to define the file with the COMPRESSED attribute to save DASD space.
- Generally, a VSE/VSAM ESDS file is portable to MVS and can be accessed by way of MVS/VSAM. See Appendix D, "Compatibility With Other Products," on page 329 for specific cases when files are not portable.

To copy data from SAM ESDS files to VSE/VSAM ESDS files, you can use the IDCAMS command REPRO; refer to page 172.

Creating a SAM ESDS File

To **create** a SAM ESDS file, you have to:

1. Set up a quantity of space (see below).
2. Define a SAM ESDS file (see below).

After you created a SAM ESDS file, you can:

- *Access* the file by using:
  - DTF and SAM imperative macros with VSE/VSAM DLBL
  - ACB and VSE/VSAM imperative macros with VSE/VSAM DLBL
  - IDCAMS commands with VSE/VSAM DLBL

  For explanations, refer to "Access to a SAM ESDS File" on page 165.
- *Delete* the file either:
  - Explicitly (through IDCAMS), or
  - Implicitly (through managed-SAM CLOSE).

    For explanations, refer to "Implicit Deletion of a SAM ESDS File" on page 173.

Setting Up a Quantity of Space

Space for a SAM ESDS file may be suballocated by VSE/VSAM from data space that was previously defined for VSE/VSAM files. You need not assign separate space for SAM ESDS files. The size and boundaries of the suballocated space are

communicated to the managed-SAM access routines at OPEN and secondary allocation time. You define this space (ideally entire volumes) in the usual way by using one or more of the following IDCAMS commands:

- "DEFINE MASTERCATALOG"
- "DEFINE SPACE"
- "DEFINE USERCATALOG"

These commands are described in the manual *VSE/VSAM Commands*.

Defining a SAM ESDS File

After sufficient space is defined, you can define a SAM ESDS file in one of two ways:

- *Explicitly*. That is, by using the IDCAMS command DEFINE CLUSTER.
- *Implicitly*. That is, by providing the required file information in the job control statements so that the file can be defined at managed-SAM OPEN.

Besides these defines, you can do the following with a file that already has been explicitly or implicitly defined and used:

extend it, or
reset it to empty, or
reuse it.

Note: If the catalog is password-protected, implicit define will request the update or higher level password of the catalog, and implicit delete will request the master password of the catalog.

The following explains explicit and implicit define.

Explicit Define Cluster (Using the DEFINE CLUSTER Command)

You define a SAM ESDS file explicitly by specifying parameters in the IDCAMS command DEFINE CLUSTER.

The following is not a complete list of the parameters that are eligible to be specified. However, for SAM ESDS files, you need to evaluate the applicability of these particular parameters:

- NAME -- Cluster level (Required parameter)
- NAME -- Data component level (Optional parameter unless you wish to request single extent primary allocation, in which case it is required)
- NONINDEXED -- (Required parameter)
- RECORDFORMAT -- (Required parameter)
- RECORDSIZE -- (Required parameter if RECORDFORMAT is in fixed format; for example, FIXUNB or FIXBLK. Optional for V, VB, or U format.)
- RECORDS or

  TRACKS or

  CYLINDERS or

  BLOCKS

  (One of these parameters is required unless a default model exists for a SAM ESDS file.)
- VOLUMES -- (Required parameter unless a default model exists for a SAM ESDS file.)

DEFINE CLUSTER Command -- Explanations of Parameters

NAME(entryname)

Specifies the file-ID of the SAM ESDS file.

For a single extent primary allocation you must specify both the cluster name and the data component name. (See "Single Extent Primary Allocation" on page 161.) Otherwise, the data component name is optional and if specified, can be any name.

Also, you can specify that a file be partition independent, or both partition independent and processor independent (see "Partition/Processor Independence Specification" on page 161.) Specifying file names at both levels (cluster and data) gives you the capability to access data under two different file IDs.

NONINDEXED

Specifies that the file defined is an ESDS file.

RECORDFORMAT(format)

Establishes a NONINDEXED file as a SAM ESDS file.

Note: This parameter is required to explicitly define a SAM ESDS file. You can specify it either at the cluster level or data component level.

format For *format*, substitute one of the following values:

Format	Abbreviation	Meaning	Type of Access
FIXUNB	F	Fixed, unblocked	Managed-SAM/VSAM
FIXBLK	FB	Fixed, blocked	Managed-SAM/VSAM
VARUNB	V	Variable, unblocked	Managed-SAM/VSAM
VARBLK	VB	Variable, blocked	Managed-SAM/VSAM
UNDEF	U	Undefined	Managed-SAM/VSAM
NOCIFORMAT	NCIF	See below	See below

FIXUNB, FIXBLK(*logicalrecordsize*), VARUNB, VARBLK, and UNDEF indicate that data records are stored in CI format and therefore are accessible and managed by VSE/VSAM.

NOCIFORMAT indicates that data is not stored in CI (VSE/VSAM) format. Therefore, DTFPH with physical I/O (EXCP) must be used to access the data records. (Do not use managed-SAM access or VSE/VSAM access.) The DTFPH method of access should only be used for local (work) files. Other SAM programs will not be able to read or write to the file (except for other programs that have been written specifically for NOCIFORMAT access; for example, EXCP). You cannot specify NOCIFORMAT together with any of the following parameters: CONTROLINTERVALSIZE, ERASE, BUFFERSPACE, EXCEPTIONEXIT, or WRITECHECK.

logicalrecordsize indicates the length of the SAM logical record. This value must always be specified when using FIXBLK format.

RECORDSIZE(average maximum) and RECORDS(primary)

When you specify the RECORDFORMAT parameter with the FIXUNB or FIXBLK subparameter, you must specify the *maximum* SAM logical block size in the RECORDSIZE(maximum) parameter.

Note: This parameter specifies the largest SAM logical block size that may be used. If a DTF is opened for OUTPUT or WORK and specifies a BLKSIZE larger than the maximum SAM logical block size allowable in the file, the OPEN fails and the job is canceled. You must be careful to specify the maximum RECORDSIZE that a system program or program product will use during explicit define of the file. If multiple system programs or program products are to use the same (work) file, the maximum RECORDSIZE should be equal to the largest record that any of the programs will use.

If the RECORDFORMAT parameter is specified as VARUNB, VARBLK, or UNDEF and the RECORDSIZE parameter is omitted, the RECORDSIZE defaults to 4089 for the average and 4089 for the maximum (that is, RECORDSIZE (4089 4089)). (Note that RECORDSIZE(maximum) is used in calculating the CI size and therefore has no meaning when NOCIFORMAT is specified.)

Whether or not you specify the NOCIFORMAT subparameter, you can use the RECORDSIZE(average) and RECORDS parameters for the suballocation of space. When using the RECORDSIZE and RECORDS parameters together, they must be consistent in units of reference (either both refer to SAM logical records or both refer to SAM logical blocks). Note that both the average and maximum record size must be specified in the RECORDSIZE parameter when one is specified.

For V, VB, or U records, the RECORDSIZE parameter is optional. For F or FB records, the RECORDSIZE parameter is required. For FB records, the RECORDSIZE must be a multiple of the SAM logical record size specified in the RECORDFORMAT parameter. For V or VB records, the maximum RECORDSIZE parameter must include room for the control information for variable length records (the record length field is four bytes and the block length field is four bytes) because the control information is part of the SAM logical block.

TRACKS | CYLINDERS | BLOCKS(primary)
The rules involved in the use of these parameters are the same for a SAM ESDS file as for a VSE/VSAM file. For information concerning the TRACKS, CYLINDERS, and BLOCKS parameters, search the index of the manual *VSE/VSAM Commands*; they exist, for example, for the "DEFINE ALTERNATEINDEX" command.

VOLUMES(volser)
Specifies the volume(s) to contain the SAM ESDS file. Every volume that you specify must be owned by the catalog that is to own the SAM ESDS file. If not specified during define of a SAM ESDS file, VSE/VSAM picks a set of volumes for you if you have a default model (DEFAULT.MODEL.ESDS.SAM) defined. For information on the VOLUMES parameter, search the index of the manual *VSE/VSAM Commands*; they exist, for example, for the "DEFINE ALTERNATEINDEX" command.

Additional Considerations

- The RECORDFORMAT attributes can be modeled by means of the MODEL parameter.
- You should specify REUSE when a SAM ESDS file is used mainly for work files. You should additionally specify NOALLOCATION in the DEFINE CLUSTER command to provide the dynamic file capability to work files.

- Do not specify RECOVERY. (VSE/VSAM defaults to SPEED for a SAM ESDS file.) You cannot build an alternate index or define a path over a SAM ESDS file.

Note: For work files, a zero retention period is the default and is normally appropriate to avoid operator communications during a subsequent OPEN if the file was not deleted at CLOSE.

Single Extent Primary Allocation

NAME(DOS.WORKFILE.SYSentryname) - data component level

Some programs that access data through DTFPH with EXCP may require that disk space for the file be allocated as a single extent. You can specify that you want the primary space allocated as a single extent by specifying the data component name as above. (Normally, VSE/VSAM may obtain an allocation in as many as five extents.) The cluster name is still chosen in the same manner as before, but DOS.WORKFILE.SYS *must* prefix the *data* component name to ensure that space is allocated within a single extent.

VSE/VSAM will deny the allocation request if it cannot obtain the primary allocation in a single extent.

Partition/Processor Independence Specification

NAME(%entryname) - Partition independent file-ID.

You specify a partition-unique file-ID by using the prefix "%" in the cluster name parameter of the DEFINE CLUSTER command. (The file-ID is limited to twenty-seven characters in this case.)

If your system also has the *Interactive Computing and Control Facility* (ICCF) installed, you are allowed only one partition-independent file for every ICCF real-partition. (ICCF pseudo-partitions do not have unique partition IDs, so there can be only one partition-independent file per partition.)

NAME(%%entryname) - Partition and processor independent file-ID.

To specify both a partition-unique and processor-unique file-ID together with a single extent primary allocation, the cluster name must be prefixed with "%%" (the file-ID is limited to twenty-seven characters in this case) and the data component name must be prefixed with "%%DOS.WORKFILE.SYS" (the file-ID is limited to an additional eleven characters in this case).

Implicit Define Cluster

Programming Interface Information

A SAM ESDS file can be defined implicitly through managed-SAM OPEN when TYPEFLE=OUTPUT or TYPEFLE=WORK is specified in the DTF. An implicit define cluster occurs as a result of the following two conditions:

- The SAM ESDS file (to be opened through the DTF and written to) is currently undefined in the VSE/VSAM catalog, or the characteristics of the file were not compatible with the DTF and the file has been implicitly deleted by OPEN.
- Enough information has been provided for the implicit define to occur. VSE/VSAM gathers the necessary information from three sources:
  - It makes several assumptions about the file.
  - It extracts information from the DTF specifications.

– It extracts information from the job control statements.

The following explains the assumptions made by VSE/VSAM, and the information gathered by VSE/VSAM from the DTF and from job control statements.

Assumptions Made by VSE/VSAM

For an implicit define, VSE/VSAM always makes the following assumptions:

Parameter
Assumption

NONINDEXED
An ESDS file is defined.

NONSPANNED
The maximum length of a SAM logical block must not be greater than the CI size minus 7.

NOWRITECHECK
VSE/VSAM access will not check for correct data transfer for records written to the file. (In managed-SAM access, checking for correct data transfer is controlled by the DTF VERIFY=YES or NO parameters. The NOWRITECHECK specification has no meaning for the managed-SAM user. It is used only during VSE/VSAM access.)

REUSE
It is possible for a user to reset an already existing file back to empty and reuse it.

SHAREOPTIONS(1 3)
Either any number of users are permitted for input processing, or one user is permitted for output processing.

SPEED
Direct access storage is not preformatted.

SUBALLOCATION
VSE/VSAM data space for the file was previously defined and a primary allocation is suballocated at define.

UNORDERED
The volumes need not be used in the order specified in the EXTENT job control statements or the default model if EXTENT statements are omitted.

USECLASS(0 P)
The file occupies class-0 data space.

Information Obtained from the DTF

VSE/VSAM extracts information either from the:
- DTFSD or
- DTFPH MOUNTED=SINGLE

to determine the following:
- CI size.
- Length of the maximum VSE/VSAM logical record (SAM logical block).
- Record format of the records in the file.
- SAM logical record size for FIXBLK.

From DTFSD Specifications (for Data files):

- CI size — derived from the CISIZE=nnnnn parameter. VSE/VSAM rounds this value up to a valid CISIZE before defining the file. If zero or no value was specified, VSE/VSAM chooses a CI size. If IOAREA2 is specified and CI size is not specified, VSE/VSAM attempts to choose a CI size that ensures that at least 2 logical blocks will fit into a CI. If no CI size was specified, VSE/VSAM computes the size on the base of the maximum record size.

- Maximum record size — derived from the BLKSIZE=nnnn parameter. This value (minus 8 for data OUTPUT DTFs) specifies the file's maximum RECORDSIZE.
- Record format — derived from the RECFORM=xxxxxx parameter. Specifies the RECORDFORMAT of the file. If the RECFORM=FIXBLK, the SAM logical record size is derived from the DTF RECSIZE=nnnnn parameter.

From DTFSD Specifications (for Work Files):

- CI size — derived from the CISIZE=nnnnn parameter. VSE/VSAM rounds this value up to a valid CISIZE before defining the file. If zero or no value was specified, VSE/VSAM chooses a CI size. If no CI size was specified, VSE/VSAM computes the size on the base of the maximum record size.
- Maximum record size — derived from the BLKSIZE=nnnn parameter. This value specifies the file's maximum RECORDSIZE.
- Record format — derived from the RECFORM=xxxxxx parameter. Specifies the RECORDFORMAT of the file; FIXUNB and UNDEF are the only valid subparameters that you can specify for work files.

From DTFPH MOUNTED=SINGLE (for Disk):

- CI size — derived from the CISIZE=nnnnn parameter If a non-zero value is specified, VSE/VSAM rounds this value up to a valid CISIZE before defining the file. Specifying zero is the same as not specifying a CI size. In this case, VSE/VSAM indicates that the file is non-CI format; it is accessible only by EXCP (not by VSE/VSAM or managed-SAM).
- Maximum record size —
  - If CI format, maximum equals the DTF CISIZE minus 7.
  - If non-CI format, this parameter does not apply.
- Record format —
  - If CI format, the RECORDFORMAT is UNDEF.
  - If non-CI format, the RECORDFORMAT is NOCIFORMAT.

Note: DTFPH (with a CISIZE of zero specified or no CISIZE specified) is the only possible way you can implicitly define a non-CI format file. Also, if a nonzero value is specified for the CISIZE parameter, it must be greater than seven in order to choose a valid maximum record size.

Information Obtained from the Job Control Statements

Programming Interface Information

Certain parameters in the VSE/VSAM DLBL and EXTENT job control statements provide the information that VSE/VSAM needs to implicitly define a file.

Information from the DLBL Statement: The DLBL statement provides the following information for implicit define:

- file-ID — This parameter provides the unique name associated with the file.

  To request single extent allocation through an implicit define, DOS.WORKFILE.SYS must prefix the file-ID.

  A partition/processor unique file-ID may also be specified. In this case the DLBL file-ID must be specified with a prefix of "%" (partition-unique) or "%%" (partition- and processor-unique) with a limit of twenty-seven characters. For both partition/processor uniqueness and single extent primary allocation, the DLBL file-ID prefix may be specified as "%%DOS.WORKFILE.SYS" (with a limit of eleven additional characters).

If your system also has Interactive Computing and Control Facility (ICCF) installed, you are allowed only one partition-independent file for every ICCF real-partition. ICCF pseudo-partitions do not have unique partition IDs, so there can be only one partition-independent file per partition.

- date — This parameter indicates either the retention period in days or the actual expiration date. If this parameter is not present the normal default applies.

  Note: For work files, specify a zero (retention period) to avoid operator communications during a subsequent OPEN if the file was not deleted at CLOSE.
- CAT=filename — This parameter indicates the catalog that owns the file. If this parameter is not present the normal default applies.
- RECORDS=(primary,secondary) — This parameter designates the number of SAM logical records for allocation purposes. This parameter must be specified if the "number of tracks or blocks" parameter is omitted from the EXTENT statement. If no secondary amount is specified, twenty percent of the primary allocation is assumed. Zero may be specified for the secondary amount. If RECORDS is specified, RECSIZE=n must also be specified.
- RECSIZE=n — This parameter indicates the average SAM logical record size; it must be specified together with the RECORDS=(primary,secondary) parameter. The value n is only used for space calculation; it does not influence the CI size.

  Note: You may alternatively specify the average SAM logical *block* size in the RECSIZE parameter. If you do this, you should also specify the number of SAM logical *blocks* in the RECORDS parameter.

Information from the EXTENT Statement: The EXTENT statement provides the following information for implicit define:

- Volume serial number — This indicates the volume that this file resides on. There must be one EXTENT statement for every volume that the file is eligible to reside on. If EXTENT statements are specified, this parameter is required on every EXTENT statement.
- Number of tracks or blocks (specified in the first EXTENT statement if multiple EXTENT statements are specified) — This indicates the number of tracks (CKD) or blocks (FBA) to be allocated to this file. A secondary allocation size of twenty percent of the primary allocation size (rounded up) is assumed. Whether it is tracks or blocks is determined by the device type of the volume serial number specified. This parameter is ignored on subsequent EXTENT statements, or if the RECORDS/RECSIZE parameters are specified in the DLBL.

Note: The EXTENT statement is not required for implicit define if a default model for a SAM ESDS file was previously defined (providing VOLUME information) and RECORDS/RECSIZE are specified in the DLBL statement (providing allocation information). When an implicit DEFINE is done, only the VOLUMES parameter is allowed to be modeled.

End of Programming Interface Information

Resetting and Reusing a Previously-Defined File

You can specify that a file is to be reset and reused by specifying the NOALLOCATION parameter together with the REUSE parameter in the DEFINE CLUSTER command. This specification indicates that data space is not to be suballocated to the file at DEFINE, but that it is to be suballocated as needed. This type of file is called a *dynamic file*.

A file may be implicitly defined at OPEN and implicitly deleted at CLOSE. However, performance is not as good as for an explicitly defined dynamic file.

Using a SAM ESDS File

Access to a SAM ESDS File

Managed-SAM access to a SAM ESDS file is provided so that you can:
- Open it through DTF
- Access it through the SAM imperative macros
- Close it through DTF

Support is also provided for DTFPH and EXCP access; in this case, it is space management support only. The data formats that can be written and read by the EXCP program are entirely under control of the EXCP program itself.

Dynamic secondary allocation is supported according to the access method or EXCP program's constraints and the allocation sizes for the file contained in the VSE/VSAM catalog.

Managed-SAM Access: Differences to (Unmanaged) SAM Access

Considerations Relating to DEFINE CLUSTER Specifications

RECORDSIZE

If the SAM ESDS file is defined *explicitly*, the maximum RECORDSIZE parameter is an important consideration. At OPEN, this maximum record size is compared to the BLKSIZE parameter in the DTF (for TYPEFLE=OUTPUT or WORK only). If the DTF BLKSIZE (minus eight for output) is greater than the maximum record size in the file's catalog entry, VSE/VSAM denies the OPEN and cancels the job. VSE/VSAM does not allow you to write a larger SAM logical block than the maximum VSE/VSAM record size specified during definition. If the SAM ESDS file is defined *implicitly*, the maximum VSE/VSAM record size for define is determined from the DTF BLKSIZE parameter so that the file's catalog entry is consistent.

If the DLBL RECORDS/RECSIZE parameters are used for allocation parameters (during implicit define), the RECSIZE parameter specifies the average record size in the file's catalog entry. VSE/VSAM DEFINE does not allow the average record size (from the DLBL RECSIZE parameter) to be greater than the maximum record size (from the DTF BLKSIZE parameter). Therefore, if the average record size is larger than the maximum record size, a VSE/VSAM implicit define sets the average RECSIZE equal to the maximum RECSIZE, and the RECORDS value is increased by the same factor that the RECSIZE was decreased by.

RECORDFORMAT

The record format specified during the explicit define of a SAM ESDS file need not match the DTF RECFORM specification. The only exception is that VSE/VSAM does not allow a non-DTFSD to access a NOCIFORMAT SAM ESDS file. Either the file is implicitly redefined or the job is canceled.

VOLUMES

The volume(s) specified during define determine the candidates eligible for file allocation. If EXTENT statements are specified with symbolic units

during access to the file, there must be an EXTENT statement and a corresponding ASSGN statement for every volume specified during define, even if all of the volumes are not written to or read from. However, EXTENT statements with symbolic units are not required and should not be specified.

SHAREOPTIONS

The share options specified during define affect the sharing characteristics of the file. For example, if SHAREOPTIONS(1) is specified for a SAM ESDS file (allowing one output or many input users), and if you are updating the file and then want to open the file for input to read it back in, you must close the file. Otherwise, the INPUT OPEN will be denied due to the SHAREOPTIONS specification.

If you want to allow multiple INPUT with UPDATE users to access a SAM ESDS file (in conjunction with the DTF HOLD=YES parameter, for example), explicitly define the file with SHAREOPTIONS(3) to allow concurrent OPENs for update (that is, OUTPUT). VSE/VSAM does not support SHAREOPTIONS(4) for an ESDS (SHAREOPTIONS(4) is treated as SHAREOPTIONS(2) during OPEN).

Regardless of the SHAREOPTIONS specification, if you have a file open for load or extension, all other attempts to open that file are denied for reasons of data integrity. Conversely, you would be denied access if you attempted to open a file for load or extension and another user had already opened that file. (A *software end-of-file* - SEOF - does not normally exist until a file is closed. If concurrent access was not denied there would be a chance that an input user would read past the end of the file.)

Considerations for Access to Files

Considerations for all Types of Files: Many of the considerations on CI format that relate to unmanaged-SAM on FBA devices are also considerations for managed-SAM access (on both FBA and CKD devices). For example, if an I/O error occurs during access, it concerns an entire CI of information rather than a single logical block. Also, logical blocks are not necessarily written to a device until a CI is full (refer also to the description of the PWRITE parameter of the "DTFSD" macro in the manual *z/VSE System Macros Reference*.

Empty Files: VSE/VSAM does not distinguish between a file that:

- Has just been defined (empty and never written to),
- Has been opened and closed with no records written into it, or
- Has been loaded but deallocated or reset at CLOSE.

A file in any of these states is considered empty (that is, the high-used RBA is zero). In any of these cases, if the file is opened for input through DTFSD TYPEFLE=INPUT, the OPEN will be successful and control will be passed to the EOFADDR on the first GET. This DTF OPEN is actually simulated because VSE/VSAM OPEN (ACB) will not open an empty file for input. However, this is transparent to the DTFSD user.

If other DTF types (such as DTFPH) are opened for INPUT on an empty file, VSE/VSAM cannot simulate the end of file condition. This OPEN cannot be allowed because the file has not been opened by VSE/VSAM and the file extents have not been located. (The file may not even be allocated in the case of a dynamic file.) Therefore, such an OPEN will be cancelled if the file is empty.

Assignments and Files Ignored: If EXTENT statements with symbolic units and ASSGN statements are used, and if any one (or more) of the assignments is ignored (IGN), then the entire file is ignored. That is, the DTF is not opened and DTF+X'10', bit 2 (X'20') will be set.

Disk-Independence: In general, you should attempt to be as disk-independent as possible. You should make no assumptions about the track size (or CI size), the size or the number of extents or even the number of volumes that the file will reside on. You should not attempt to choose a BLKSIZE that will maximize disk utilization because CI format is used and also, your program cannot know what disk device type will be used for the file before OPEN. It is better to use a smaller BLKSIZE that will be reasonable for any disk device type -- it enables to process on any disk device type. You should not assume that a particular symbolic unit will be used. This will allow you to take advantage of VSE/VSAM's job control simplifications. Note that (unmanaged) SAM now provides for disk independence by ignoring the DTF DEVICE= parameter during OPEN.

GETVIS Space: Sufficient GETVIS space must be provided for managed-SAM access; enter the specifications in the SIZE parameter of the EXEC job control statement, or in the SIZE job control command. The partition GETVIS area must contain at least 40KB for the VSE/VSAM catalog, plus 10KB for every SAM ESDS file, plus storage for the CI buffer for every SAM ESDS file.

Work Files:

- The format of NOTE/POINT IDs for a managed-SAM CKD file is similar to unmanaged SAM FBA NOTE/POINT ID format. That is, for all devices, the managed-SAM NOTE/POINT ID format is CCCN rather than (as for unmanaged SAM) CCHR for CKD and CCCN for FBA. Therefore, you should not generate or modify a NOTE/POINT ID. Also, do not move or modify the DTF between OPEN and CLOSE.
- The DELETFL=NO parameter of DTFSD TYPEFLE=WORK is determined at OPEN. Modifying this indicator after OPEN will have no effect on the CLOSE disposition. Note that DLBL DISP specification overrides the DTF DELETFL indicator. If there are any other DTFs or ACBs currently open for this file at CLOSE, the file is not deleted. If the DTF is not closed by the end of job step, automatic CLOSE attempts to close the file.
- Files accessed through DTFSD TYPEFLE=WORK are normally reset at OPEN. If you wish to read a file using a work file DTF, specify DISP=OLD in the DLBL to avoid losing the data due to reset.

Using SAM ESDS Files: Restrictions

Device-Dependent SAM Functions

The following device-dependent SAM functions are not supported:

- Split cylinders
- FEOVD (ignored)
- CNTRL (ignored)
- Subsetting of the input file through EXTENT statement specifications.

SAM ESDS Files

The following restrictions apply:

- SAM ESDS files are limited to 16 extents per volume, unless they are explicitly defined as non-reusable (NOREUSE).
- DSF (data secured file) is not supported (it is ignored) in the VSE/VSAM DLBL control statement. (VSE/VSAM password-protection may be used.)

- SAM ESDS files are not portable and cannot be imported (through IMPORT) to MVS SAM or VSE/VSAM.
- IJSYSxx file restrictions:
  - The only system data file that is supported is SYSLNK (IJSYSLN). The job is canceled if any other system data files are specified at OPEN.
  - System work files (IJSYSnn) are supported unless restricted by the program accessing the system work file.
- Some system programs or program products may have restrictions on the use of managed-SAM files. (For example, the files may be limited to a single extent, or managed-SAM files may not be supported.) Please consult the appropriate VSE/VSAM or Program Product publication for planning and support considerations.

DTF Specifications

Programming Interface Information

The following restrictions apply to specifications in the DTF:

- User labels are not supported. The LABADDR specification of the DTF is ignored.
- CISIZE or BLKSIZE *override* of the DTF is not supported in the VSE/VSAM DLBL job control statement (although the VSE/VSAM catalog entry CISIZE does override the DTF).
- Because managed-SAM records are in CI format, SAM spanned records are not supported. VSE/VSAM maximum record size (32KB minus 7) is not limited by the device track size. The job is canceled if RECFORM=SPNUNB or RECFORM=SPNBLK is specified in the DTF.

End of Programming Interface Information

DTFPH Specifications

Programming Interface Information

The following restrictions apply to the use of DTFPH:

- A file created with DTFPH with CISIZE=0 is not supported by managed-SAM request macros (GET,PUT, and so on). That is, the file can only be read with DTFPH and EXCP. The managed-SAM request macro routines support CI format only. The job is canceled if a non-DTFPH OPEN is issued against a NOCIFORMAT SAM ESDS file (unless the file can be implicitly deleted and defined by OPEN).
- Conversely, if a file is created with DTFSD and is to be read through DTFPH with EXCP, the EXCP routine must support it in CI format. If the DTFPH is a “version 3 DTF”, OPEN stores the CISIZE in the version 3 extension and the OPEN is successful. You must reference this CISIZE when you read the file. If the DTFPH is not a version 3 DTF, OPEN has no means of indicating the CISIZE and the job is canceled.
- There is no way to restrict a DTFPH EXCP user from opening a password-protected SAM ESDS file for input (requiring a read password) and then writing to the file. A DTFPH user may also violate SHAREOPTIONS integrity protection in this same manner.

End of Programming Interface Information

VSE/VSAM Access of SAM ESDS Files: Considerations

Programming Interface Information

VSE/VSAM access of a SAM ESDS file processes SAM logical records. It uses the RECORDFORMAT information in the catalog to block SAM logical records into SAM logical blocks and de-block SAM logical blocks into SAM logical records. Therefore, it is important that the RECORDFORMAT information in the catalog matches the actual SAM record format of the data.

The valid SAM logical record formats are:
- Fixed unblocked
- Fixed blocked (logical record size)
- Variable unblocked
- Variable blocked
- Undefined

SAM access (through DTF) of V or VB records returns the RL (record length field) at the beginning of the record. VSE/VSAM access (through ACB) does not return it. Correspondingly, for a PUT for V or VB records, no RL should be at the beginning of the record when it is passed to VSE/VSAM because VSE/VSAM prefixes the RL. A program using VSE/VSAM access (through ACB) for sequential processing can process a VSE/VSAM ESDS file or a SAM ESDS file.

Differences between the VSE/VSAM access of a VSE/VSAM ESDS file and the VSE/VSAM access of a SAM ESDS file are:
- VSE/VSAM always loads and extends a SAM ESDS file in SPEED mode.
- VSE/VSAM does not build an alternate index over a SAM ESDS file.
- VSE/VSAM does not support path entries over a SAM ESDS file.
- VSE/VSAM does not support VSE/VSAM SPANNED records for a SAM ESDS file.

The following applies to VSE/VSAM access of a SAM ESDS file:
- TCLOSE and ENDREQ do not imply TRUNC or RELSE. VSE/VSAM continues processing from the last SAM logical record.
- For direct requests or POINT, the ARG parameter of the RPL always specifies the RBA (relative byte address) of the SAM logical record. The RL (record length) and the BL (block length) fields are not included; however, they are accounted for by VSE/VSAM. On a direct retrieval, you must supply to VSE/VSAM the same RBA as returned during a VSE/VSAM load of a SAM ESDS file.
- For both the RPL and control block manipulation macros, RECLEN is always the SAM logical record length. It is not the SAM block length (that is, VSE/VSAM record length), and does not include any RL or BL fields.
- When issuing a SHOWCB or TESTCB macro, the following apply:
  - NLOGR refers to logical records. For blocked record files, NLOGR could be greater than the number of VSE/VSAM records.
  - NRETR and NUPDR have similar meanings; they are the number of retrieved and updated SAM logical records, respectively.

When a SAM ESDS file is extended through managed-SAM access, managed-SAM always starts with a new CI. When a SAM ESDS file is extended through

VSE/VSAM access, VSE/VSAM attempts to continue storing records into the last CI of the file. Additionally, the number and method of blocking records in a SAM logical block may differ between managed-SAM access and VSE/VSAM access. The following parameters will cause differences:

- Logical record size comes from the LRECL in "RECORDFORMAT".
- Block size comes from the maximum record size in "RECORDSIZE".
- Average record size in "RECORDSIZE" is only used together with the "RECORDS" parameter.

End of Programming Interface Information

The IDCAMS Commands for a SAM ESDS File

The following lists only those commands (and parameters) that need *special consideration* when used with the *VSE/VSAM Space Management for SAM Function*. The commands are listed in alphabetical order. For the complete set of the parameters available with the commands, see the manual *VSE/VSAM Commands*.

ALTER Command
entryname/password
BUFFERSPACE(size)
ERASE | NOERASE
EXCEPTIONEXIT(mname)
WRITECHECK | NOWRITECHECK

The ALTER command is used to change attributes in catalog entries. The subparameters and their use for a SAM ESDS file are explained here.

entryname/password
is a required parameter that names the SAM ESDS file to be altered and supplies its master password if it is password-protected. For a NOCIFORMAT SAM ESDS file, the specification of BUFFERSPACE, ERASE, EXCEPTIONEXIT (except as a subparameter of NULLIFY), or WRITECHECK causes the ALTER command to terminate.

BUFFERSPACE(size)
specifies the minimum space to be provided for buffers. For a NOCIFORMAT SAM ESDS file, the specification of BUFFERSPACE causes the ALTER command to terminate.

ERASE | NOERASE
specifies whether the SAM ESDS file is to be erased when its entry in the catalog is deleted. For a NOCIFORMAT SAM ESDS file, the specification of ERASE causes the ALTER command to terminate.

EXCEPTIONEXIT(mname)
specifies the name of the user module to be given control when an exception occurs during the processing of the SAM ESDS whose entry is altered. For a NOCIFORMAT SAM ESDS file, the specification of EXCEPTIONEXIT causes the ALTER command to terminate. (EXCEPTIONEXIT can be specified as a subparameter of NULLIFY.)

WRITECHECK | NOWRITECHECK
specifies whether to check the data transfer of records written in the SAM ESDS through VSE/VSAM (ACB) access. For a NOCIFORMAT SAM ESDS file, the specification of WRITECHECK causes the ALTER command to terminate.

DEFINE CLUSTER Command

For the applicable DEFINE CLUSTER parameters, see "Explicit Define Cluster (Using the DEFINE CLUSTER Command)" on page 158.

DELETE Command

You can use the DELETE command as described in the manual *VSE/VSAM Commands*, except that the ERASE parameter is not valid for a NOCIFORMAT SAM ESDS file.

An implicitly defined SAM ESDS file may be deleted by way of the DELETE command in the same manner as an explicitly defined SAM ESDS file. Refer also to "Implicit Deletion of a SAM ESDS File" on page 173.

EXPORT Command

If you are exporting a CI-format SAM ESDS file, VSE/VSAM treats it as an ESDS file. If you attempt to export a NOCIFORMAT SAM ESDS file, VSE/VSAM issues an error message and terminates the command.

You cannot use a SAM ESDS file as the portable file (OUTFILE parameter).

IMPORT Command

IMPORT provides full import support for those SAM ESDS files which can be exported. When attempting to import a SAM ESDS file into a predefined empty file, IMPORT ensures that the exported file and the predefined file have fully consistent RECORDFORMAT parameter values and that the maximum record size of the predefined file is not less than that of the file originally exported. Any mismatch causes an error message and command termination.

LISTCAT Command

You can display space for a SAM ESDS file by specifying LISTCAT SPACE. You can display all files that have been defined for a particular catalog by using the LISTCAT command; this includes all SAM ESDS files defined either explicitly or implicitly.

The ATTRIBUTES portion of LISTCAT output is modified as follows for ESDS:

- CISIZE indicates 0 if RECORDFORMAT(NOCIFORMAT) was specified on either an explicit or implicit define.
- CI/CA indicates 0 if RECORDFORMAT was specified on either an explicit or implicit define.
- SAMLRECL indicates the SAM logical record length (listed for SAM ESDS files only). This value is the user-supplied record length for FIXBLK SAM files and is zero for all other record format SAM files.
- RECORDFORMAT indicates the SAM record format (listed for SAM ESDS files only). FIXBLK, FIXUNBLK, VARBLK, VARUNBLK, UNDEFINED, or NOCIFMT are the possible values for this attribute.
- IMP-DEFINE is listed if the SAM ESDS file has been implicitly defined; otherwise EXP-DEFINE is listed (applies to SAM ESDS files only).
- SAMDATASET is listed if the ESDS is a managed-SAM file; otherwise VSE/VSAMDATASET is listed.

The Statistics Group (data) is listed for a SAM ESDS file. However, it should be noted that these statistics are maintained during VSE/VSAM access only and not during managed-SAM access. For more information on the statistics, refer to the manual *VSE/VSAM Commands* under "Interpreting LISTCAT and LISTCRA Output."

PRINT Command

You can print a CI-format SAM ESDS file by way of managed-SAM access or VSE/VSAM access. The output is always SAM logical records. You cannot print a NOCIFORMAT ESDS file through either managed-SAM or VSE/VSAM access.

For managed-SAM access:

- Specify the ENVIRONMENT parameter.
- The output format is the same as unmanaged SAM (no RBA display, record length field at the beginning of the record for format V and VB).
- The SKIP and COUNT parameters can be used and the value always indicates the number of SAM logical records to be skipped or listed.

(For an example of printing a SAM ESDS file by retrieving the SAM logical records with managed-SAM, see "Example 4: Define a Dynamic SAM ESDS File and Access" on page 177.)

For VSE/VSAM access:

- Omit the ENVIRONMENT parameter.
- The output format is a VSE/VSAM ESDS file (RBA display, no record length field at the beginning of format V and VB RECORDS. VSE/VSAM uses the RECORDFORMAT information recorded in the catalog to determine the SAM record format for access). Note that the record or block size and format has not been changed through DTF-ACCESS.
- The SKIP and COUNT parameters can be used and the value always indicates the number of SAM logical records to be skipped or listed.
- The FROMADDRESS and TOADDRESS parameters can be used. (The RBA value for FROMADDRESS must be the exact beginning of a SAM logical record.)

REPRO Command

CI-format SAM ESDS files can be used as input or output files in a REPRO command wherever SAM files or VSE/VSAM ESDS files are currently allowed. (Do not specify a NOCIFORMAT SAM ESDS as an input or output file.) You can use the REPRO command to convert an unmanaged-SAM file to a SAM ESDS file by using the following specifications:

- INFILE(dname,ENVIRONMENT(subparameters))

  Indicates the unmanaged SAM file to be used as the input file.
- OUTFILE(dname/password ENVIRONMENT(subparameters))

  Indicates the CI-format SAM ESDS to be used as the output file. If the output file is a managed-SAM file that is to be created by way of managed-SAM access, and it has not been previously defined, it will be implicitly defined if the job control statements meet the requirements of implicit define.
  - For both the INFILE and OUTFILE parameters, *dname* specifies the *filename* of the DLBL job control statement that identifies the file to be copied. The ENVIRONMENT parameter is not always required. Coding the ENVIRONMENT (...) parameter instructs IDCAMS to use SAM access, that is,

access through a DTF control block. Without the ENVIRONMENT parameter VSAM access will be used (ACB). The ENVIRONMENT parameter is required in one of the following cases:

- To access an unmanaged SAM file
- To allow an implicit definition of an output file

password is not allowed for SAM access.

- FROMADDRESS(address) TOADDRESS(address)

  You can specify FROMADDRESS and TOADDRESS for VSE/VSAM access (not managed-SAM access). The RBA value for FROMADDRESS must be the exact beginning of a SAM logical record.
- SKIP(count) COUNT(count)

  You can specify SKIP and COUNT (for both VSE/VSAM and managed-SAM access) and the value always indicates the number of SAM logical records to be skipped or copied.

VERIFY Command

If the VERIFY command is executed on a CI-format SAM ESDS file, you can discover whether the file was successfully closed (warning messages are issued), but you cannot cause the end-of-file indicator in the catalog entry to be updated. This is because a SAM ESDS file is always loaded and extended in SPEED mode. A SAM ESDS file cannot be accessed for input by VSE/VSAM unless it was successfully closed after initially loaded. (If the file is accessed for input by managed-SAM without closed, an OPEN in a subsequent job step will be successful and the first GET will cause the user to be sent to the EOFADDR routine.) The file can only be accessed by VSE/VSAM up to the data written by the last successful CLOSE if extended. After extension, a SAM ESDS file can be accessed by managed-SAM even if the CLOSE was unsuccessful; however, the file may not terminate with an SEOF.

The VERIFY command terminates due to an OPEN error if it is executed on a NOCIFORMAT SAM ESDS file.

Implicit Deletion of a SAM ESDS File

Programming Interface Information

An implicitly defined SAM ESDS file may be deleted by way of the DELETE command in the same manner as an explicitly defined SAM ESDS file.

You can use the "DELETE" command as described in the manual *VSE/VSAM Commands*, except that the ERASE parameter is not valid for a NOCIFORMAT SAM ESDS file.

An implicit delete of a SAM ESDS file occurs if all the following conditions are true for any of the following cases:

Case 1

During OPEN of DTF (implicit delete followed by implicit define)

- The catalog entry has been implicitly defined.
- The DTFSD maximum logical block size exceeds the VSE/VSAM catalog maximum RECORDSIZE of the SAM ESDS file or the RECORDFORMAT of the file is NOCIFORMAT.

- DTFSD TYPEFLE=OUTPUT, WORK, or WORKMOD.
- The file is unexpired and the operator has responded “delete” to message 4233A EQUAL FILE-ID IN CATALOG, or the file is expired.
- DISP=OLD is not specified.

Case 2

During CLOSE of DTF

- The catalog entry has been implicitly defined.
- DISP=(,delete)

  Note: The job control statement overrides the DTF.

Case 3

During CLOSE of DTF

- The catalog entry has been implicitly defined.
- DISP=(,date)

  Note: The job control statement overrides the DTF.
- The expiration date has passed.

In all cases, if another user has the same file open for access, the file is not deleted.

End of Programming Interface Information

Sample Programs and Job Streams

- Example 1 loads a SAM ESDS file by way of managed-SAM access (source code).
- Examples 2, 3, and 4 use this program assuming that it is cataloged under the phase name SDOUTPUT.

Example 1: Load a SAM ESDS File by Way of Managed-SAM Access

```
Col.       Col.    Col.                                              Col.
1          10      16                                                72

SDOUTPUT   START   X'200078'
           BALR    2,0
           USING   *,2
           OPEN    SDOUT,PRINT
           LA      5,1             INITIAL COUNT TO 1
           L       6,MAXRCDS       LOAD NO. OF RECORDS TO WRITE
LOOP       CR      5,6             WRITTEN LAST RECORD YET
           BH      CLOSE           YES
STORE      ST      5,RECNO         NO, STORE RECORD NUMBER
           CVD     5,DWB           CONVERT KEY TO DECIMAL
           UNPK    NUM(15),DWB(8)  UNPACK KEY
           TM      UNPKSIGN,X'10'   SEE IF NUMBER WAS NEGATIVE
           BO      NEG1             YES, NEGATIVE
           MVI     SIGN,C'+'        MAKE OUTPUT SHOW POSITIVE
           B       CONTINUE
NEG1       MVI     SIGN,C'-'        MAKE OUTPUT SHOW NEGATIVE
CONTINUE   OI      UNPKSIGN,X'F0'   MAKE LAST BYTE A NUMBER
           PUT     PRINT           PRINT KEY
           PUT     SDOUT,WORKAREA  PUT FROM WORKAREA
           LA      5,1(5)          INCR RECORD NO.
           B       LOOP            GO BACK
CLOSE      CLOSE   SDOUT,PRINT     CLOSE THE FILE
           EOJ
           EJECT
*
SDOUT      DTFSD   BLKSIZE=2008,[1]                                  X
                   DEVADDR=SYS007,[2]                                X
                   IOAREA1=OUTPUT1,                                   X
                   DEVICE=2314,[3]                                   X
                   RECFORM=FIXBLK,[4]                                X
                   RECSIZE=80,                                        X
                   TYPEFLE=OUTPUT,[5]                                X
                   WORKA=YES
           EJECT
PRINT      DTFDI   DEVADDR=SYSLST,                                    X
                   IOAREA1=IOAREA,                                    X
                   RECSIZE=17
*
           EJECT


DWB        DC      D'0'            USED TO CVD INTO
*
IOAREA     DC      0CL17' '
           DC      C' '            PRINT CONTROL
OUT        DC      0CL16' '
SIGN       DC      C' '            PRINTED SIGN
NUM        DC      0CL15' '        PRINTED KEY
           DC      CL14' '
UNPKSIGN   DC      C' '            LAST BYTE OF UNPACKED NUMBER
*
MAXRCDS    DC      F'400'              NO. OF RECORDS TO WRITE
*
WORKAREA   DC      0CL80' '
RECNO      DC      F'0'                CURRENT RECORD NO.
           DC      CL76' '
OUTPUT1    DC      CL8' '              AREA FOR COUNT
           DC      25CL80' '
           EJECT
*          SDMODFOB[6]
           DIMOD TYPEFLE=OUTPUT
           END SDOUTPUT
```

Explanations for Example 1:

1 The BLKSIZE specifies the logical block size of the SAM file. The extra eight bytes specified include the count area required for DTFSD OUTPUT data files.

2 This symbolic unit is ignored. The symbolic unit either comes from the EXTENT statement or is dynamically chosen by VSE/VSAM.

3 The device type specified is ignored. VSE/VSAM determines the device type from the volume serial of the volume that the file resides on. The volume serial is specified either in the EXTENT statement during implicit define, or in the VOLUMES parameter of the DEFINE CLUSTER command, or is chosen by VSE/VSAM from a default model during explicit or implicit define.

4 The RECFORM along with the BLKSIZE and RECSIZE information is used to determine the record format and size characteristics of the file to be written. In addition, if the file is implicitly defined, this information is stored into the VSE/VSAM catalog to be used if the file is accessed through VSE/VSAM (ACB).

5 An output file normally implies reset. That is, the file is set to empty before the records are written into the file. This may be overridden by the DLBL DISP parameter. If DISP=OLD is specified, the file will not be reset and an existing file will have this data added to the end of the file. (If the file does not exist or is empty, DISP=OLD or NEW has no effect.)

6 No SD logic module needs to be assembled or included. (Note that it is a comment.)

Note: If IOAREA2 is specified in the DTFSD (in combination with either IOREG or WORKA) and implicit define occurs, VSE/VSAM will attempt to choose a CI size that will hold at least two SAM logical blocks.

Example 2: Implicit Define of a SAM ESDS File

A job that loads a SAM ESDS file through managed-SAM access (execution). This job implicitly defines a SAM ESDS file.

```
// JOB LOAD A MANAGED SAM FILE (400 RECORDS)
// DLBL SDOUT,'MANAGED.SAM.FILE1',0,VSAM,RECORDS=400,RECSIZE=80[1]
// EXTENT ,VSER01[1]
// EXEC SDOUTPUT,SIZE=AUTO[2]
/&
```

Explanations for Example 2:

1 The information from the DLBL and the EXTENT statement (together with the DTF information) provides the information to do an implicit define.

- The DLBL specifies VSE/VSAM indicating that the SAM file is to be a SAM ESDS file.
- A retention period of 0 indicates that the file can be deleted at any time (assuming it is not in use).
- RECORDS and RECSIZE specify that the primary allocation size should be large enough to hold four hundred records of eighty bytes each. The secondary allocation size is assumed to be twenty percent of the primary allocation size.

— The default disposition for an OUTPUT DTFSD data file is (NEW,KEEP).

— The EXTENT statement specifies that the file is to reside on volume VSER01 and that the logical unit is to be dynamically assigned by VSE/VSAM. (This assumes, of course, that there is VSE/VSAM data space available on volume VSER01.)

2 A SAM ESDS file needs about 52KB of GETVIS space for access (12KB for the file and CI buffers, and a one-time requirement of 40KB for the catalog). SIZE=AUTO will ensure that the maximum GETVIS space is available to VSE/VSAM.

Example 3: Define a Default Model SAM ESDS File

A job stream that loads a SAM ESDS file through SAM access (execution). This job stream defines a default model for a SAM ESDS file and then implicitly defines a SAM ESDS file, using the default model to obtain a volume list (to allow elimination of the EXTENT statement).

```
// JOB DEFINE DEFAULT MODEL FOR SAM ESDS FILE
// EXEC IDCAMS,SIZE=AUTO
                 DEFINE CLUSTER -
                  (NAME(DEFAULT.MODEL.ESDS.SAM) -1
                   VOLUMES(VSER02) -2
                   RECORDS(100 25) -
                   RECORDSIZE(2000 2000) -
                   RECORDFORMAT(UNDEF) -
                   REUSE -
                   NOALLOCATION -3
                   NONINDEXED)
                 LISTCAT -
                   ENTRIES(DEFAULT.MODEL.ESDS.SAM) -
                   ALL
/*
/&
// JOB LOAD A MANAGED SAM FILE (400 RECORDS)
// DLBL SDOUT,'MANAGED.SAM.FILE2',0,VSAM,RECORDS=400,RECSIZE=804
// EXEC SDOUTPUT,SIZE=AUTO
/&
```

Explanations for Example 3:

1 This is the required file-ID for a default model for a SAM ESDS file.

2 This is the volume that will be used for any SAM ESDS file implicitly defined with no EXTENT statement specified (or explicitly defined with no VOLUMES parameter specified).

3 NOALLOCATION is required for default model.

4 The same DLBL information is specified as in Example 2, but the volume that the file is to reside on is retrieved from the default model rather than an EXTENT statement. (The file will reside on VSER02.) Also, the symbolic unit is dynamically chosen and assigned by VSE/VSAM. Note that allocation size and retention period are still obtained from the DLBL statement. The only information retrieved from the default model during implicit define is the volume list.

Example 4: Define a Dynamic SAM ESDS File and Access

A job stream that loads a SAM ESDS file through managed-SAM access (execution). This job stream defines a dynamic SAM ESDS and then accesses the defined file allowing elimination of the EXTENT statement.

```
// JOB ESDS DEFINE FOR SAM ESDS FILE
// EXEC IDCAMS,SIZE=AUTO
                 DEFINE CLUSTER -
                  (NAME(MANAGED.SAM.FILE3) -
                   VOLUMES(VSER03) -
                   RECORDS(16 4) -1
                   RECORDSIZE(2000 2000) -2
                   RECORDFORMAT(FIXBLK(80)) -3
                   REUSE -4
                   NOALLOCATION -
                   NONINDEXED)5
                 LISTCAT -
                   ENTRIES(MANAGED.SAM.FILE3) -
                   ALL
/*
/&
// JOB LOAD A MANAGED SAM FILE (400 RECORDS)
// DLBL SDOUT,'MANAGED.SAM.FILE3',,VSAM,DISP=(NEW,KEEP)6
// EXEC SDOUTPUT,SIZE=AUTO
/&
// JOB EXTEND A MANAGED SAM FILE (ANOTHER 400 RECORDS)
// DLBL SDOUT,'MANAGED.SAM.FILE3',,VSAM,DISP=(OLD,KEEP)7
// EXEC SDOUTPUT,SIZE=AUTO
/&
// JOB ESDS PRINT A MANAGED FILE WITH SAM
// DLBL ESD1,'MANAGED.SAM.FILE3',,VSAM,DISP=(OLD,DELETE)8
// EXEC IDCAMS,SIZE=AUTO
                 PRINT INFILE(ESDS1 -
                   ENVIRONMENT -9
                       (BLOCKSIZE(2000) -
                       RECORDFORMAT(FIXBLK) -
                       RECORDSIZE(80)))
/*
/&
```

Explanations for Example 4:

1 This specifies the number of VSE/VSAM logical records (SAM logical blocks) for primary and secondary allocation. Sixteen is specified for the primary allocation since sixteen 2000-byte logical blocks will be written to hold four hundred 80-byte SAM logical records.

2 This specifies the average and maximum VSE/VSAM logical record size (SAM logical block size) for the file.

3 This specifies the SAM logical record size for the file.

4 REUSE in connection with NOALLOCATION makes the file a dynamic file.

5 NONINDEXED is required for a SAM ESDS file.

6 The DISP parameter specifies that the file is to be reset at OPEN, and kept at CLOSE. The file resides on volume VSER03 as specified in the define cluster. A symbolic unit will be dynamically assigned.

7 The DISP parameter specifies that the file is *not* to be reset at OPEN, that is, the file will be extended with the records written by JOB EXTEND. The file is to be kept at CLOSE.

8 The DISP parameter specifies that the file is *not* to be reset at OPEN. (A specification of DISP=NEW would be an error in this case since the file will be opened for input.) When the file is closed, it will be deleted (that is, deallocated since this file was defined as a dynamic file).

9 The file is read by way of managed-SAM access by way of the

ENVIRONMENT parameter. The ENVIRONMENT subparameters specify the information required to generate a DTF. The file may be accessed through VSE/VSAM by omitting the ENVIRONMENT parameter. In that case, VSE/VSAM gets the SAM file characteristics from the VSE/VSAM catalog entry for the file.

Differences Between VSE/VSAM ESDS and SAM ESDS File Format

How CIs are Formatted into CAs

Figure 30 on page 180 and **Figure 31 on page 181** illustrate the way in which CIs are physically formatted into CAs for VSE/VSAM ESDS files as compared to SAM ESDS files.

For values for CI size and tracks, refer to Figure 27 on page 91.

VSE/VSAM ESDS Files

A VSE/VSAM ESDS file formats CIs into CAs in CA format. This means that CIs cannot be written across CA boundaries. If there is not sufficient space at the end of a CA to write a complete CI, an area of unusable space is left and the CI to be formatted is written at the beginning of the next CA. This is illustrated in Figure 30 on page 180.

Assumptions::

```
Device type=3390
Allocation specified=TRK(3 1)
CI size=14KB
Physical block size=7KB
1 track=7 blocks (PR)
11 CIs of data are written
CA=Min (primary (3 TRKs), secondary (1 TRK), Max-CA(1 CYL))
```

Therefore: CA=1 track

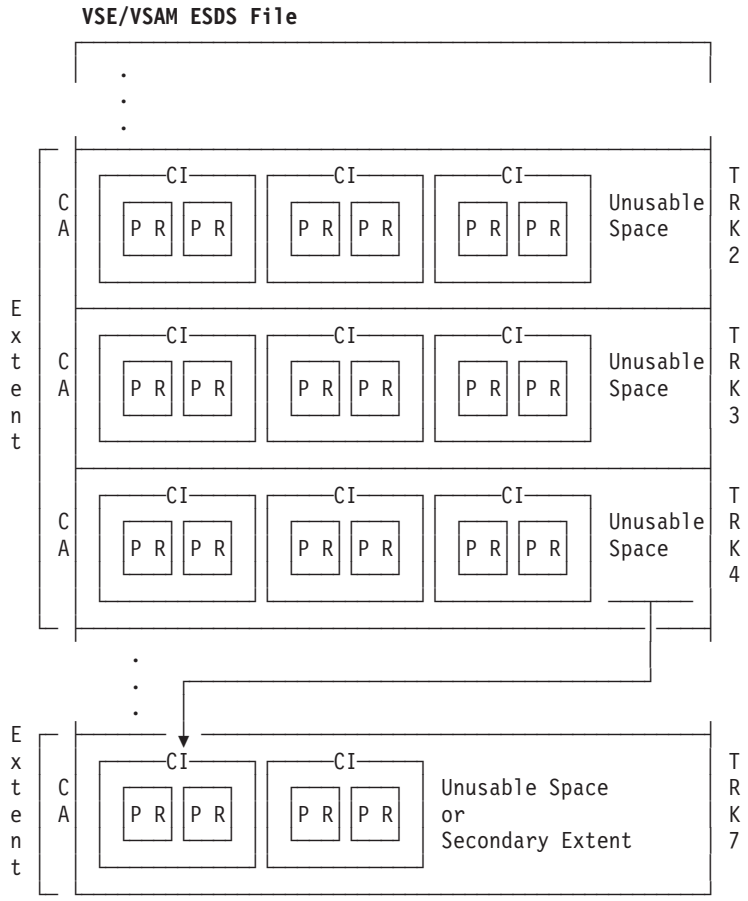


Figure 30. Example of CA Format Using a VSE/VSAM Entry-Sequenced File

SAM ESDS Files

A SAM ESDS file formats CIs into CAs in non-CA format. This means that a CI can be written across a CA boundary (tracks 2 and 3) but not across an extent boundary. If there is not sufficient space at the end of the CA to write a complete CI, the CI will be written across a CA boundary causing the CI to have part of its contents in one CA and the rest of its contents in another. This is illustrated in Figure 31 on page 181.

Assumptions:

```
Device type=3390
Allocation specified=TRK(3 1)
CI size=14KB
Physical block size=7KB
1 track=7 blocks (PR)
11 CIs of data are written
CA=Min (primary (3 TRKs), secondary (1 TRK), Max-CA(1 CYL))
```

Therefore: CA=1 track

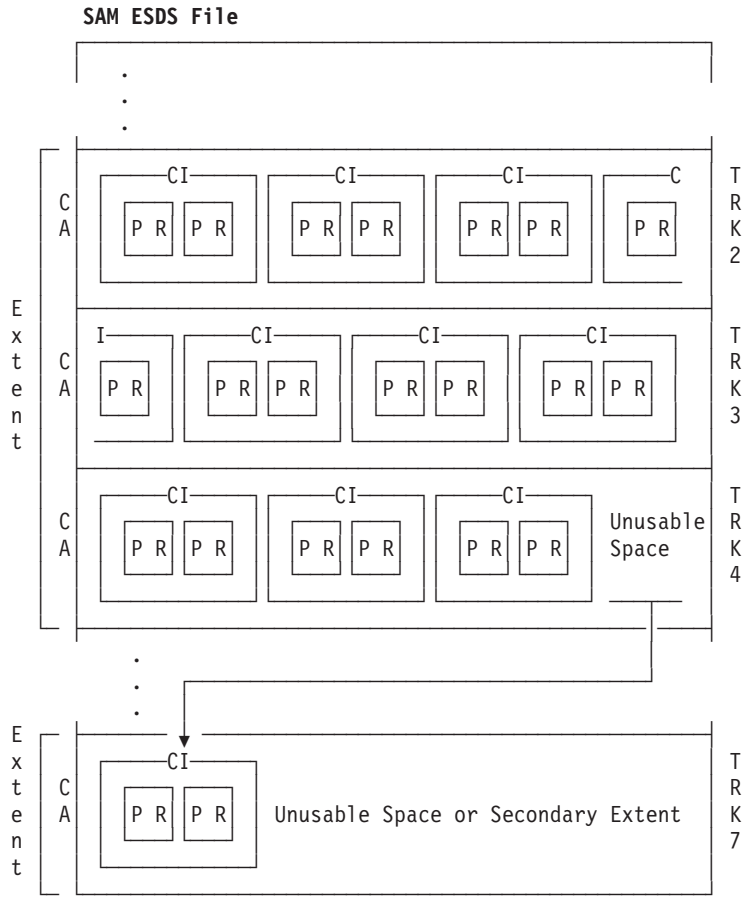


Figure 31. Example of Non-CA Format Using a SAM ESDS File

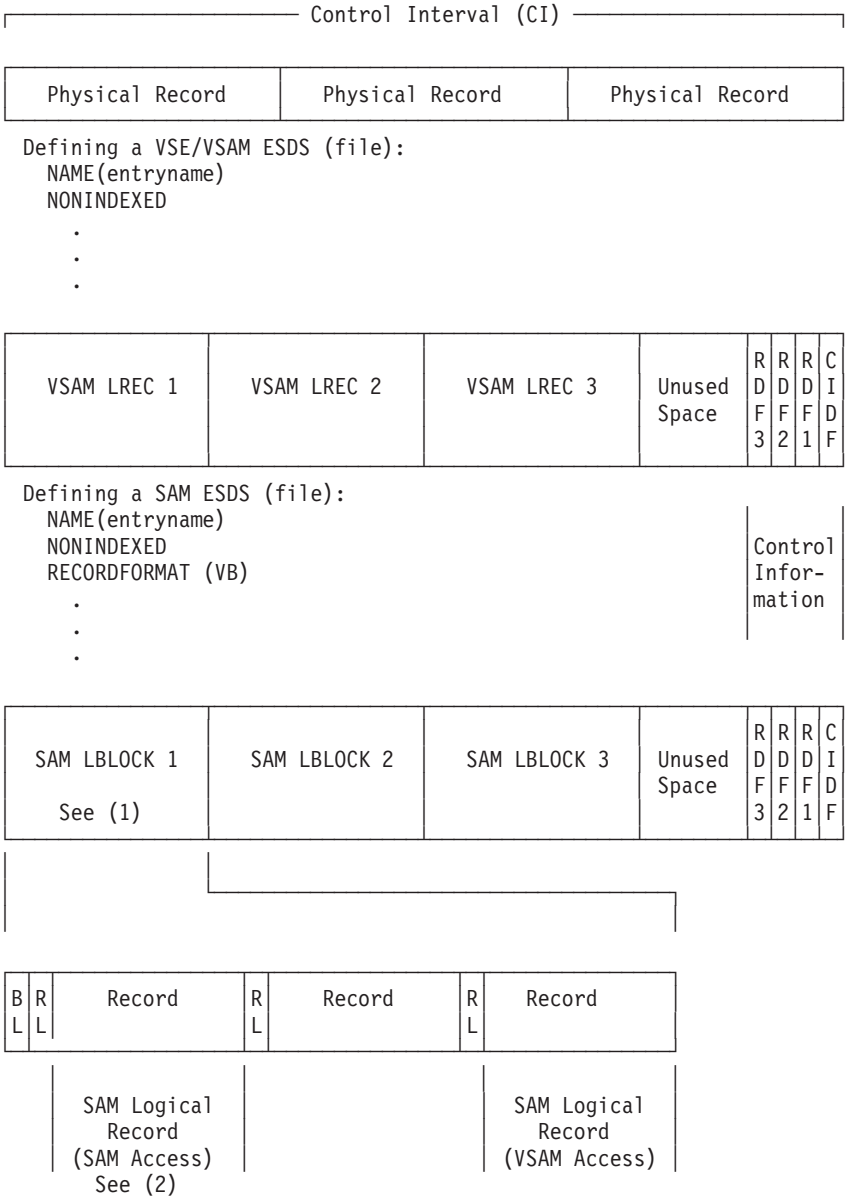
Relationship of Physical and Logical Layout

Figure 32 on page 182 shows the construction of a CI and how records are physically and logically laid out for a VSE/VSAM ESDS file and a SAM ESDS file. It explains the relationship between *VSE/VSAM logical records* and *SAM logical blocks*.

When you have defined a VSE/VSAM ESDS file, the CI is made up of VSE/VSAM logical records and their related control information. When you define a SAM ESDS file, the VSE/VSAM logical records become SAM logical blocks. The CI size is a multiple of the VSE/VSAM block size and normally determined by VSE/VSAM, not by you, at DEFINE time. Control information in a CI consists of a CIDF and RDFs. There is an RDF for every SAM logical block (VSE/VSAM logical record) indicating its length, except in the case of consecutive logical blocks of equal length, in which case the first RDF (right-most of the pair) describes the length of the logical blocks and the second RDF (left-most of the pair) tells how many logical blocks the first RDF describes.

The SAM logical block consists of SAM logical records. In the case of VB format, every logical record is prefixed with an RL (record length) field which indicates the length of the record. The SAM logical block begins with a BL (block length) field which indicates the length of the block. During managed-SAM (DTF) access of V or VB records, the RL is returned at the beginning of the record. For VSE/VSAM

(ACB) access, it is not. A program using VSE/VSAM (ACB) access for sequential processing can process a SAM ESDS file or a VSE/VSAM ESDS file.



(1) The SAM LOGICAL BLOCK size is what you specify in the RECORDSIZE parameter when you DEFINE a SAM ESDS file, and define in the BLKSIZE parameter of the DTF.

(2) The SAM LOGICAL RECORD size is what you specify in the SIZE parameter of the DTF.

Figure 32. Comparison of a VSE/VSAM Block to a SAM Logical Block

Chapter 10. Using VSE/VSAM Macros

This Chapter ...

- Documents Programming Interface information. See "Notices" on page xiii.
- Explains the use of the VSE/VSAM macros. VSE/VSAM macro instructions are coded in an assembler program to access the data.
- Assigns the macros to various tasks, there are macros for:
  - Relating a program to the data. They identify the file and describe the kind of processing to be done. They are ACB, EXLST, RPL, and GENCB.
  - Connecting and disconnecting the program to the file. They are OPEN, CLOSE, and TCLOSE.
  - Displaying and changing the information relating a program to the data and thus changing the type of processing. They are SHOWCB, TESTCB, and MODCB.
  - Initiating transfer of data between disk storage and processor storage, positioning within the file, or deletion of records. They are GET, PUT, POINT, ERASE, WRTBFR, and ENDREQ.
  - Sharing I/O Buffers and control blocks among files, and to write out buffers. They are BLDVRP, DLVRP, and again WRTBFR. Also ACB, RPL, and the macros of the second group have been extended for sharing resources and managing I/O buffers.
  - Displaying catalog information: SHOWCAT.

Groups of Macros

The VSE/VSAM macros can be grouped according to main tasks and the relationship between the various macros. The following shows the groups and outlines the purpose of the individual macro.

Declarative VSE/VSAM macros:

- ACB specifies the file to be processed and the access type.
- EXLST specifies a list of user-supplied exit routines.
- RPL specifies information for a request to access a particular record.

Macros to Share Resources Between Several Files

- BLDVRP builds a VSE/VSAM pool of buffers, control blocks, and channel programs.
- DLVRP deletes such a resource pool.
- WRTBFR writes waiting buffer contents to satisfy a GET request.

Request Macros

- GET retrieves a record from a file for processing.
- PUT inserts a record in a file.
- POINT positions control on a specific address in the file.
- ERASE deletes a record in a file.
- ENDREQ ends processing of a GET or POINT request.

© Copyright IBM Corp. 1979, 2005

Control Block Manipulation Macros

- GENCB specifies declarative parameters, but during execution of a program.
- MODCB changes declarative parameters.
- SHOWCB displays declarative parameters in effect.
- TESTCB checks declarative parameters (or their error codes) and branches accordingly.
- SHOWCAT displays data from the catalog in a buffer you have supplied.

OPEN/CLOSE Macros

- OPEN connects a program to a file.
- CLOSE prepares the separation and disconnects a program from a file.
- TCLOSE prepares the separation but leaves program and file connected.

Relating a Program and the Data

The ACB macro specifies the file to be processed and the types of access you want to use.

The EXLST macro specifies a list of user-supplied exit routines.

The RPL macro specifies information for a request to access a particular record in the file.

These declarative macros are used while assembling or compiling an assembler program.

The GENCB macro can be used in place of the ACB, EXLST, or RPL macros to generate processing specifications while the processing program is running.

ACB: Specifying the Access Method Control Block

Every VSE/VSAM file has an ACB (Access Control Block) that contains information about it. The file name of the DLBL job control statement that describes the file is included, so that the Open routine can connect a program to the data.

The other information that you specify enables OPEN to prepare the kind of processing to be done by your program.

Exit Routines

The address of a list of exit-routine names that you supply (EXLST parameter). You use the EXLST macro, described next, to construct the list.

I/O Buffers

The amount of space for I/O buffers (BUFSP parameter) and the number of I/O buffers (BUFND and BUFNI parameters) that VSE/VSAM will use to process data and index records. The minimum number of buffers allowed depends on how much buffer space is allocated, the number of concurrent requests to be allowed, and whether processing will be direct or sequential.

Password

The password, if required, indicates the level of authorization to access the file: read, read and update, and so on (PASSWD parameter).

Processing Options

The processing options to be used (MACRF parameter): keyed or addressed access, or both; sequential, direct, or skip sequential processing, or a combination; retrieval, storage, or update (including deletion), or a combination; whether to use the shared resource pool and to defer the writing of updated records.

Concurrent Requests

For processing concurrent requests (STRNO parameter), the number of requests that are defined for processing the file (see the discussion of the RPL macro following EXLST).

Error Messages

Address and length of an area for error messages from OPEN, CLOSE, or TCLOSE (MAREA and MLEN parameters).

EXLST: Specifying the Exit List

The EXLST macro specifies the addresses of optional exit routines that you can supply for analyzing physical errors and logic errors, for end-of-file processing, for overlapping I/O operations, and for writing a journal. Any number of ACBs in a program can indicate the same exit list, and an exit routine can be used for several files.

Analyzing Physical Errors (SYNAD)

When VSE/VSAM encounters an error in an I/O operation that the z/VSE error recovery routines cannot correct, it exits to the physical-error analysis (SYNAD) routine. VSE/VSAM sets a code in the RPL to indicate whether the I/O error occurred during reading or writing the data or the index.

Analyzing Logic Errors (LERAD)

Errors not directly associated with an I/O operation, such as an invalid request, cause VSE/VSAM to exit to the logic error analysis (LERAD) routine. VSE/VSAM sets a code in the RPL that indicates the type of logic error.

End-of-File Processing (EODAD)

When your program requests a record beyond the last record in the file during sequential access, your end-of-file (EODAD) routine is given control. The last record is the highest-addressed record for addressed or control-interval access or the highest-keyed record for keyed access. If an EODAD exit routine is not available, control is given to the LERAD exit routine.

Overlapping I/O Operations (EXCPAD)

When VSE/VSAM starts an I/O operation caused by a request macro, the execute-channel-program (EXCPAD) exit routine is given control. The EXCPAD routine must return control to VSE/VSAM, which continues your mainline routine at the instruction following the request macro. The EXCPAD exit is intended for use by programmers of utilities and systems.

Writing a Journal (JRNAD)

You can use the JRNAD routine to journalize the transactions made against your file and to keep track of RBA changes.

For recording transactions, VSE/VSAM exits to the JRNAD routine every time your processing program issues a GET, PUT, or ERASE. For keeping track of RBA changes, VSE/VSAM takes the JRNAD exit every time data is shifted within a CI or moved to another CI. To process a key-sequenced file with addressed access, you need to know whether any RBAs have changed during keyed processing.

VSE/VSAM takes the JRNAD exit before transmitting to direct-access storage the contents of a CI in which there was an RBA change.

RPL: Specifying the Request Parameter List

The RPL macro produces a Request Parameter List (RPL) which contains all the information needed by a request macro to access a record in the file. The request macros are GET, PUT, POINT, ERASE, and WRTBFR. The RPL identifies the file to which the request is directed by naming the ACB of the file.

You can use a single RPL to define parameters that apply to several requests. With the MODCB macro (described below) you can modify some of the parameters to change the type of processing, such as from direct to sequential or from update to non-update.

For concurrent requests, which require VSE/VSAM to keep track of more than one position in a file, any number of RPL macros may be used asynchronously by a processing program or its subtasks to process a file. The requests can be sequential or direct or both, and they can be for records in the same part or different parts of the file. You need specify only the RPL parameters appropriate to a given request, as follows:

Processing Options for a Request (OPTCD)

A request is for keyed, addressed, or control-interval access. The processing can be sequential, skip sequential (keyed access only), or direct. For keyed and addressed access and for sequential or direct processing, records may be retrieved in backward direction. A request may be for updating or not updating a record. A non-update direct request to retrieve a record can optionally cause positioning at the following record for subsequent sequential access.

For a keyed request, you specify either a generic key or a full key to which the key field of the record is to be matched. A generic key can match several records while a full key matches only one record. You can also specify that, if the key does not match the key of any record in the file, the record with the next greater key will be processed.

For retrieval, a request is either for a data record to be placed in a work area in the processing program (move mode) or for the address of the record within VSE/VSAM's I/O buffer to be passed to the processing program (locate mode).

Address of the Work Area for, or Pointer to, a Data Record (AREA)

For retrieval, update, insertion, or addition of a record, you must provide a work area in which the record is to be processed (move mode). For retrieval, you can have VSE/VSAM give you the address of the record within VSE/VSAM's I/O buffer (locate mode) in this field.

Size of the Work Area for a Data Record (AREALEN)

This parameter specifies either the length of the work area in which a record is placed (for move mode) or the four-byte address of the record in VSE/VSAM's I/O buffer (for locate mode). Having a work area that is too small is considered a logic error.

Length of the Data Record Being Processed (RECLEN)

For storage, your processing program indicates the length to VSE/VSAM; for retrieval, VSE/VSAM indicates it to your program.

Length of the Key (KEYLEN)

This parameter is required only for processing by generic key. For ordinary keyed access, the full key length is available from the catalog.

Address of the Area Containing the Search Argument (ARG)

The search argument is either a key (including a relative-record number) or an RBA. If the OPTCD parameter indicates a generic key, the KEYLEN parameter tells how many high-order (leftmost) bytes of the search argument will be used.

Address of the Next RPL in a Chain (NXTRPL)

You can process several records with a single GET, PUT, or ERASE by chaining RPLs together. For example, every RPL in a chain could contain a unique search argument and point to a unique work area. A single GET macro would retrieve a record for every RPL in the chain. A chain of RPLs is processed as a single request. (Chaining RPLs is not the same as issuing concurrent requests that require VSE/VSAM to keep track of multiple positions in a file.)

Transaction-ID (TRANSID)

With this parameter you can create a logical relationship between I/O requests issued for different VSE/VSAM files.

GENCB: Generating Control Blocks and Lists

You can use the GENCB macro to generate an ACB, EXLST, or RPL during the execution of your processing program, rather than to assemble it with the corresponding macro. GENCB is coded in the same way as the other macros, but it generates one or more copies of a control block or list and allows you to code parameter values in more ways.

Connecting and Disconnecting a Processing Program and a File

OPEN connects a processing program to a file, so that VSE/VSAM can satisfy the program's request for data. CLOSE completes processing and frees resources that were obtained by the Open routine. TCLOSE causes buffers to be written out and the catalog to be updated.

OPEN: Connecting a Processing Program to a File

The OPEN macro calls the Open routine, which verifies that the processing program has authority to process the file, constructs VSE/VSAM control blocks and establishes linkages to VSE/VSAM routines. By examining the DLBL statement indicated by the DDNAME operand in the ACB macro and the volume information in the catalog, Open verifies that the necessary volumes have been mounted. When you are opening a key-sequenced file or an alternate index, VSE/VSAM issues an error code to warn you if the data has been updated separately from its index.

CLOSE: Disconnecting a Processing Program from a File

The Close routine completes any I/O operations that are outstanding when a processing program issues a CLOSE macro for a file. It writes any output buffers that have not been stored.

Close updates the catalog entries for any changes in the attributes of a file; it also updates the statistics on file processing (such as number of records inserted). The addition of records to a file may cause its end-of-file indicator to change, in which case Close updates the end-of-file indicator in the catalog. These end-of-file

indicators help ensure that the entire file is accessible. If an error prevents VSE/VSAM from updating the indicators, the file is flagged as not properly closed. When a processing program subsequently issues an OPEN macro, it is given an error code indicating the failure.

Because it is essential for the integrity of a file that it is closed properly, z/VSE automatically attempts to close all open VSE/VSAM files within the partition at both normal and abnormal termination of a job step. If any control blocks for a file have been destroyed through an error in your program, this file cannot be closed and a message is issued to the operator. EXLST routines are not entered during automatic CLOSE.

Close restores control blocks to the status that they had before the file was opened, and frees the virtual storage space that Open used to construct VSE/VSAM control blocks.

TCLOSE: Securing Records Added to a File

The TCLOSE macro performs the functions of CLOSE, except that it leaves the program and the file connected so that you can continue processing without reopening the file. You can use the TCLOSE macro to protect data while the file is loaded or extended. Positioning is lost when a TCLOSE is issued.

Manipulating and Displaying the Information Relating Program and Data

The MODCB, SHOWCB, and TESTCB macros are used for modifying, displaying, and testing the contents of an ACB, EXLST, or RPL.

MODCB: Modifying the Contents of Control Blocks and Lists

The MODCB macro is used to specify new values for fields in an ACB, EXLST, or RPL. For example, to use a single RPL to retrieve directly the first record having a certain generic key and then to retrieve sequentially the rest of the records having that generic key, you would use MODCB to alter the RPL to change from direct to sequential access.

SHOWCB: Displaying Fields of Control Blocks and Lists

SHOWCB allows you to examine the contents of fields in an ACB, EXLST, or RPL. VSE/VSAM displays the requested fields in an area you provide. You can also display fields in addition to those defined in the macros. For example, when a file is open, you can display various counts, such as number of CI splits, number of deleted records, and number of index levels. The RBA of the last record accessed and the error codes set in the ACB or RPL after macro execution can also be displayed.

TESTCB: Testing the Contents of Control Blocks and Lists

The TESTCB macro enables you to test the contents of a field or combination of fields in an ACB, EXLST, or RPL for a particular value and to alter the sequence of your processing steps as a result of the test. Thus, TESTCB is similar to a branch instruction. You can test the error codes set in the ACB or the RPL, for instance, or the attributes of a file, such as record length.

Requesting Data Transfer, Positioning, and Deletion of Records

All of the preceding macros prepare to process a file. The request macros (GET, PUT, POINT, ERASE, and WRTBFR) initiate an access to data. Another request macro, ENDREQ, is provided to (1) terminate processing of a request when completion is not required, or (2) free VSE/VSAM from having to keep track of a position in the file. Each of these macros is associated with an RPL (or chain of RPLs) that fully defines the request. The only parameter that is needed with a request macro is the address of the RPL that defines the request.

Displaying Catalog Information

SHOWCAT: Retrieving Information from a Catalog

With the SHOWCAT macro, you can retrieve information from a catalog about any non-open file defined in the catalog.

The entries in a catalog are related. Several entries are required to describe an object and its associated objects (for example, a cluster and its data and index components); one entry points to one or more other entries, which point to yet others. Figure 33 on page 190 shows the relationship of entries that describe the following *types of objects*:

- Alternate index (G)
- Cluster (C)
- Data component (D)
- Index component (I)
- Path (R)
- Upgrade set (Y)

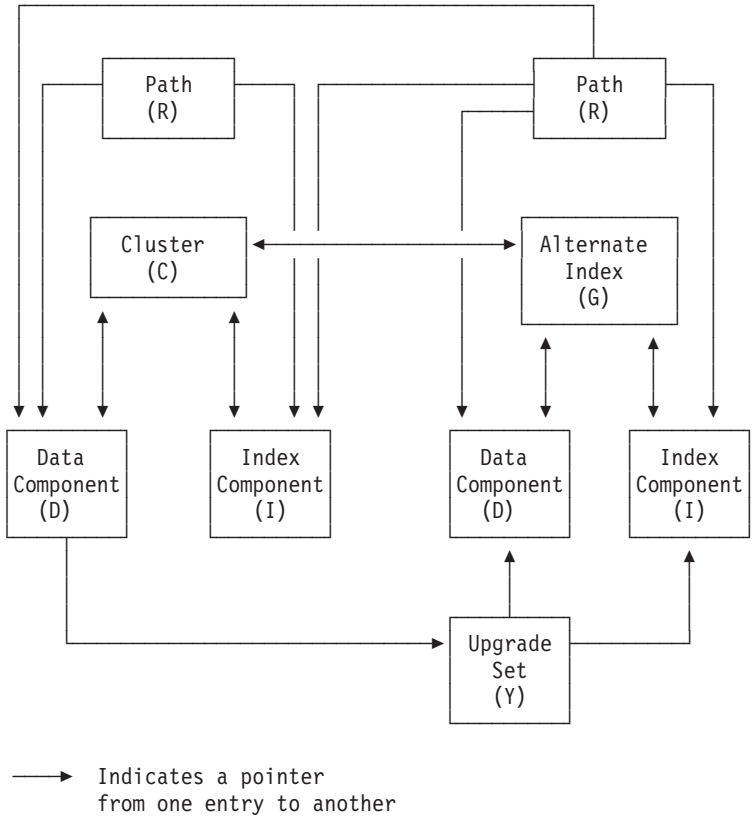


Figure 33. Relationship of Catalog Entries

For example, an *alternate index entry* points to the entries of its data and index components, its base cluster, and its path. SHOWCAT enables you to follow the arrows pictured in Figure 33. You first issue SHOWCAT by specifying the name of the object you want to display. The information VSE/VSAM returns to you (only if EXTOPT is not specified) includes the CI numbers of the catalog entries that describe any associated objects. You then issue subsequent SHOWCATs to retrieve information from these associated entries by specifying the CI numbers that VSE/VSAM has returned. The first time you issue SHOWCAT, VSE/VSAM searches catalogs (in the following order) to locate the entry that describes the object to be displayed:

1. The catalog identified by the SHOWCAT CATDSN parameter, if specified.
2. The catalog identified by the DLBL CAT parameter for the VSE/VSAM file.
3. The job catalog identified by the IJSYSUC DLBL statement, if supplied.
4. The master catalog (IJSYSCT).

You must provide DLBL cards for:

- The master catalog – if the entry is in the master catalog.
- The master and the job catalogs – if the entry is in the job catalog.
- The master catalog – if the entry is in a user catalog specified by either the SHOWCAT CATDSN parameter or the DLBL CAT parameter for the VSE/VSAM file.

VSE/VSAM returns to you the address of the ACB that defines the catalog containing the entry to be displayed. The subsequent times you issue SHOWCAT, you can specify that address, which causes VSE/VSAM to search only the corresponding catalog.

Sharing Resources Among Files and Displaying Catalog Information

Normally, buffers and control blocks are allocated statically to a file at the time the file is opened; they are freed when the file is closed. As long as the file is open, these buffers and control blocks cannot be used by any other file.

The Shared Resources facility, however, allows you to share buffers, I/O control blocks, and channel programs among several VSE/VSAM files within a partition, and to manage I/O buffers. These buffers and control blocks are allocated out of a common resource pool at the time you issue an I/O request for a file. When the request is satisfied, the same buffers and control blocks can be assigned to another file (for direct requests).

Sharing these resources optimizes their use and also reduces the amount of virtual storage required (the working set) per partition. The facility is especially useful in an environment in which (a) many VSE/VSAM files are open and it is therefore difficult to predict the amount of activity that will occur at a given time, or (b) every transaction may refer to several files.

Managing I/O buffers includes:

- Deferring write operations for direct PUT requests, thus reducing the number of I/O operations.
- Correlating deferred requests by a transaction ID.
- Writing out buffers whose writing has been deferred.

Managing I/O buffers should enable you to speed up direct processing of VSE/VSAM files whose activity is unpredictable.

When you share resources for sequential access, you have to establish positioning before you can issue your initial retrieval request, because with shared resources VSE/VSAM does not automatically position itself at the beginning of a file opened for sequential access. Also note that you may not use shared resources to load records into an empty file.

The macros you use to share resources and write I/O buffers are:

- BLDVRP (build VSE/VSAM resource pool)
- DLVRP (delete VSE/VSAM resource pool)
- WRTBFR (write buffer)

In addition, the SHOWCAT macro is provided to display, for non-open files, the catalog information needed for the proper specification of some of the BLDVRP operands.

The ACB, RPL, SHOWCB, MODCB, and TESTCB macros have been extended to provide for sharing resources and managing I/O buffers.

Data Set Name Sharing

Normally, VSE/VSAM handles data sets that are opened through different access method control blocks (ACBs) always as different data sets; this applies even if such ACBs point to the same data set. Thus, when a file is opened through different ACBs, the *read integrity* may be impaired. Also, for non-shared resources (NSR), individual buffers with different copies of the same data and index records are in virtual storage, but cannot be shared. This results in unnecessary input/output operations, and negatively affects read/write integrity.

VSE/VSAM provides the processing option *data set name sharing*. Using this option:

- Improves data integrity when opening a file through different ACBs.
- Does not violate data integrity when writing to base clusters directly, or when writing through paths or alternate indexes simultaneously.
- Allows local shared resources (LSR) or non-shared resources (NSR) to share I/O buffers and control blocks of a file that has been opened through different ACBs.

  ACBs that are created by VSE/VSAM internally can also access shared buffers; this, however, does not apply to catalogs.

To use *Data Set Name Sharing*, you essentially have to make entries in the ACB macro; in the:

- MACRF operand -- you have to specify DSN and DDN.
- BSTRNO operand -- you have to consider additional requirements for handling the base cluster of an alternate index (AIX).

The VSE/VSAM control block (CB) manipulation macros GENCB, MODCB, SHOWCB, and TESTCB are available to manipulate the ACB.

The following ACB specifications are MVS compatible:

```
MACRF=(DDN|DSN) and BSTRNO=number
```

Considerations

If you use *Data Set Name Sharing*, note that:

- The first opening ACB has to define the *total number of strings* for the first and all following ACBs. (This is similar to processing LSR resource pools.)
- All buffers have to be defined with the first ACB.
- All ACBs that are to be opened for a specific file must use the same resource pool. That is, you have to specify the *same* SHRPOOL number in each ACB.
- VSE/VSAM ignores the definition of STRNO, BSTRNO, BUFSP, BUFNI and BUFND for the second and further data set name shared ACBs.
- VSE/VSAM rejects an open to a reusable data set if ACB MACRF=(....DSN,RST...) is specified.
- VSE/VSAM rejects an open if ACB MACRF=(....DSN,UBF...) is specified.
- Before issuing TCLOSE, issue ENDREQ to the ACB-related RPLs. This avoids unpredictable results that could be caused by outstanding input/output processing.
- For DSN shared ACBs, VSE/VSAM ignores the share option specified in the IDCAMS commands ALTER and DEFINE. That is, if you specify *data set name sharing*, and whenever VSE/VSAM has opened a file, all data integrity within the DSN structure is handled internally by VSE/VSAM without the z/VSE LOCK facility. Additional OPENs to this file without *data set name sharing* are handled by VSE/VSAM depending on the specifications in the share option.

- It is *not* possible to share a path through an alternate index and a *single alternate index* (either opened as a key sequenced file, or opened through a path specified wit ACB MACRF=(AIX)). The reason for this is: there is a possibility that buffers containing base cluster records and alternate index records are mixed.

Processing

The sharing of buffers and control blocks of a data set is initiated at OPEN time through the operands MACRF=(DSN) and BSTRNO=number.

If a mix-up of input/output ACBs occurs, VSE/VSAM issues a warning message; nevertheless, opening of the file will be successful. VSE/VSAM handles a mix-up as follows:

- The first opening ACB designates whether the whole structure is for input or output. After the first open, you cannot change the structure anymore.
- If the first ACB is opened with MACRF=(...OUT,DSN...), and if one of the *following ACBs* is opened with MACRF=(...IN,DSN...), each insert/update request through such *following ACB* is rejected.
- If the first ACB is opened with MACRF=(...IN,DSN...), each insert/update request through this or each following ACB is rejected, even if the following ACB has been opened with MACRF=(...OUT,DSN).

Specifying Manipulation Macros

The VSE/VSAM control-block manipulation macros GENCB, MODCB, SHOWCB, and TESTCB are available to manipulate the ACB.

The following outlines the use of MACRF operand and the BSTRNO value:

GENCB ACB

```
,MACRF=(..,DSN),BSTRNO=number ...
```

This generates an ACB with MACRF=DSN, and sets the ACB field BSTRNO to the additional base cluster string number.

MODCB ACB

```
,MACRF=(..,DDN|DSN),BSTRNO=number ...
```

This modifies the ACB to MACRF=DSN or DDN, and sets the ACB field BSTRNO to the additional base cluster string number.

SHOWCB ACB

```
,FIELDS=(BSTRNO) ...
```

This shows the value of the ACB field BSTRNO.

TESTCB ACB

```
,MACRF=(..,DDN|DSN) ...
,BSTRNO=number ...
```

This tests the ACB for MACRF=DDN or DSN, and tests for the value of the ACB field BSTRNO.

Buffers and LSR Pools above 16MB Line of Storage

VSE/VSAM allows to allocate virtual storage for I/O buffers and for multiple local shared resources (LSR) pools *above* or *below* the 16MB line of address space.

The option can be specified through the parameter RMODE31 that is available in the macros ACB and BLDVRP. Refer to "The ACB Macro" on page 196 and "The BLDVRP Macro" on page 207. For information on how VSE/VSAM allocates buffers, refer to "Buffer Allocation above the 16MB Line of Storage" on page 16.

Note that a *program check* may occur if:

- A program uses a 24-bit address and if you attempt to reference control blocks, I/O data buffers, or LSR pools that are located above the 16MB line of storage.
- You attempt to use LOCATE mode:
  While in 24-bit mode, and
  RMODE31=ALL was specified.

When you use 31-bit addresses in your programs, note the following:

- All VSE/VSAM control blocks that have fields defined as 31-bit addresses *must* contain 31-bit addresses.

  Do not use the high-order byte of a 31-bit address field as a user-defined flag field. This applies to 24-bit and 31-bit addressing.
- You may obtain I/O data buffers from above or below the 16MB line as follows:
  - Below the 16MB line by taking the default (=NONE) in the ACB or BLDVRP macro.
  - Above the 16MB line by specifying RMODE31=ALL in the ACB or BLDVRP macro.
- The parameter list that is passed to your exit routine resides below the 16MB line.
- You must recompile the portion of your program that contains the ACB, BLDVRP, and DLVRP macro specifications, including control block manipulation requests.

Chapter 11. Descriptions of VSE/VSAM Macros

This Chapter ...

- Documents Programming Interface information. See "Notices" on page xiii.
- Describes the macros in *alphabetical* order. For each macro, you find an explanation of the format and operands, and other related details. When details apply to *macro groups*, the information is organized as follows:
  - Declarative Macros (ACB, EXLST, RPL):
    - "Examples: ACB, EXLST, and RPL Macros" on page 265.
  - Request Macros (GET, PUT, and so on):
    - "Examples of Request Macros" on page 267.
    - "Return Codes of Request Macros" on page 289.
  - Control Block Manipulation Macros (GENCB, MODCB, and so on):
    - "List, Execute, and Generate Forms of the Control Block Manipulation Macros" on page 291.
    - "Return Codes from the Control Block Manipulation Macros" on page 291.
  - OPEN/CLOSE Macros:
    - "OPEN/CLOSE/TCLOSE Message Area" on page 204.
- For information on the various *macro groups*, refer to Chapter 10, "Using VSE/VSAM Macros," on page 183.

Syntax of VSE/VSAM Macros

For the general command description conventions, refer to the manual *VSE/VSAM Commands* under "Understanding Syntax Diagrams."

In the VSE/VSAM macros, you can code **address** as a symbolic name. Except for the ACB, EXLST, and RPL macros, you can also code an address as a register, using either ordinary register notation (with registers 2 through 12) or, if shown in the format description as a decimal number in parentheses, special register notation. For example:

```
RPL=address|(1)
```

means that you can specify either a symbolic address, any of the registers 2 to 12, or Register 1.

The use of Registers 0, 1, 13, 14, and 15 is the same as for z/VSE macros. VSE/VSAM does not save the contents of registers 0, 1, 14, 15 before using them. The highest order part of register 13 can be changed, depending on the caller's AMODE. If you use these registers, you must either save their contents yourself (and reload them later) or finish with them before VSE/VSAM uses them. For additional information about the use of registers, see the *z/VSE System Macros Reference*.

You can code a **value** (number) as any absolute expression, except for a self-defining character term. You can code a **name** according to the rules of the

© Copyright IBM Corp. 1979, 2005

assembler. The control block manipulation macros (GENCB, SHOWCB, MODCB, and TESTCB) can be coded in even more ways as shown in "Operand Notation for VSE/VSAM Macros" on page 295.

Some operands of the VSE/VSAM macros can have more than one parameter. These operands are shown with parentheses around the parameters (for example, the MACRF operand of the ACB macro). This means that you can code the operand, if it has only one parameter, with or without parentheses around the parameter:

```
MACRF=option
MACRF=(option)
```

However, if the operand is coded with two or more parameters, enclosing parentheses are required:

```
MACRF=(option,option)
```

VSAM Executable Macros and Their Mode Dependencies

Table 10. VSAM Macros and Their Mode Dependencies

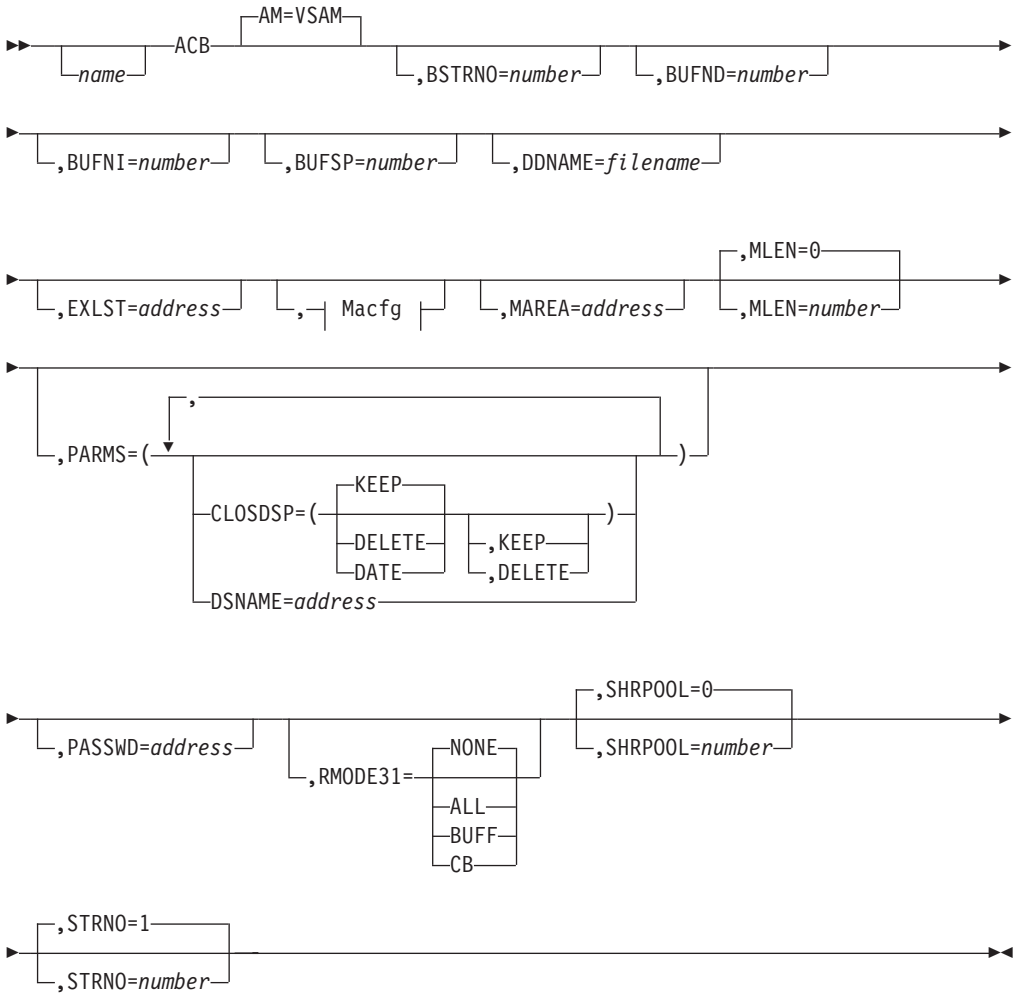
Macro	AMODE	RMODE	Comment
CLOSE	ANY	24	ACB has to be allocated below 16MB
OPEN	ANY	24	ACB has to be allocated below 16MB
TCLOSE	ANY	24	ACB has to be allocated below 16MB
GET (RPL)	ANY	ANY	RPL has to be allocated below 16MB
PUT(RPL)	ANY	ANY	RPL has to be allocated below 16MB
ENDREQ	ANY	ANY	RPL has to be allocated below 16MB
POINT	ANY	ANY	RPL has to be allocated below 16MB
GENCB	ANY	ANY	ACB,RPL,EXLST,and WAREA must be allocated below 16MB
SHOWCB	ANY	ANY	ACB,RPL,EXLST must be allocated below 16MB
MODCB	ANY	ANY	ACB,RPL,EXLST must be allocated below 16MB
TESTCB	ANY	ANY	ACB,RPL,EXLST must be allocated below 16MB
SHOWCAT	ANY	24	all parameters must be allocated below 16MB
BLDVRP	ANY	ANY	
DLVRP	ANY	ANY	
WRTBFR	ANY	ANY	

The ACB Macro

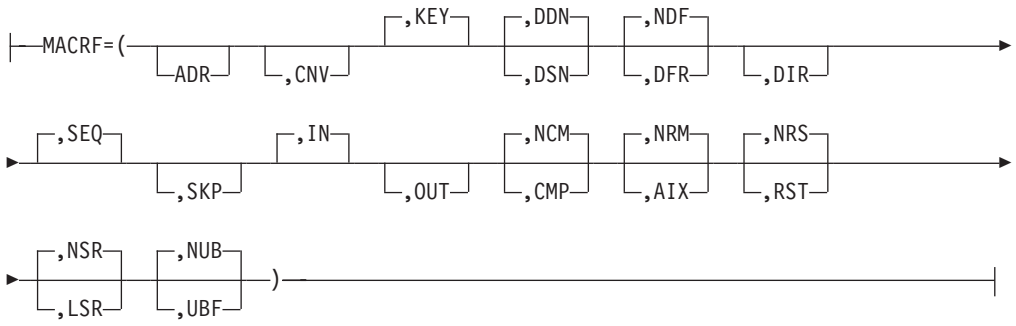
You specify most information (such as key length or record format) about the file in the DEFINE command of IDCAMS. That information then resides in the VSE/VSAM catalog and is brought into virtual storage when the ACB is opened.

You code the values for the ACB macro operands as absolute numeric expressions, character strings, codes, and expressions that generate valid relocatable A-type address constants. Ordinary register notation cannot be used for address.

Format of the ACB Macro



Macfg:



name
one through eight characters that provide a symbolic address for the ACB that is assembled. If you omit the DDNAME parameter, the specified name serves as the file name that you must specify in the DLBL JCL statement. In that case, the name you use must not exceed seven characters, and its first character must be a letter (A - Z).

AM=VSAM
specifies that this is a VSE/VSAM control block. You may want to specify this operand for documentation purposes if your installation also uses VTAM.

BSTRNO=number
specifies additional buffers and strings that are required whenever a path is opened to handle the base cluster of an alternate index (AIX). It specifies the number of strings that VSE/VSAM is to allocate internally for access to the base cluster of a path.

BSTRNO applies only in conjunction with *data set name sharing*.

If you omit the operand or specify BSTRNO=0, the number of internally created strings is twice that specified in STRNO.

No dynamic increase of string numbers is possible under VSE/VSAM.

If the value specified in BSTRNO is insufficient, requests could fail. BSTRNO is accepted if the sum of the values in STRNO and BSTRNO does *not* exceed 255; this applies even if the opened ACB does not belong to the path.

It is important to define a reasonable value with the *first* ACB that opens for data set name (DSN) sharing. This is necessary, because VSE/VSAM ignores the BSTRNO values of subsequently opened ACBs with DSN sharing to the same data set.

For further information and considerations, refer to "Data Set Name Sharing" on page 192.

BUFND=number
specifies the number of I/O buffers to be used to hold CIs containing data records. Every buffer is the size of one data CI. The allowable minimum specification (and also the default) is the number specified for STRNO, plus one. (The default for STRNO is one.) If you specify the BUFND operand, but your specification is less than the minimum, VSE/VSAM overrides your specification and uses the minimum. However, VSE/VSAM issues no message to inform you of this.

VSE/VSAM increases the number of data buffers you specify if the amount of virtual storage available for buffers differs from the storage requirements indicated by the BUFND and BUFNI operands. See the BUFSP operand for an explanation. For examples of BUFND use, see "Buffer Specification" on page 95.

BUFNI=number
specifies the number of I/O buffers to be used to hold index CIs (index records). Every buffer is the size of one index CI. The minimum number you can specify is the number specified for the STRNO operand. (If you omit STRNO, BUFNI must be at least one, because the default for STRNO is one.) If BUFNI is omitted, the default is the number specified for STRNO, because the smallest number of index buffers allowed is one for every string. If you specify the BUFNI operand, but your specification is less than the minimum, VSE/VSAM overrides your specification and uses the minimum. However, VSE/VSAM issues no message to inform you of this.

VSE/VSAM increases the number of index buffers you specify if the amount of virtual storage available for buffers differs from the storage requirements indicated by the BUFND and BUFNI operands. See the BUFSP operand for an explanation. For examples of BUFNI use, see "Buffer Specification" on page 95.

BUFSP=number
specifies the size, in bytes, of an area for data and index I/O buffers.

VSE/VSAM issues a GETVIS macro to obtain the buffer area in your processing partition. It must be at least as large as the buffer space size recorded in the catalog entry for the file. If your specification is too small, VSE/VSAM overrides it and uses the value recorded in the catalog for buffer space size. However, VSE/VSAM issues no message to inform you of this.

If you do not specify the BUFSP operand, the buffer space size will be the larger of (1) the size recorded in the catalog or (2) the size determined from the values specified for BUFND and BUFNI. (The size recorded in the catalog was specified by the BUFFERSPACE parameter in the DEFINE command of IDCAMS. If that parameter was omitted when the file was defined, a default value was set in the catalog. This default value, the minimum amount of buffer space allowed by VSE/VSAM, is enough space for two data CIs and one index CI.)

You can also specify buffer space by means of the BUFSP=number operand on the DLBL statement that identifies the file to be processed. This value overrides the BUFSP operand in the ACB macro. It also overrides the BUFFERSPACE parameter in the DEFINE command if it is greater than the BUFFERSPACE parameter value.

If the values you code for BUFND, BUFNI, and BUFSP are not consistent with each other, VSE/VSAM increases the number of buffers to conform to the size of the buffer area.

If BUFSP is greater than the minimum requirements and greater than the BUFND and BUFNI requirements, the extra space will be allocated between data and index buffers as follows:

- If the ACB MACRF operand indicates direct processing only:

  First, BUFND and BUFNI are allocated as specified. Then, all additional space is allocated to index buffers.
- If the ACB MACRF operand indicates sequential processing:

  First, BUFND and BUFNI are allocated as specified. Then, one additional buffer is allocated to the index and the remaining space is allocated to data buffers. Any space remaining, but insufficient for a single data buffer, is allocated to an index buffer.

If BUFSP is greater than the minimum requirements, but less than the BUFND and BUFNI requirements, the buffer space will be changed to conform to the BUFND and BUFNI requirements.

If you provide your own pool of I/O buffers for CI (CNV) access (MACRF=UBF), the BUFND, BUFNI, and BUFSP operands have no effect. The AREA and AREALEN parameters in the RPL macro then define the area for user buffers.

For examples of BUFSP use, see “Buffer Specification” on page 95.

DDNAME=filename
specifies a character string of up to seven bytes and is the same as the filename parameter specified in the DLBL statement that identifies the VSE/VSAM file or path to be processed.

If the ‘file ID’ in the DLBL statement indicates a path, but you want to process only the alternate index of the path, you must specify MACRF=AIX (see the discussion of the MACRF operand). If the file ID does not indicate a path, the AIX option is ignored.

If you omit the DDNAME operand, you must specify the DLBL filename as the name (label) of the ACB macro.

EXLST=address
specifies the address of a list of user exit-routine addresses. The list is generated by the EXLST macro (or the GENCB macro). If you use the EXLST macro, you can specify its name (label) here as the address of the exit list. If you use the GENCB macro, you can specify the address of the EXLST returned by GENCB in Register 1. Omitting this operand indicates that you have no user exit routines.

MACRF=
specifies the kind(s) of processing you will do with the file. The options must be meaningful for the file. For example, if you specify keyed access for an entry-sequenced file, you will not be able to open the file. You must specify all of the types of access you are going to use, whether you use them concurrently or by switching from one to the other.

For information on the interaction between the DLBL DISP parameter and the ACB MACRF specification when a file is opened, refer to "File Disposition" on page 29.

Mutually exclusive options are:

```
AIX and NRM       NDF and DFR
IN and RST        LSR and UBF
NRS and RST       LSR and RST
NUB and UBF       NSR and DFR
NSR and LSR       NCM and CMP
```

The following restrictions apply to a SHAREOPTIONS 4 key-sequenced output file:

- One ACB cannot specify both KEY and ADR (or both KEY and CNV). Attempts to do this result in an OPEN failure.
- If the file is open for output under one ACB for keyed access, an attempt to open it under another ACB with MACRF=(OUT,ADR) or MACRF=(OUT,CNV) will fail.
- If the file is open for output under one ACB for addressed or CI access, an attempt to open it under another ACB with MACRF=(OUT,KEY) will fail.

Options of the MACRF Parameter

The following explains the MACRF parameter options. The options are arranged in groups, where every group has a default value (highlighted). You can specify options in any order. You may specify both DIR and SEQ; with keyed access, you may specify SKP as well. If you specify OUT and want simply to retrieve some records as well as update, delete, or insert others, you need not also specify IN. You may specify both ADR and KEY to process a key-sequenced file.

ADR
Addressed access (for key-sequenced and entry-sequenced files).

CNV
CI access.

KEY
Keyed access (for key-sequenced or relative-record files).

DDN
specifies to open the data set according to the DDNAME specification.

DSN
specifies that VSE/VSAM handles the first ACB that is opened as if you specified DDN. However, VSE/VSAM also:
- Remembers that a *DSN structure* was built.
- Connects the second and all following ACBs that open the first data set to the structure of the first ACB.

If DSN is specified, VSE/VSAM shares control blocks and I/O buffers.

For further information and considerations, refer to "Data Set Name Sharing" on page 192.

DFR
Write operations are to be deferred when possible.

NDF
Write operations are not to be deferred.

DIR
Direct processing.

SEQ
Sequential processing.

SKP
Skip sequential processing (for key-sequenced or relative-record files).

IN Retrieve records only.

OUT
Retrieve, insert, add-to-end, or update records (keyed access); retrieve, update, or add-to-end (addressed access).

NCM
All data records exchanged between VSE/VSAM and the application are in uncompressed (expanded) format. If the file is in COMPRESSED format, MACRF=CNV must not be specified.

CMP
If the file is in COMPRESSED format, all data (records or control intervals) exchanged between VSE/VSAM and the application are in compressed format. The record includes the compressed record prefix (see "Data Format of Records" on page 69). SHOWCB and TESTCB also take the prefix length into account for the LRECL, RKP, and RECLEN keywords, if MACRF=CMP is specified.

NRM
The file (or path) named in the DDNAME operand or in the DLBL statement is to be processed.

AIX
The alternate index of the path specified in the DDNAME operand is to be processed. If no path is specified there, this option is ignored. The AIX option causes the path restrictions (that is, the restrictions limiting the access through a path) to be ignored so that the alternate index can be processed like a key-sequenced file. The alternate index of the path can be opened for input (IN), output (OUT), or it can be reset (RST), provided it was defined with the REUSE attribute (in the DEFINE ALTERNATEINDEX command).

NRS
The Open routine does not reset the file to 'empty'. Output operations will cause updating or extension of the existing record. DISP=OLD on the DLBL statement is equivalent to MACRF=NRS and will override MACRF=RST.

RST
The Open routine resets the catalog information about the file (cluster or alternate index) to its original status, that is, to the status it had before it was opened for the first time. The file must have been defined with the REUSE attribute for RST to be effective. Although the file is not erased, you can handle it like a new file and use it as a work file. After the Open routine has performed the reset operation, the RST option is equivalent to the OUT option. DISP=NEW on the DLBL statement is equivalent to MACRF=RST and will override MACRF=NRS.

NSR
Non-shared resources (normal operation).

LSR
Local shared resources (LSR).

When you specify LSR in the ACB, VSE/VSAM ignores the BUFND, BUFNI, BUFSP, and STRNO operands, because it uses the BUFFERS and STRNO values specified in the BLDVRP macro.

For more information, see "Sharing Resources Among Files and Displaying Catalog Information" on page 191.

NUB
No user buffers; VSE/VSAM supplies buffers for I/O operations (KEY, ADR, and CNV access).

UBF
User buffers (only CNV access can be specified). VSE/VSAM will read and write CIs in a buffer you supply. It is pointed to by the AREA parameter of the RPL.

MAREA=address
specifies the address of an optional OPEN/CLOSE/TCLOSE message area. (See "OPEN/CLOSE/TCLOSE Message Area" on page 204.)

MLEN=number
specifies the length of an optional OPEN/CLOSE/TCLOSE message area. The default is 0, the maximum value can be 32,768 bytes.

PARMS=(CLOSDSP=options DSNAME=address)

CLOSDSP=options
specifies the CLOSE disposition for a reusable file. Options specified in the DLBLs DISP=(,option) JCL statement override the options specified in this parameter.

If a second option (either KEEP or DELETE) is specified, this indicates whether the file should be kept or deleted if it was opened during a job that ended abnormally. For example, if you open a file with PARMS=(CLOSDSP=(DELETE,KEEP)) specified, then this file is deleted only if the job comes to a *normal* end. In any other case, the file is kept so that you can rerun the job without reloading the file.

KEEP
indicates that the file and its contents are to be preserved. DISP=(,KEEP) on the DLBL statement is equivalent to PARMS=(CLOSDSP=KEEP) and will override any CLOSDSP specified in the ACB.

DELETE
indicates that the file and its contents need not be preserved.

VSE/VSAM is free to release resources associated with the file. DISP=(,DELETE) on the DLBL statement is equivalent to PARMS=(CLOSDSP=DELETE) and will override any CLOSDSP specified in the ACB.

DATE
indicates that disposition depends on the current system date and the file's expiration date. If the expiration date is yet future relative to the system date, the file is treated as though KEEP were specified. Otherwise the file is treated as though DELETE were specified. DISP=(,DATE) on the DLBL statement is equivalent to PARMS=(CLOSDSP=DATE) and will override any CLOSDSP specified in the ACB.

If disposition parameters indicate that file resources can be freed, VSE/VSAM releases as many resources as allowed by the sharing status and by the characteristics defined for the file. For details, refer to "File Disposition" on page 29.

DSNAME=address
specifies the address of a 88 byte area that contains the data set names of the cluster and the catalog that contains the data set. The DSNAME operand allows to open a file without referring to a matching label (DLBL). The format of the area pointed to by `address` is:

Offset Dec	Hex	Bytes	Description
0	0	44	Entry name of cluster or component to be used
44	2C	44	Entry name of the catalog

PASSWD=address
specifies the address of a field that contains the highest-level password required for the type(s) of access indicated by the MACRF operand. The first byte of the field pointed to contains the length (in binary) of the password (maximum of 8 bytes). A zero in the length byte indicates that no password is supplied. If the file is password-protected and you do not supply a required password in the ACB, VSE/VSAM gives the console operator the opportunity to supply it when opening the file.

RMODE31=ALL|BUFF|CB|NONE
specifies whether VSE/VSAM OPEN is to obtain virtual storage for I/O buffers above or below the 16MB line of address space. The default is NONE.

Note: The internal VSE/VSAM buffers and control blocks are not affected by the RMODE31 parameter. If possible, VSE/VSAM places such buffers and control blocks above the 16MB line.

(Internal VSE/VSAM buffers are, for example, NSR index buffers, path buffers, and upgrade set buffers.)

ALL|BUFF specifies that VSE/VSAM I/O buffers may to be obtained from *above* the 16MB line.

CB|**NONE** specifies that VSE/VSAM I/O buffers must be obtained from *below* the 16MB line.

ALL and *CB* are allowed for reasons of DFSMSdfp compatibility.

SHRPOOL=number
identifies which LSR pool is to be connected to the ACB. This parameter is

only valid when MACRF=LSR is also specified. For *number* specify the identification number of the shared pool; it can be a number from 0 through 15. The default is 0.

STRNO=number
indicates how many concurrent active requests VSE/VSAM is to handle. The maximum value is 255. The default is one.

During initial load of a file, VSE/VSAM ignores your specification and sets the value to one because a file can be loaded by one string only. After the file is loaded and successfully closed, it can be reopened and processed by as many strings as specified under STRNO.

Several requests, with the corresponding RPLs pointing to the same ACB, can be active at the same time. Thus, you can access simultaneously (a) different parts of a file, and (b) in different modes of operation (sequential and direct, or keyed and addressed, for example). You may, for example, process one part of a file sequentially (forward or backward) and intermix sequential processing with direct requests to any part of the file, without affecting the sequential positioning.

Every request is activated by its own RPL and action (GET, PUT, etc.) macro. Positioning information is maintained separately for every RPL, so that every request can be processed independently from all other requests.

A request is defined either by a single RPL or by a chain of RPLs (see "RPL: Specifying the Request Parameter List" on page 186). Specify for STRNO the total number of RPLs or chains of RPLs that you will use to define requests. For a chain of RPLs VSE/VSAM needs to remember only one position. However, every position beyond the minimum number that VSE/VSAM needs to remember requires additional virtual-storage space for:

- A minimum of one data I/O buffer and, for keyed access, one index I/O buffer (the size of an I/O buffer is the CI size of a file).
- Internal control blocks and other areas.

VSE/VSAM remembers its position in the file for any sequential or update request. For example, sequential access depends on VSE/VSAM being able to determine the location of the next record from the location of the present record. Updating or deleting a record depends on VSE/VSAM remembering its location after you retrieve it. Also, processing a record in the I/O buffer requires VSE/VSAM to remember its location in the buffer.

Note: If the number of concurrent requests (RPLs or chain of RPLs) exceeds the number you have specified in the ACB STRNO operand, you must disconnect a request from its RPL by means of the ENDREQ macro before you can issue another concurrent request. The ENDREQ macro is discussed under "The ENDREQ Macro" on page 213.

OPEN/CLOSE/TCLOSE Message Area

Providing the Area

After you have issued an OPEN, CLOSE, or TCLOSE macro, the ACB error code is either zero, indicating that the files were opened or closed successfully, or non-zero, indicating that a warning or error condition occurred. You can examine this code by specifying the ERROR keyword in the SHOWCB or TESTCB macro. However, during an OPEN, CLOSE, or TCLOSE more than one warning or error condition may be detected, in which case the error code which you get when you specify the ERROR keyword reflects only the warning or error condition which

occurred last. The error code does not indicate any other (earlier) conditions which might have occurred during the OPEN, CLOSE, or TCLOSE.

In order to save such multiple warning or error conditions, you can provide a message area in which those conditions are to be stored, together with additional information. This message area is connected to the ACB when you specify the following parameters in the ACB macro:

```
MAREA=address,MLEN=length
```

MAREA must be allocated below 16MB. If MAREA or MLEN are not specified or a length of zero has been specified for MLEN, no area is provided and the ACB error code is then the only indication for errors or warnings which occurred during OPEN, CLOSE, or TCLOSE. If you have specified both MAREA and MLEN (with a non-zero value) and error or warning conditions are detected, appropriate messages are stored into the message area.

The OPEN/CLOSE/TCLOSE message area is also used by VSE/VSAM record management if resources such as buffers and control blocks are shared among files. If a GET request is issued for a file using the common resource pool, it can happen that (owing to deferred write operations for PUT requests) the resource pool is filled up with modified buffers forcing VSE/VSAM to write a buffer for another file before it can satisfy the GET request. If an error occurs in writing out such a buffer, this is indicated in the message area, together with the ACB name of the affected file.

The message information provided by VSE/VSAM consists of the message area header and the message list.

The message area header contains statistical, pointer, and general information. Its contents are unrelated to the individual messages.

Format of the Message Area Header

The format of the message area header is:

Byte **Meaning**

0 Flag byte

Bit 0=1:
At least one warning or error condition has occurred and the complete header is stored.

Bit 0=0:
Either no warning or error condition has occurred or the message area is too short for the complete header. No further header information and no messages are stored.

Bits 1-7
Reserved (set to binary zero)

1-2 Length of message area header

3 Request type code:
X'01' OPEN
X'02' CLOSE
X'03' TCLOSE
X'04' GET (for shared resources only)

4-11 Filename used for ACB

12-13 Total number of error or warning conditions issued by OPEN, CLOSE, TCLOSE, or (for shared resources only) by record management

14-15 Number of messages stored into message area

16-19 Address of message list, that is, of first message in the message area

Apart from the flag byte, message area header information is stored only when a warning or error condition was detected (the ACB or RPL error code is non-zero) and the length of the message area (MLEN) is large enough to accommodate the full message area header. Thus, before accessing bytes 1-19 of the message header information, you should test byte 0 to see whether further information was stored at all.

The message list contains the individual messages corresponding to the warning or error conditions detected. It is pointed to by bytes 16-19 of the message area header. Within the message list, the individual messages are stored continuously one after another in the form of variable-length records. The number of messages stored is contained in the message area header (bytes 14-15). Before investigating the message list, you should check whether the stored-message count is zero or greater than zero. The format of a message is as follows:

Byte **Contents**

0-1 Total length of message (including length bytes).

2 ACB error code corresponding to error or warning condition represented by this message, or (for shared resources) RPL error code indicating write error.

3 Function-type code (indicates the component which produced the error or warning condition and the state of the upgrade set):

- **X'00'** Condition occurred during accessing the base cluster. Upgrade set is correct.
- **X'01'** Condition occurred during accessing the base cluster. Upgrade set may be incorrect as a consequence of this request.
- **X'02'** Condition occurred during accessing the AIX over a base cluster. Upgrade set is correct.
- **X'03'** Condition occurred during accessing the AIX over a base cluster. Upgrade set may be incorrect as a consequence of this request.
- **X'04'** Condition occurred during upgrade processing. Upgrade set is correct.
- **X'05'** Condition occurred during upgrade processing. Upgrade set may be incorrect as a consequence of this request.
- **X'06'** Condition occurred during writing of a buffer which does not belong to the file for which the GET request was issued (for shared resources only).

4-nn File ID of the component indicated by the function-type code (up to 44 bytes), or (for shared resources) name of the ACB identifying the file for which the buffer write error occurred.

Exceptional Conditions for the Message Area

- Message area (MLEN) is too small to contain the complete message area header:
  Byte 0, bit 0=0: The whole message area is overwritten with binary zeros.
- Message area is too small to contain a complete message:

Byte 0, bit 0=1: The header exists.
Bytes 12-13 not equal to 0: Warning or error conditions have occurred.
Bytes 14-15=0: No message stored.

- Message area is too small to contain all messages:
  Byte 0, bit 0=1: The header exists.
  Bytes 12-13 not equal to 0: Warning or error conditions have occurred.
  Bytes 14-15 not equal to 0: If the value in bytes 14-15 is lower than in bytes 12-13, then not all messages have been stored.

The BLDVRP Macro

To share resources, you must provide a resource pool which you build by issuing the BLDVRP (build the VSE/VSAM resource pool) macro. Issuing BLDVRP causes VSE/VSAM to share the I/O buffers, I/O control blocks, and channel programs of those files whose ACBs indicate the local shared resource (LSR) option. Control blocks and channel programs are shared automatically; you can control the sharing of buffers.

When issuing BLDVRP, you must specify one or more *buffer pools* within the resource pool, and also the size and number of buffers in every buffer pool. A file uses the buffer pool whose buffer size exactly matches the file's CI size or, if this CI size is not available, the buffer pool with the next-larger buffer size. The file uses only the one buffer pool.

To share resources, you must do all of the following:
- Issue the BLDVRP macro to build a resource pool.
- Code the LSR option in the MACRF operand in the ACBs of your files.
- Issue OPEN to connect these files to the resource pool.

When you issue a BLDVRP macro, Register 13 must contain the address of a 72-byte save area that you are providing. When you issue a BLDVRP macro from within one of your exit routines such as LERAD or SYNAD, your program must provide a second 72-byte save area for use by VSE/VSAM, because the original save area is still in use by the external VSE/VSAM routine.

Deciding How Big a Pool to Provide

You have to provide a resource pool before any clusters or alternate indexes are opened to use it. Specifying the BUFFERS, KEYLEN, and STRNO operands of the BLDVRP macro requires knowledge of the size of the CIs, data records (if spanned), and key fields in the components that will use the resource pool and knowledge about the way the components are processed.

Displaying Information about an Unopened File

The SHOWCAT macro enables you to 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 of the components to find out the sizes of CIs, records, and keys. This information enables the program to calculate values for the BUFFERS and KEYLEN operands of BLDVRP.

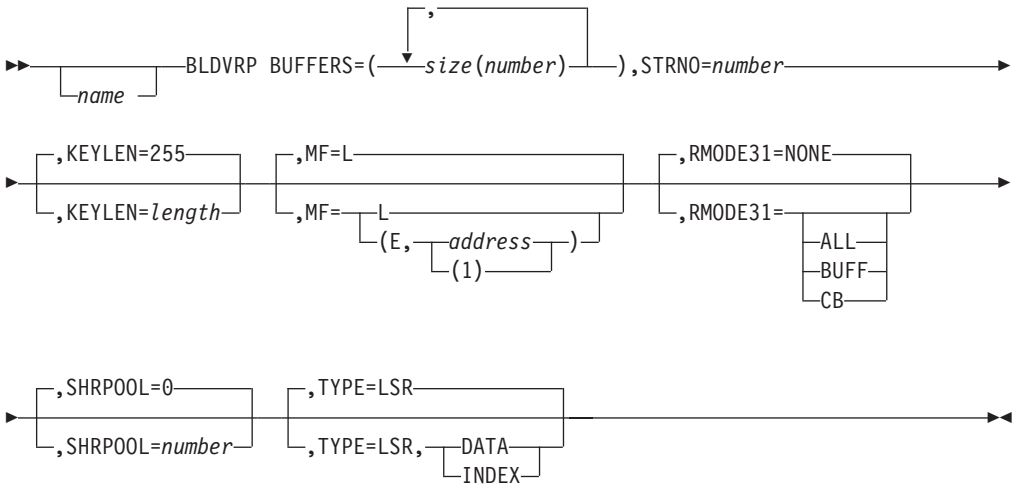
Displaying Statistics about a Buffer Pool

You can get statistics about the use of buffer pools to determine how you could improve a previous definition of a resource pool and the mix of files that use it. The SHOWCB macro enables you to get statistics about a buffer pool. The statistics are available from an ACB that describes an open file 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 of the statistics except one are for a single buffer pool. To get statistics for the whole resource pool, issue SHOWCB for every buffer pool in it.

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.

Format of the BLDVRP Macro



name
one to eight characters that provide a symbolic name.

BUFFERS=size(number)
specifies the size and the number of buffers in every buffer in the resource pool. The number of buffer pools in the resource pool is implied by the number of **size(number)** pairs you specify. The buffer sizes should normally be set equal to the CI sizes of the files to be processed. (You can find out the CI size of a file by issuing the SHOWCAT macro for that file.) If you do not specify the exact buffer(=CI)size for a file, VSE/VSAM will use buffers from the buffer pool with the next larger buffer size.

When you process a key-sequenced file, the index component, as well as the data component, shares the buffers of a buffer pool. When you use an alternate index to process a base cluster, the components of the alternate index and the base cluster share buffers. The components of alternate indexes in an upgrade set share buffers. Buffers of the appropriate size and number must be provided for all of these components, each of which uses the buffer pool whose buffers are exactly the right size or the next-larger size.

size is 512, 1024, 2048, 4096, or so on in increments of 4096 to a maximum of 32KB.

number is at least 3 but must not exceed 32767.

KEYLEN=length
specifies the maximum key length of the files that are to share the resource pool. The default is 255.

The keys whose lengths must be provided for are the prime key of every KSDS and the alternate key of every alternate index that is used for processing or is upgraded. The key length (relative record number) of a relative record file is 4. If the buffer pool is to contain for entry-sequenced files only, specify KEYLEN=0. (You can find out the key length of a file by issuing the SHOWCAT macro for that file.)

MF=L
indicates that this is the list form of BLDVRP. The list form builds a parameter list when the macro is assembled. It is not executable. If you do not specify STRNO in the list form of BLDVRP, you must specify it in the execute form.

The list form of the BLDVRP macro is especially useful when the buffer sizes of the VSE/VSAM files are not known. In that case you can first retrieve from the VSE/VSAM catalog the CI sizes of the files to be processed via the SHOWCAT macro and then enter these values in the BLDVRP parameter list.

The format of the BLDVRP parameter list is described in "The BLDVRP Parameter List" on page 312.

MF=(E)
indicates that this is the execute form of BLDVRP. address is the address of the parameter list built by the list form of BLDVRP.

If you use register notation, you may use Register 1, as well as any register from 2 through 12. The execute form puts the address of the parameter list in Register 1 and passes control to VSE/VSAM to process the list. You may, however, first change the values for STRNO and/or KEYLEN (which are both optional in the execute form of BLDVRP). BUFFERS may not be specified in the execute form of BLDVRP, because this operand affects the length of the parameter list.

If the MF operand is omitted, the standard form of the BLDVRP macro is assumed, which builds a parameter list, puts its address in Register 1, and passes control to VSE/VSAM to process the list.

RMODE31=ALL|BUFF|CB|NONE
specifies where the I/O buffers for the LSR pool are to reside. (The pools are identified in the SHRPOOL keyword.) The default is NONE.

ALL|BUFF specifies that the buffers may reside *above* the 16MB line.

CB|NONE specifies that the buffers must reside *below* the 16MB line.

ALL and *CB* are allowed for reasons of DFSMSdfp compatibility.

SHRPOOL=number
specifies the identification number of a shared resources pool that is to be build. Specify a number from 0 through 15. The default is 0.

STRNO=number
specifies the maximum number of requests that may be issued concurrently for all of the files that are to share the resource pool. The number must be at least one and no more than 255.

If you want to find out how effectively your resource pool is utilized during execution, you can display the maximum number of requests which were concurrently active because the resource pool was built by issuing a SHOWCB ACB=...,FIELDS=(STRMAX) in your processing program. Depending on the result, you may want to redefine STRNO=number the next time you build your resource pool. (You cannot redefine the pool while it is in use.)

The ACB specified in the SHOWCB macro can be any ACB that describes an open file that is using the resource pool.

TYPE=LSR(,DATA|INDEX)
Allows definition of separate LSR pools for data and index. If only LSR is specified, there is one LSR pool for both data and index. Definition of an INDEX LSR pool requires a previous definition of a DATA LSR pool. The form "TYPE=LSR" is implemented for compatibility with z/OS (DFSMSdfp). No type other than LSR (such as GSR on z/OS) is accepted by VSE/VSAM.

Return Codes from BLDVRP

When VSE/VSAM returns to your processing program after a BLDVRP request, Register 15 contains one of the following return codes:

Return Dec	Code Hex	Meaning
0	X'00'	VSE/VSAM completed the request.
4	X'04' No new pool was build.	The resource pool already exists in the partition.
12	X'0C'	The request was not executed because an error was encountered while VSE/VSAM routines were loaded (for example, CDLOAD failed), or there was insufficient GETVIS space for the partition that uses the BLDVRP macro.
20	X'14'	STRNO is less than one or greater than 255.
24	X'18'	BUFFERS is specified incorrectly: size or number is invalid.
28	X'1C'	SHRPOOL is less than 0 or greater than 15.
36	X'24'	BLDVRP was issued to build an index resource pool, but the required corresponding data resource pool does not exist.

Connecting a File to a Resource Pool

After having built a resource pool, you cause a file to use that pool by specifying the SHRPOOL=number and MACRF=(LSR) operands of the file's ACB before you open the file, thus:

```
ACB   SHRPOOL=number,MACRF=(LSR)
```

When you have specified LSR in the ACB, VSE/VSAM ignores the BUFND, BUFNI, BUFSP, and STRNO operands, because it uses the BUFFERS and STRNO values that you have specified in the BLDVRP macro.

Restrictions

UBF (user buffering) may not be specified together with LSR. LSR may not be specified in the ACB of an empty file (which implies that the file is to be loaded).

Apart from the standard error codes from the Open routine, you may get additional error codes in the ACB ERROR field when you try to open a file with the LSR option. These error codes are listed in the "VSE/VSAM return and error codes" section of the *z/VSE Messages and Codes*.

The CLOSE Macro

After your last request for access to the file, you will normally issue a CLOSE macro to complete processing of that file and disconnect your program from the file. If you have not issued a CLOSE macro before end-of-job or if your job terminates abnormally, your file might not be closed properly and subsequent processing of that file might cause errors.

Because it is essential for the integrity of a file that it is closed properly, z/VSE automatically attempts to close all open VSE/VSAM files in the partition at both normal and abnormal termination of a job step. If any control blocks for a file have been destroyed through an error in your program, this file cannot be closed and a message is given to the operator. EXLST routines are not entered during an automatic CLOSE.

The TCLOSE macro performs the functions of CLOSE, except that it leaves the program and the file connected so that you can continue processing without reopening the file.

The Close routine completes any I/O operations that are outstanding when a processing program issues a CLOSE macro for a file. It writes any output buffers that have not been stored.

The Close routine updates the catalog entries of the file, including pointers to the end of the file and statistics on file processing (such as number of records inserted). If the file was loaded and the SPEED option was specified (in the DEFINE command), the Close routine formats the last CA in the file to ensure that the entire file is accessible.

The Close routine restores the ACB to the status that it had before the file was opened and frees the virtual storage that the Open routine used to construct VSE/VSAM control blocks.

You must specify a CLOSE macro to change from loading a file to retrieving records from that file in the same run.

Format of the CLOSE Macro

name
one through eight characters that provide a symbolic name.

address
specifies up to 16 addresses of ACBs and DTFs that define the file(s) to be closed. You can specify address:

- In register notation, using a register from 1 through 12. Specify within parentheses.

  Or
- With an expression that generates a valid relocatable A-type address constant.

A return code is set in Register 15 to indicate whether the ACBs were closed successfully. ACBs should be coded together (following the DTFs) to apply to all of them. If, for example, you coded:

```
CLOSE  ACB1,DTF1,ACB2
```

the return code will apply to ACB2 only. If ACB2 closed successfully and ACB1 did not, the return code will still be X'00'. (The Close routine sets Register 15 to zero when it receives control after a DTF has been closed.) To ensure that the return code is valid and applies to both ACBs, write the macro in the following way:

```
CLOSE  DTF1,ACB1,ACB2
```

The Close routine sets one of the following return codes in Register 15:

Return Code	Meaning
X'00'	All ACBs were closed successfully.
X'04'	One or more ACBs were not closed successfully.
X'08'	One or more Close routines could not be loaded because there was not enough virtual storage space, or the modules could not be found. Processing cannot continue.

If Register 15 contains X'04', an error code is set in one or more ACBs. You can use the ERROR keyword of the SHOWCB or TESTCB macro to examine the error code. For an explanation of the VSE/VSAM CLOSE (and TCLOSE) error codes, see the "VSE/VSAM return and error codes" in the *z/VSE Messages and Codes*.

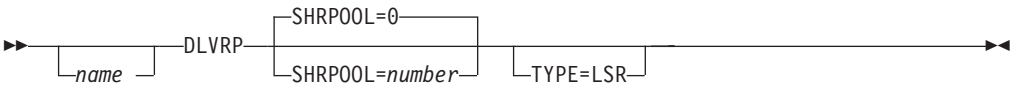
The DLVRP Macro

After all the files using the resource pool have been closed, you must delete the resource pool by issuing the DLVRP (delete VSE/VSAM resource pool) macro.

If you do not delete the resource pool with the DLVRP macro, it will automatically be deleted at the end of the job step, because it resides in virtual storage, which is invalidated at the end of a job step.

When you issue a DLVRP macro, Register 13 must contain the address of a 72-byte save area that you are providing. When you issue a DLVRP macro from within one of your exit routines such as LERAD or SYNAD, your program must provide a second 72-byte save area for use by VSE/VSAM, because the original save area is still in use by the external VSE/VSAM routine.

Format of the DLVRP Macro



name
one to eight characters that provide a symbolic name.

SHRPOOL=number
specifies the identification number of a shared resources pool that is to be deleted. Specify a number from 0 through 15. The default is 0.

TYPE=LSR
This parameter is accepted by VSE/VSAM for compatibility with zOS. It is ignored, however, since no type other than LSR is available under VSE/VSAM. See also BLDVRP. Separate deletion of data and index LSR pools is not possible because they form a unit and must not be deleted individually.

Return Codes from DLVRP

When VSE/VSAM returns to your processing program after a DLVRP request, following return codes:

Return Dec	Code Hex	Meaning
0	X'00'	VSE/VSAM completed the request.
4	X'04'	There is no resource pool to be deleted.
8	X'08'	There is at least one other open file using the resource pool.
12	X'0C'	The request was not executed because an error was encountered while VSE/VSAM routines were loaded (for example, CDLOAD failed).
28	X'1C'	SHRPOOL is less than 0 or greater than 15.

The ENDREQ Macro

When you issue an ENDREQ macro, Register 13 must contain the address of a 72-byte save area that you are providing. When you issue the macro from within one of your exit routines such as LERAD or SYNAD, you must provide a second 72-byte save area for use by VSE/VSAM.

This macro causes VSE/VSAM to end a request, that is, to forget its position for the specified RPL and to release its associated buffers for use by another RPL. Before you can issue a request specifying an RPL for which an ENDREQ macro was executed, you have to reposition VSE/VSAM.

An ENDREQ macro is required in your program whenever you have already issued as many concurrent active requests as you have specified for STRNO operand of the ACB and you want to issue yet another request. (Refer to the discussion under "VSE/VSAM is Not Yet Positioned" on page 225.)

If an I/O operation was started, it will be allowed to complete. Also, I/O operations required to maintain the integrity of the file will be performed.

If the request involves a chain of RPLs, all records specified by the request may not be processed. For example, two RPLs are chained in a PUT request to add two new records to the file and an ENDREQ is issued after VSE/VSAM started the I/O operation to add the first record. That I/O operation will be completed and, if it causes a CI split, subsequent I/O operations will be performed to complete the split and update the index. However, VSE/VSAM will then return control to the processing program without adding the second record.

The ENDREQ macro causes VSE/VSAM to cancel the position in the file established for that request and also invalidates data and index buffers to force refreshing of all requests subsequent to the end request. There is, however, *no* buffer invalidation for:
- SHAREOPTION 1 files
- SHAREOPTION 2 files opened for output

- Higher level index buffers (only sequence set invalidation).

Format of the ENDREQ Macro

name
one through eight characters that provide a symbolic name.

RPL=address|(1)
specifies the address of the RPL (or first RPL in a chain of RPLs) that defines the request to be terminated. You can specify address:
- In register notation, using a register from 2 through 12. Specify within parentheses, or
- With an expression that generates a valid relocatable A-type address constant.

The ERASE Macro

When you issue an ERASE macro, Register 13 must contain the address of a 72-byte save area that you are providing. When you issue the macro from within one of your exit routines such as LERAD or SYNAD, you must provide a second 72-byte save area for use by VSE/VSAM.

This macro deletes the record previously retrieved for update (with the GET macro, OPTCD=UPD). You can delete records in a key-sequenced file by keyed or addressed access, but you cannot delete records in an entry-sequenced file. You can delete records in a relative-record file by keyed access. You cannot delete CIs (OPTCD=CNV).

Format of the ERASE Macro

name
one through eight characters that provide a symbolic name.

RPL=address|(1)
specifies the address of the RPL (or the first RPL in a chain of RPLs) that defines the ERASE request. You can specify address:
- In register notation, using a register from 2 through 12. Specify within parentheses.

  Or
- With an expression that generates a valid relocatable A-type address constant.

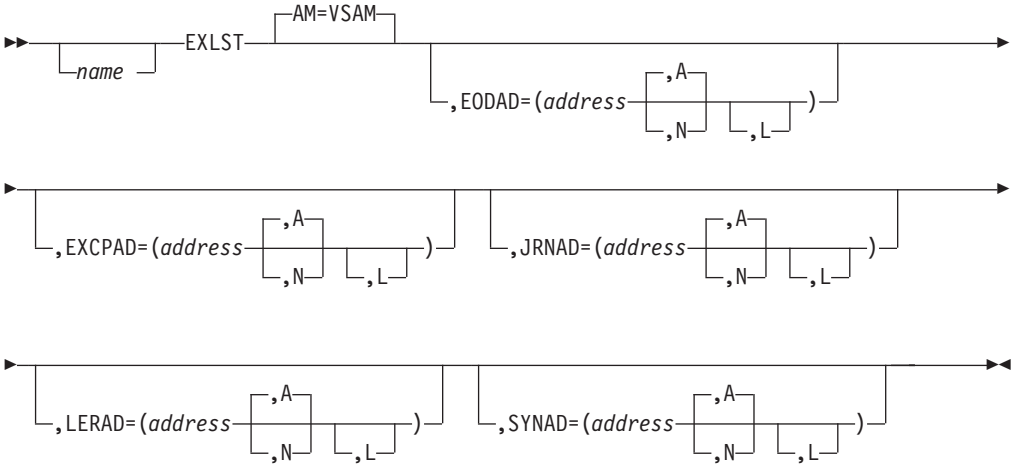
The EXLST Macro

Assembly of the EXLST (exit list) macro produces an optional list of addresses of user exit routines. An exit routine is entered when VSE/VSAM detects the condition (such as an I/O error) that the routine is supposed to handle. The exit list is associated with an ACB by the EXLST operand of the ACB macro. Two or more ACBs can refer to the same exit list.

The number of exit addresses in a list is variable and depends on the number of operands you code. You cannot add addresses to the list after it is generated, but you can change an address or the indication of whether or not an exit is active (with the MODCB macro).

Values for EXLST macro operands can be specified as codes and expressions that generate valid relocatable A-type address constants. Do not use register notation.

Format of the EXLST Macro



name
one through eight characters that provide a symbolic address for the exit list that is established.

AM=VSAM
specifies that this is a VSE/VSAM control block. You may want to specify this operand for documentation purposes if your installation also uses VTAM.

EODAD
specifies that an exit is provided for special processing when the end of a file is reached by sequential or skip sequential access.

EXCPAD
specifies that an exit is provided to receive control from VSE/VSAM when an I/O operation is started.

JRNAD
specifies that an exit is provided for journalizing as you process data records.

LERAD
specifies that an exit is provided for analyzing logic errors.

SYNAD
specifies that an exit is provided for analyzing physical errors.

address
is the address of a user-supplied exit routine. The address must always be specified first.

A|N
specifies that the exit routine is active (A) or not active (N). VSE/VSAM does not enter a routine whose exit is marked not active.

L specifies that the address is the address of an eight-byte field that contains the name of a phase that VSE/VSAM is to load for exit processing. If L is omitted, the address gives the entry point of the exit routine in virtual storage. L can precede or follow the A or N specification.

EODAD Exit Routine to Process End-of-File

An EODAD routine finishes the processing of a file when VSE/VSAM reaches the end of the file. VSE/VSAM exits to this routine when: (1) you attempt to sequentially retrieve or point to a record beyond the last record in the file, that is, the record with the highest key or the highest relative-record number (for keyed access), or with the highest RBA (for addressed access); (2) during sequential backward retrieval when the records in reverse sequence are exhausted or; (3) when you have specified CI access and user buffers and there is no more data after a GET request or a PUT for update request.

If your program retrieves records sequentially with a request defined by a chain of RPLs, your EODAD routine must determine whether the end of the file was reached for the first RPL in the chain. If not, then one or more records have been retrieved but not yet processed by your program.

If you do not have this exit routine, VSE/VSAM exits to the routine for analyzing logic errors (see the LERAD operand). If you do not have the LERAD exit routine, VSE/VSAM returns to your processing program at the instruction following the last executed instruction. In that case, Register 15 contains X'08', and register 1 contains the address of the RPL. Your program can examine the feedback field in the RPL with the SHOWCB or TESTCB macro to see whether VSE/VSAM has reached the end of the file.

When the exit receives control, it is in the same AMODE that was in effect when the request was issued.

When VSE/VSAM exits to the EODAD routine, the contents of the registers (**Reg**) are as follows:

Reg	Contents
0	Unpredictable.
1	Address of the request parameter list that defines the request that occasioned VSE/VSAM's reaching the end of the file. The register must contain this address if you return to VSE/VSAM.
2-13	Same as when the request macro was issued. Register 13, by convention, contains the address of your processing program's 72-byte save area, which may not be used as a save area by the EODAD routine if it returns control to VSE/VSAM.
14	Return address to VSE/VSAM.
15	Entry address to the EODAD routine.

If the EODAD exit routine returns to VSE/VSAM and you issue another GET macro, VSE/VSAM enters the EODAD exit routine again. This can cause your program to loop. If, however, you reach end-of-file during keyed access and then change to addressed access, additional records may be retrieved provided they are physically after the last record in key sequence (because of a CI or control-area split).

EXCPAD Exit Routine

An EXCPAD routine receives control from VSE/VSAM when an I/O operation is started. By supplying an EXCPAD exit routine, you can overlap VSE/VSAM I/O operations with the execution of your processing program.

The exit routine must return to VSE/VSAM, so that VSE/VSAM can return to your processing program at the instruction following the I/O request macro.

When the exit receives control, it is in the same AMODE that was in effect when the request was issued.

When VSE/VSAM exits to the EXCPAD routine, the contents of the registers (**Reg**) are as follows:

Reg	**Contents**
0	Unpredictable.
1	Address of a parameter list with the following contents: **Offset** **X'00'** Address of the RPL. **X'04'** Address of the IORB. **X'08'** EXCPAD lock word. **X'0C'** 116 bytes available to the user.
2-13	Same as when the request macro was issued. Register 13, by convention, contains the address of the user's 72-byte save area, which must not be used as a save area by the EXCPAD routine (because EXCPAD must return control to VSE/VSAM).
14	Return address to VSE/VSAM.
15	Address of EXCPAD routine.

If the exit routine uses Register 1, it must restore that register with the parameter-list address before returning to VSE/VSAM. (The routine must return for completion of the request that caused VSE/VSAM to exit.)

The EXCPAD routine may test the traffic bit of the IORB to determine whether the VSE/VSAM I/O operation has been completed. However, the routine must not change the contents of the IORB because the IORB is used by VSE/VSAM.

The EXCPAD lock word normally contains zero, in which case the routine may issue any other VSE/VSAM request with another RPL to the same file, except a CLOSE. When a CI split, control-area split or spanned record update occurs, the lock word contains the address of the RPL for the request. In that case, the EXCPAD routine must either complete the request because a second (simultaneous) request in the same file results in a system deadlock, or issue a request against another file. A split or spanned record update may occur during UPGRADE processing of an AIX. If this happens, no other UPGRADE request may be issued through *any* path to the same base cluster.

The EXCPAD exit routine may be entered more than once for a VSE/VSAM request because a request may require more than one I/O operation. The EXCPAD routine is not entered in the following cases:

- When the I/O operation completes before VSE/VSAM is ready to wait on it.
- During processing to complete pending I/O at CLOSE time.
- During Upgrade processing.

- When VSE/VSAM is forced to do a PUT because of insufficient buffers available (that is, when VSE/VSAM writes a buffer to be able to use this buffer for other data).

JRNAD Exit Routine to Journalize Transactions

A JRNAD routine records transactions made against a file and keeps track of RBA changes. VSE/VSAM exits to this routine every time the processing program issues a GET, PUT, or ERASE and every time data is shifted right or left in a CI or is moved to another CI (because one or more records have been inserted, deleted, shortened, or lengthened). VSE/VSAM takes the JRNAD exit before transmitting to direct-access storage the contents of a CI in which there was an RBA change. (You need to know whether RBAs have changed during keyed processing if later on you want to process your key-sequenced file with addressed access.)

VSE/VSAM also takes the JRNAD exit whenever a segment of a spanned record is transmitted to or from direct-access storage. This allows you to keep track of the CIs occupied by a spanned record.

The JRNAD exit must return to VSE/VSAM for completion of the request that caused VSE/VSAM to exit.

When the exit receives control, it is in the same AMODE that was in effect when the request was issued.

When VSE/VSAM exits to the JRNAD routine, the contents of the registers (**Reg**) are as follows:

Reg **Contents**

0 Unpredictable.

1 Address of a parameter list with the following format:

4 bytes
Address of RPL of the request that caused the exit.

4 bytes
Address of a field:
- 4 bytes: address of ACB in RPL of request that caused the JRNAD exit.
- 1 byte : reserved.

4 bytes
For RBA changes, the RBA of the first byte of data that is shifted or moved. For a GET or PUT request against a spanned record segment, the RBA of the first byte of the segment.

4 bytes
For RBA changes, the number of bytes of data that is shifted or moved. (The number of bytes does not include free space (if any), or control information - except for a control-area split, when the whole contents of a CI are moved to a new CI).

For a GET or PUT request against a spanned record segment, the number of bytes in the segment.

4 bytes
For RBA changes only, the RBA of the first byte to which data is shifted or moved.

1 byte Indication of the reason VSE/VSAM exited to the JRNAD routine:
X'00' GET request
X'04' PUT request

X'08' ERASE request
X'0C' RBA request
X'10' GET request against a spanned record segment
X'14' PUT request against a spanned record segment
X'18' Reserved
X'1C' Reserved

1 byte Reserved

2-13 Unpredictable,

14 Return address to VSE/VSAM.

15 Entry address to the JRNAD routine.

If, in your exit routine, you intend to issue the GENCB, MODCB, SHOWCB, or TESTCB macros, make sure that you save the contents of Register 14 before you issue the macro and restore these contents in Register 14 before your exit routine returns to VSE/VSAM. The same applies accordingly if, in your exit routine, you intend to use registers. Your exit routine must return to VSE/VSAM for completion of the request that caused VSE/VSAM to exit.

For journalizing transactions (when VSE/VSAM exits because of a GET, PUT, or ERASE), you can use the SHOWCB macro, for example, to display information in the RPL about the record that was retrieved, stored, or deleted by specifying:

```
FIELDS=(AREA,KEYLEN,RBA,RECLEN)
```

You can also use the TESTCB macro to determine whether a GET or a PUT was for update (OPTCD=UPD).

You cannot use the keywords RBA or RECLEN to display the RBA or length, respectively, of a spanned record segment retrieved or stored. Instead, this information is given in the parameter list at offsets 8 and 12, respectively.

For recording RBA changes, you must calculate how many records there are in the data shifted or moved, so you can keep track of the new RBA for every one. With fixed-length records, you calculate the number by dividing the record length into the number of bytes of data shifted. With variable-length records, you could 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 lengths.

Some CI splits cause data to be moved to two new CIs, and control-area splits normally cause the contents of many CIs to be moved. In these cases, VSE/VSAM exits to the JRNAD routine for every separate movement of data to a new CI.

If your JRNAD routine only journals transactions, it should ignore calls with the reason code X'0C' and return to VSE/VSAM; conversely, if it only records RBA changes, it should ignore all calls with reason codes other than X'0C'.

The only journaling you can do during processing of a path is to record transactions made against the base cluster; access to the alternate index during retrieval of a base record or during upgrading cannot be journaled. Journaling for path processing is triggered by the specification of the JRNAD exit in the EXLST of the ACB identifying the base cluster.

The JRNAD exit must be indicated as active before the file 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 ACB for a file and if you want to have a JRNAD routine, the first ACB you open for the file must specify the exit list that identifies the routine.

LERAD Exit Routine to Analyze Logic Errors

A LERAD routine analyzes logic errors and all other error conditions except I/O errors encountered by VSE/VSAM during execution of a GET, PUT, POINT, ENDREQ or ERASE macro. The routine determines what error has occurred by issuing a SHOWCB or TESTCB macro to examine the feedback (FDBK) field in the RPL. The contents of FDBK will be 0000xx, where xx is the error code indicating the type of error.

If the routine cannot correct the error, it should either:

- Close the file, or
- Return to VSE/VSAM (which will return to your processing program at the instruction following the last executed instruction).

If you do not have the LERAD exit routine and VSE/VSAM encounters a logic error, VSE/VSAM returns to your processing program at the instruction following the last executed instruction. Register 15 then contains X'08', and Register 1 contains the address of the RPL. Your program can examine the feedback field in the RPL with the SHOWCB or TESTCB macro to identify the logic error.

When the exit receives control, it is in the same AMODE that was in effect when the request was issued.

When VSE/VSAM exits to the LERAD routine, the contents of the registers are:

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 VSE/VSAM.
2-13	Same as when the request macro was issued. Register 13, by convention, contains the address of your processing program's 72-byte save area, which may not be used as a save area by the LERAD routine if the routine returns control to VSE/VSAM.
14	Return address to VSE/VSAM.
15	Entry address to the LERAD routine. The register does not contain the logical-error indicator.

SYNAD Exit Routine to Analyze Physical Errors

A SYNAD routine can analyze physical I/O errors that were detected by VSE/VSAM during execution of a GET, ENDREQ, PUT, POINT, ERASE, or CLOSE macro and that the system error routine was unable to correct. The exit routine determines what error has occurred (reading or writing either the data or the index component) by issuing a SHOWCB or TESTCB macro to examine the feedback (FDBK) field in the RPL. The contents of FDBK will be 0000xx, where xx is the error code indicating the type of error.

If your exit routine cannot correct or bypass the error, it is recommended that the routine (1) issues the PDUMP macro to obtain a dump of the contents of all pertinent control blocks, including the IORB involved in the failing I/O operation; (2) closes the files used by your program; and (3) ends the job. If the error occurred while VSE/VSAM was closing the file or index, and if another error occurs after the exit routine issues a CLOSE macro, VSE/VSAM does not exit to the routine a second time.

If the exit routine returns to VSE/VSAM, whether the error was corrected or the file closed, VSE/VSAM drops the request and returns to your processing program at the instruction following the last executed instruction.

If you do not have this exit routine and VSE/VSAM detects a physical error, VSE/VSAM returns to your processing program at the instruction following the last executed instruction. Register 15 then contains X'0C', and Register 1 contains the address of the RPL. Your program can examine the feedback field in the RPL with the SHOWCB or TESTCB macro to identify the physical error.

An I/O error that occurs when processing a data CI during the execution of a sequential GET request positions VSE/VSAM at the next CI in key sequence for keyed access or in entry sequence for addressed access. The next GET after the error will return the first record from the CI following the index CI, VSE/VSAM is not positioned at the next index control in further.

Errors that occur while VSE/VSAM writes a CI cause the loss of positioning.

When the exit receives control, it is in the same AMODE that was in effect when the request was issued.

When VSE/VSAM exits to the SYNAD routine, the contents of the registers are:

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 request macro; if you issued a CLOSE macro, the RPL was built by VSE/VSAM to process the close request. Register 1 must contain this address if the SYNAD routine returns to VSE/VSAM
2-13	Same as when the request macro or CLOSE macro was issued. Register 13, by convention, contains the address of your processing program's 72-byte save area, which may not be used by the SYNAD routine if it returns control to VSE/VSAM.
14	Return address to VSE/VSAM.
15	Entry address to the SYNAD routine. The register does not contain the physical-error indicator.

The GENCB Macro

The GENCB macro generates an ACB, an EXLST, or an RPL when it is executed. You can use it in place of the ACB, EXLST, and RPL macros to avoid (1) reassembling your programs should the format or length of the control block or list change, and (2) generating more than one copy of a control block or list.

VSE/VSAM returns, in Register 1, the address of the first (or only) control block and, in Register 0, the total length of the control block(s) built. You can find out the length of every control block by dividing the length of the area by the number of copies. The address of every control block can then be calculated by this offset from the address in Register 1.

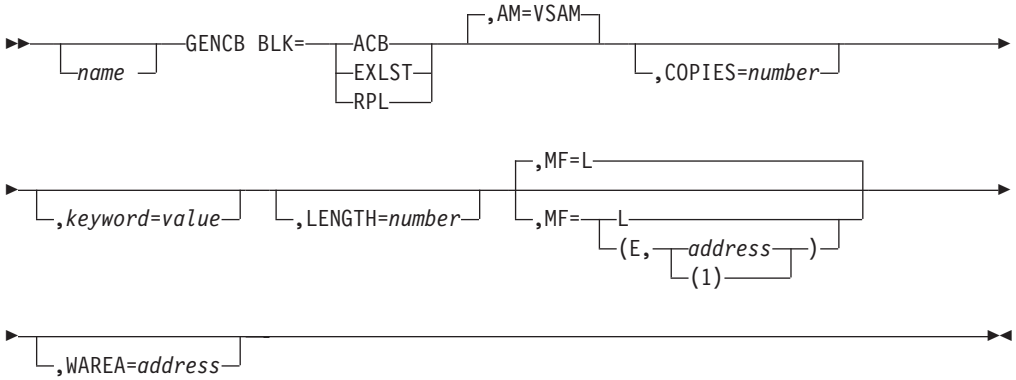
GENCB generates the control block(s) or list(s) either in an area you specify or, if you do not specify an area, in an area obtained by VSE/VSAM in your partition. The area obtained by VSE/VSAM can contain other control blocks too. It will not be freed at closing time but at end-of-job or end-of-job step only.

When you issue a GENCB macro, Register 13 must contain the address of a 72-byte save area that you are providing. When you issue a GENCB macro from within one of your exit routines (such as LERAD or SYNAD), your program must provide a second 72-byte save area for use by VSE/VSAM, because the original save area is still in use by the external VSE/VSAM routine.

The operands of the GENCB macro are specified as absolute numeric expressions, as character strings, as codes, as expressions that generate valid relocatable A-type address constants, in register notation, as S-type address constants, and as indirect S-type address constants. “Operand Notation for VSE/VSAM Macros” on page 295 gives all the ways of coding every operand for the macros that work at execution.

If you use register notation to specify specific addresses in your GENCB macro, be sure that these registers contain the correct addresses before you issue the GENCB macro. This is necessary because the assembler-generated instructions for this macro store the addresses contained in the specified registers in the appropriate control fields.

Format of the GENCB Macro



name
one through eight characters that provide a symbolic name.

AM=VSAM
specifies that this is a VSE/VSAM control block. You may want to specify this operand for documentation purposes if your installation also uses VTAM.

BLK=ACB|EXLST|RPL
specifies whether you want to generate an ACB, an EXLST, or an RPL.

COPIES=number
specifies the number of control blocks or lists you want VSE/VSAM to

generate. The default is 1. If you generate two or more, they are generated next to each other. They are identical, so you must use MODCB to tailor them for a particular file or request.

VSE/VSAM returns, in Register 1, the address of the first (or only) control block and, in Register 0, the total length of the control block(s) built. You can find out the length of every control block by dividing the length of the area by the number of copies. The address of every control block can then be calculated by this offset from the address in Register 1.

keyword=value
The operands you code are identical to those of the ACB, EXLST, and RPL macros, except that you can code them in more ways, as described in "Operand Notation for VSE/VSAM Macros" on page 295. If you do not code any operands, VSE/VSAM builds:

- For BLK=ACB, an ACB with default values provided by VSE/VSAM when you open the file. You must supply the DDNAME=filename operand before the file is opened.
- For BLK=EXLST, a complete EXLST with zeros for addresses and all entries flagged inactive.
- For BLK=RPL, an RPL with default values.

LENGTH=number
specifies the length of the area, if any, you provided by the WAREA operand. You can determine the length required for a control block or list by using the SHOWCB macro.

MF=
For information on specifying this operand, refer to "List, Execute, and Generate Forms of the Control Block Manipulation Macros" on page 291.

WAREA=address
specifies the address of an area in which you want VSE/VSAM to generate the control block(s) or list(s). The area must begin on a fullword boundary. If WAREA is specified, the LENGTH operand must also be specified. If you do not specify WAREA, VSE/VSAM obtains an area in your processing partition in which to generate the control block(s) or list(s). When control is returned to you, Register 1 contains the address of the control block or list and Register 0 contains the total length of the control block(s) or list(s).

Examples of the GENCB Macro

Figure 34 shows examples of how to specify VSE/VSAM control blocks by using the GENCB macro. With GENCB, the control blocks are created dynamically during execution of the program. The same parameters are specified in this example as are specified in the example of ACB, EXLST, and RPL macros shown in Figure 43 on page 265. VSE/VSAM obtains space for every control block in your partition. The address of every control block is set in Register 1 after the GENCB is executed.

```
*       GENERATE VSE/VSAM CONTROL BLOCKS
        GENCB  BLK=EXLST,EODAD=(ENDUP,N),
               LERAD=LOGERR,
               SYNAD=(IOERR,L)
        LTR    15,15                          GENCB successful?
        BNZ    GENERR                         No, go to error routine
        LR     3,1                            Yes, save EXLST address
        GENCB  BLK=ACB,EXLST=(3),PASSWD=PASS,
               BUFND=4,BUFNI=3,BUFSP=11064,
               MACRF=(KEY,SEQ,DIR,OUT),
               DDNAME=VFILENM
        LTR    15,15                          GENCB successful?
        BNZ    GENERR                         No, go to error routine
        LR     2,1                            Yes, save ACB address
*
*
        GENCB  BLK=RPL,AREA=WORK,
               AREALEN=125,OPTCD=(DIR,NSP),
               ARG=SEARCH,ACB=(2)
        LTR    15,15                          GENCB successful?
        BNZ    GENERR                         No, go to error routine
        LR     4,1                            Yes, save RPL address
*
*       PROCESSING ROUTINES
*
        GET    RPL=(4)
*
*       CONSTANTS AND WORK AREAS
*
PASS    DC     FL1'6',C'CHANGE'
WORK    DS     CL125
SEARCH  DS     CL4
```

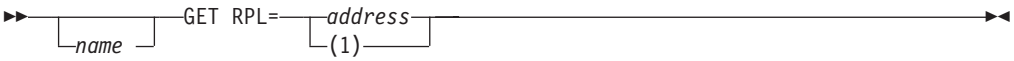
Note: The continuation characters required in column 72 are not shown.

Figure 34. GENCB Macro Examples

The GET Macro

When your program issues a request macro, its processing does not continue until VSE/VSAM completes the request. At that time, VSE/VSAM sets a return code in Register 15. If end-of-file is reached or an error or other special condition occurs during the request, VSE/VSAM sets a code containing additional information in the feedback (FDBK) field of the RPL and takes any required exit. The return codes and codes set in the feedback field of the RPL are described later in this section.

Format of the GET Macro



name
one through eight characters that provide a symbolic name.

RPL=address|(1)
specifies the address of the RPL (or the first RPL in a chain of RPLs) that defines the GET request.

You may specify the address in register notation (using a register from 1 through 12, enclosed in parentheses) or specify it with an expression that generates a valid relocatable A-type address constant.

When you issue a GET macro, Register 13 must contain the address of a 72-byte save area that you are providing. When you issue the macro from within one of your exit routines such as LERAD or SYNAD, you must provide a second 72-byte save area for use by VSE/VSAM.

This macro retrieves the next record in key sequence or the record with the next higher relative-record number with RPL operand OPTCD=(KEY,SEQ), and the next record in entry sequence with OPTCD=(ADR,SEQ). It retrieves the record specified by the key or relative record number in the search-argument field with OPTCD=(KEY,SKP) or OPTCD=(KEY,DIR), and by the RBA in the search-argument field with OPTCD=(ADR,DIR). With skip sequential retrieval, every key or relative-record number that you specify must be greater by number or alphabet than the key or relative-record number of the previous record retrieved.

GET retrieves the next CI with OPTCD=(CNV,SEQ) and the CI specified by the RBA in the search-argument field with OPTCD=(CNV,DIR).

You must issue a GET with OPTCD=UPD to update (PUT with OPTCD=UPD) or to delete (ERASE) a record. You can have the record moved to your work area (OPTCD=MVE) or you can have VSE/VSAM leave the record in its I/O buffer and pass you the address of the record (OPTCD=LOC). The AREA operand of the RPL macro points to your work area or to a field in which VSE/VSAM will place a record address.

You can also keep VSE/VSAM positioned for subsequent sequential or skip sequential processing when you issue a direct GET request with OPTCD=(DIR,NSP) or OPTCD=(DIR,UPD). With OPTCD=(DIR,UPD) however, positioning is canceled when you issue a PUT for update or an ERASE following the GET for update.

VSE/VSAM is Not Yet Positioned

If VSE/VSAM does not already have positioning for the RPL (or chain of RPLs) for which the GET request is to be issued, then you may have to issue an ENDREQ macro for a different RPL. An ENDREQ must be issued to free a position if the number of positions that VSE/VSAM must remember is already the same as the value specified in the STRNO=number operand of the pertinent ACB macro. At any particular time, VSE/VSAM will remember positions for any request macro in process by VSE/VSAM, and for a succeeding request for any RPL (or chain of RPLs) for which the preceding request was one of the following:

GET	DIR,LOC
GET	DIR,MVE,NSP
GET	DIR,MVE,UPD
GET	SEQ
GET	SKP
POINT	any
PUT	DIR,NSP
PUT	SEQ
PUT	SKP
ERASE	SEQ
ERASE	SKP

The MODCB Macro

The MODCB macro modifies the addresses, values, options, and names that you can establish with the ACB, EXLST, RPL, and GENCB macros in an ACB, EXLST, or RPL.

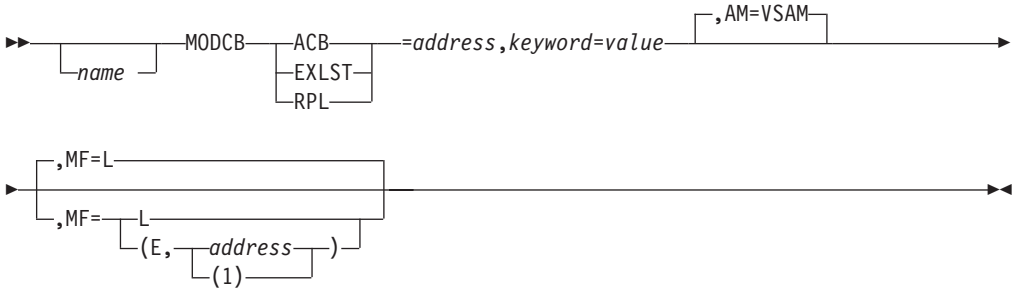
The operands of the MODCB macro are specified as absolute numeric expressions, as character strings, as codes, as expressions that generate relocatable A-type address constants, in ordinary z/VSE register notation, as S-type address constants, and as indirect S-type address constants. "Operand Notation for VSE/VSAM Macros" on page 295 gives all the ways of coding every operand for the macros that work at execution.

When you issue a standard MODCB macro (not the short form described below), Register 13 must contain the address of a 72-byte save area that you are providing. When you issue a MODCB macro from within one of your exit routines such as LERAD or SYNAD, your program must provide a second 72-byte save area for use by VSE/VSAM because the original save area is still in use by VSE/VSAM.

If you want to modify only the length of a data record (the value of the RECLEN field of the corresponding RPL), you can do so without any call to a VSE/VSAM routine by issuing the MODCB macro in the following short form: **MODCB RPL=(1),RECLEN=(0)**

The address of the RPL must be contained in Register 1 (short form only). The record length, stored in Register 0, will be placed into the RPL. No parameter list is created. For other MODCB functions, you must use the standard form of the MODCB macro.

Format of the MODCB Macro



name
one to eight characters that provide a symbolic name.

AM=VSAM
specifies that this is a VSE/VSAM control block. You may want to specify this operand for documentation purposes if your installation uses also VTAM.

ACB|EXLST|RPL=address
specifies whether you want to modify an ACB, an EXLST, or an RPL and specifies its address. Do *not* use the MODCB macro to:

- Modify an open ACB
- Activate or deactivate a JRNAD exit if the ACB to which the EXLST is connected is already open. (See the discussion of JRNAD in the EXLST macro.)

- Add entries to or delete entries from a field in an EXLST. (You can modify a field in an EXLST at any time.)
- Modify an active RPL, that is, one that defines a request that has been issued but not completed.

With the execute form of MODCB, you can change the address of the block or list to be modified, but not the type.

keyword=value
The operands you code are identical to those for the ACB, EXLST, and RPL macros, except that:

- You can code them in more ways, as shown in "Operand Notation for VSE/VSAM Macros" on page 295.
- There are no defaults for the options of the ACB MACRF operand or the RPL OPTCD operand. With OPTCD, when you set on a new option with the MODCB macro, the old option is automatically turned off, because you can specify only one option in every one of its groups (see "The RPL Macro" on page 231).
- You can make an address in an EXLST active or not active without specifying the address by coding: keyword=(,A|N).
- When you specify an address for an entry in an EXLST that previously contained zeros (possible if you generated a default list with the GENCB macro), you must code keyword=(addr,A) to make the address active, because A is not a default for the MODCB macro.

MF=
For information on specifying this operand, refer to "List, Execute, and Generate Forms of the Control Block Manipulation Macros" on page 291.

The MODCB macro cannot be used to reset a MACRF option which was set in an ACB unless this option is mutually exclusive with the new intended option. For example, if the options

```
KEY,SEQ,OUT
```

were set and you wish to have the options

```
ADR,SEQ,OUT
```

instead, then specifying MACRF=ADR in a MODCB macro results in options

```
KEY,ADR,SEQ,OUT
```

being set in the pertinent ACB.

Examples of the MODCB Macro

Figure 35 on page 228 shows two examples of modifying VSE/VSAM control blocks by using the MODCB macro.

The MODCB (short form) is used to place the length of a record in the RPL when variable-length records are added to a file:

```
MODCB  RPL=(1),RECLEN=(0)       Current length in register 0
LTR    15,15                    MODCB successful?
BNZ    MODERR                   No, go to error routine
PUT    RPL=(1)                  Yes, write record
```

The MODCB is used to activate the EODAD exit specified in the GENCB example of Figure 34 on page 224.

```
MODCB  EXLST=(3),EODAD=(,A)
LTR    15,15                    MODCB successful?
BNZ    MODERR                   No, go to error routine
```

Figure 35. MODCB Macro Examples

The OPEN Macro

The information you have specified in the ACB and EXLST macros must be connected with the file to be processed so that you can gain access to the data. To this purpose, you must supply, in the job stream, job control statements defining the file and issue, in your program, an OPEN macro for the ACB you have set up for the file.

The OPEN macro calls the Open routine, which verifies that the processing program has authority to process the file. The Open routine constructs VSE/VSAM control blocks and establishes linkages to those VSE/VSAM routines that are needed to process your file(s).

By examining the DLBL statement indicated by the DDNAME operand in the ACB macro and the volume information in the catalog, the Open routine verifies that the necessary volumes have been mounted. If a key-sequenced file is opened, VSE/VSAM issues an error code to warn you if the data has been updated separately from its index.

Format of the OPEN Macro

name
one through eight characters that provide a symbolic name.

address
specifies the address of the ACB or DTF for the file(s) to be opened. You can specify address:

- In register notation, using a register from 1 through 12. Specify within parentheses.

  Or
- With an expression that generates a valid relocatable A-type address constant.

  You can specify up to 16 addresses of ACBs and DTFs that define the files to be opened.

Return Codes from OPEN

A return code is set in Register 15 to indicate whether the ACBs were opened successfully. ACBs should be coded together to ensure that the return code will apply to all of them. If, for example, you coded:

```
OPEN  ACB1,DTF1,ACB2
```

the return code will apply to ACB2 only. If ACB2 opened successfully and ACB1 did not, the return code will still be X'00'. (The Open routine sets Register 15 to zero when it receives control after a DTF has been opened.) To ensure that the return code is valid and applies to both ACBs, the macro should be coded in the following way:

```
OPEN  DTF1,ACB1,ACB2
```

The Open routine sets one of the following return codes in Register 15:

Return Code	Meaning
X'00'	All ACBs were opened successfully.
X'04'	All ACBs opened successfully, but one or more ACBs had a warning message.
X'08'	One or more ACBs were not opened successfully. The entries with errors are restored to their pre-open status.

If Register 15 contains X'04', an error code is set in one or more ACBs to indicate a warning message. All ACBs are open and, unless you prevent it, processing will continue on the file that the message applies to. You can use the ERROR keyword of the SHOWCB or TESTCB macro to examine the code.

If Register 15 contains X'08', an error code is set in one or more ACBs. Again, you can use the ERROR keyword of the SHOWCB or TESTCB macro to examine the code. Note that Register 15 contains the maximum (worst) return code encountered while opening a list of ACBs. This means that some of the ACBs in the list may have been opened successfully, even though Register 15 contains X'04' or X'08'.

For an explanation of the VSE/VSAM OPEN error codes, see the "VSE/VSAM return and error codes" in the *z/VSE Messages and Codes*.

The POINT Macro

When you issue a POINT macro, Register 13 must contain the address of a 72-byte save area that you are providing. When you issue the macro from within one of your exit routines such as LERAD or SYNAD, you must provide a second 72-byte save area for use by VSE/VSAM.

When OPTCD=KEY was specified in the pertinent RPL, this macro positions VSE/VSAM at the record whose key or relative-record number you specify in the search argument field. You can use the macro to position VSE/VSAM for subsequent sequential or skip sequential processing, either forward or backward in the file.

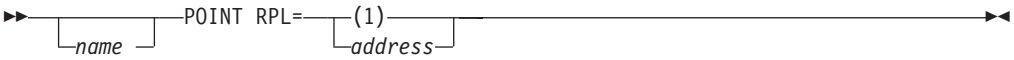
When OPTCD=ADR or OPTCD=CNV was specified in the pertinent RPL, the POINT macro positions VSE/VSAM at the record or CI whose RBA you specify in the search argument field. You can cause the macro to position VSE/VSAM for subsequent sequential processing, either forward or backward in the file.

Note: You cannot issue the POINT macro in one mode of access, change to another mode of access, and then request VSE/VSAM to continue processing the file sequentially. This will result in termination of the request with an error. For example, you cannot change from the mode OPTCD=ADR to another mode such as OPTCD=KEY.

VSE/VSAM can also be positioned for sequential processing by either a direct GET or a direct PUT as described in the preceding sections on the GET and PUT macros.

You may have to issue an ENDREQ macro before you can issue a POINT request in your program. Information about this possible requirement is given at the end of the discussion of the GET macro.

Format of the POINT Macro



name
one through eight characters that provide a symbolic name.

RPL=address|(1)
specifies the address of the RPL (or the first RPL in a chain of RPLs) that defines the POINT request. You can specify address:

- In register notation, using a register from 2 through 12. Specify within parentheses.

  Or
- With an expression that generates a valid relocatable A-type address constant.

The PUT Macro

When you issue a PUT macro, Register 13 must contain the address of a 72-byte save area that you are providing. When you issue the macro from within one of your exit routines such as LERAD or SYNAD, you must provide a second 72-byte save area for use by VSE/VSAM.

This macro stores a new record in key sequence or relative-record sequence if one of the following combinations of options is set in the RPL:

OPTCD=(KEY,DIR,NSP)
OPTCD=(KEY,SKP,NUP)
OPTCD=(KEY,DIR,NUP)
OPTCD=(KEY,SEQ,NUP)
OPTCD=(KEY,SEQ,NSP)

If you specify OPTCD=(KEY,DIR,NSP), VSE/VSAM is kept positioned at the next record in key sequence or relative-record sequence for subsequent sequential processing.

PUT stores a new record at the end of an entry-sequenced file with OTPCD=ADR. (You cannot store a new record in a key-sequenced file with addressed access.)

With skip sequential storage, OPTCD=(KEY,SKP), the key or relative-record number of every record that you store must be greater by number or alphabet than the key or relative-record number of the previous record stored.

With CI access, OPTCD=(CNV,NUP), PUT stores a new CI at the end of an entry sequenced file.

When loading or extending a file with the PUT macro, you must specify sequential or skip sequential processing (OPTCD=SEQ or OPTCD=SKP).

To store a changed record or CI, you must have previously retrieved it with option OPTCD=UPD set in the RPL for both the GET and the PUT. You cannot change the length of a record in either a relative record file or an entry-sequenced file.

The record to be added or updated with a PUT macro must be in your work area (OPTCD=MVE); you cannot use OPTCD=LOC with the PUT macro. The AREA operand of the RPL macro points to your work area.

You may have to write an ENDREQ macro before you can issue a PUT NUP or PUT NSP request in your program. Information about this possible requirement is given at the end of the discussion of the GET macro.

Format of the PUT Macro

name
one through eight characters that provide a symbolic name.

RPL=address|(1)
specifies the address of the RPL (or the first RPL in a chain of RPLs) that defines the PUT request. You can specify address:

- In register notation, using a register from 2 through 12. Specify within parentheses.

  Or
- With an expression that generates a valid relocatable A-type address constant.

The RPL Macro

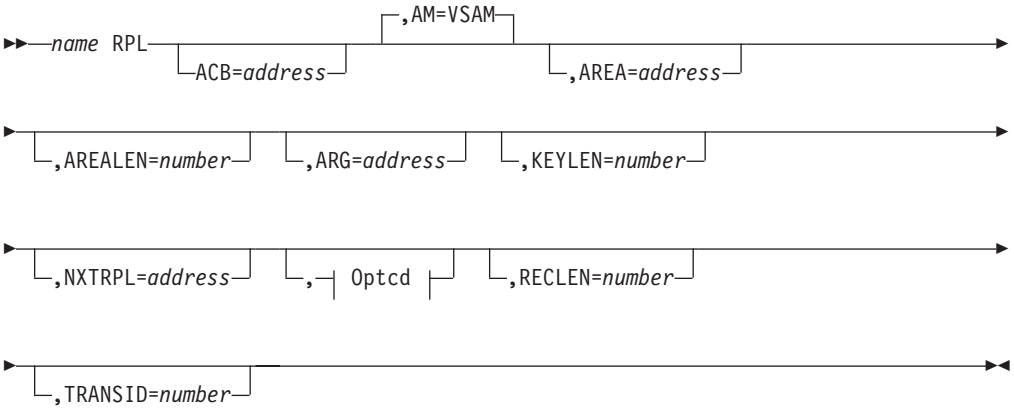
You define a request with the RPL macro, which produces a 'request parameter list' (RPL). Every request macro (GET, PUT, POINT, ERASE, and ENDREQ) has one and only one operand, the address of the request parameter list that defines the request. Thus, the information a request macro needs to access a record in a file (such as the ACB of the file to which the request is directed, or the search argument for the record) is always in the RPL instead of in the request macro itself.

The RPL does not indicate a specific request, such as GET or PUT, for example; you can use a single RPL, without modification, for several requests. However, if you want to use the same RPL for different types of processing (for both direct and sequential processing, for example), you must modify the RPL (with the MODCB macro) every time you change from one type of processing to another.

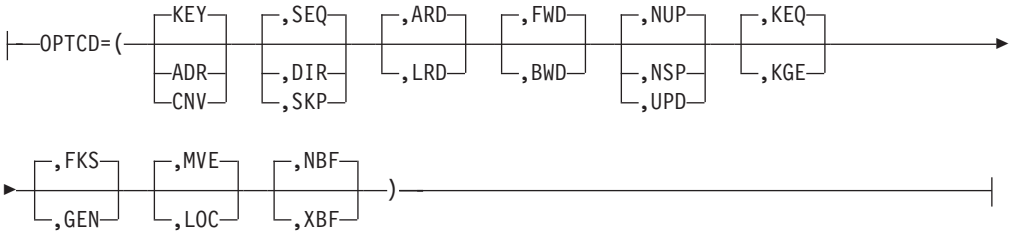
As was pointed out in the discussion of the STRNO operand of the ACB macro, several requests, with the corresponding RPLs pointing to the same ACB, can be active at the same time. You may specify any number of RPLs for requests requiring concurrent positioning, provided you do not exceed the maximum number of concurrent active requests you have specified in the STRNO operand. The requests can be for sequential or direct retrieval or both, and they can be for records in the same part of the file or in different parts.

Values for RPL macro operands can be specified as absolute numeric expressions, character strings, codes, and expressions that generate valid relocatable A-type address constants. Register notation cannot be used for *addresses*.

Format of the RPL Macro



Optcd:



name
one through eight characters that provide a symbolic address for the request parameter list that is generated. You can use it in the request macros to give the address of the list. You can also use it in the NXTRPL operand of the RPL macro, when you are chaining request parameter lists, to indicate the address of the next list.

ACB=address
specifies the address of the access method control block that identifies the file to which access will be requested. If you used the ACB macro to generate the control block, you can specify the label of that macro for the address. If you omit this operand you must issue a MODCB macro to specify the address of the file's ACB before you can issue a request against the RPL.

AM=VSAM
specifies that this is a VSE/VSAM control block. You may want to specify this operand for documentation purposes if your installation also uses VTAM.

AREA=address
specifies the address of your I/O work area to and from which VSE/VSAM moves the record (OPTCD=MVE) for GET and PUT requests. You process the record in this work area. If you process the records in VSE/VSAM's I/O buffer (OPTCD=LOC), VSE/VSAM puts into this work area the address of the record in the I/O buffer (GET only).

If you omit this operand you must issue a MODCB macro to specify the address of the request against the RPL.

When you specify user buffers (MACRF=UBF in the ACB) for CI (CNV) access, AREA specifies the address of a single I/O buffer. VSE/VSAM uses the buffer to read and write CIs.

AREALEN=number
specifies the length, in bytes, of the work area. For OPTCD=MVE, the work area must be large enough to contain the largest record in the file. For OPTCD=LOC, the work area must be at least 4 bytes long to contain the address of the record in the I/O buffer. For OPTCD=CNV, the work area must be at least the size of a CI.

If you omit this operand, you must issue a MODCB macro to specify the length of the request against the RPL.

ARG=address
specifies the address of a field that contains the search argument for:
- Direct or skip sequential retrieval (GET)
- Sequential positioning (POINT)
- Direct or skip sequential storage (PUT) for a relative-record file

For keyed access (OPTCD=KEY), the search argument may be a:
- Full key (OPTCD=FKS)
- Generic key (OPTCD=GEN) In this case you must also indicate its size in the KEYLEN operand.
- Relative-record number (which is treated as a key of 4 bytes length)

For addressed access (OPTCD=ADR), the search argument is always an RBA (relative byte address of a length of 4 bytes). To determine the RBA of a record to which you have gained access sequentially or directly by key, you can use the SHOWCB macro to display the RBA of the last record processed. (See "The SHOWCB Macro" on page 251).

Note: Addressed access is not available with an extended-addressed KSDS, and an RBA does not apply.
For CI access with user buffering and user-supplied RBA, the record is written only to this RBA if positioning is not established by a previous request.

When records are inserted (sequentially or directly) into a key-sequenced file, VSE/VSAM obtains the key from the record itself. When records are sequentially inserted into, or retrieved from, a relative-record file, VSE/VSAM returns the assigned relative-record number in the ARG field (as a four-byte binary number).

KEYLEN=number
When a generic key is used as a search argument (OPTCD=GEN), this operand specifies the length of the generic key in number of bytes. KEYLEN can be any value from 1 to 255.

If, for example, the full key is 50 bytes long and KEYLEN=10 is specified, VSE/VSAM uses the leftmost 10 bytes of the 50-byte key field for comparison with the search argument. The length of the full key is in the catalog. It can be obtained through the KEYLEN parameter of the SHOWCB macro. You place the key (full or generic) in a field pointed to by the ARG parameter.

NXTRPL=address
indicates the address of the next RPL in a chain of RPLs; it is required when you chain several RPLs together.

The standard request for access to a file retrieves, stores, or deletes a single record by means of one RPL specified in the request macro. If you want to retrieve or store more than one record with a single GET or PUT, you can chain several RPLs together so that every RPL indicates a different data record. For example, every RPL in the chain could contain a unique search argument and point to a unique work area. For a GET against such a chain of RPLs, VSE/VSAM retrieves a record for every RPL in the chain.

The positioning information, normally maintained for every RPL, is maintained only once for the total chain, so a chain of RPLs is processed as a single request. (Chaining RPLs is not the same as processing concurrent requests, where every request requires that VSE/VSAM keep track of a position in the file.) See the discussion of the STRNO operand under “The ACB Macro” on page 196.

Figure 36 on page 235 shows how to build a chain of RPLs by specifying the NXTRPL operand. When you issue a request that is defined by a chain of RPLs, specify in the request macro the address of the first RPL in the chain. This request macro determines the request type for the whole chain, and the same major operation, GET for example, is performed for all RPLs in the chain. However, other options such as the request options, which you specify in the OPTCD operand of the RPL macro, may vary from one RPL to another. Thus, an RPL with the option SEQ may be followed by an RPL with the option DIR.

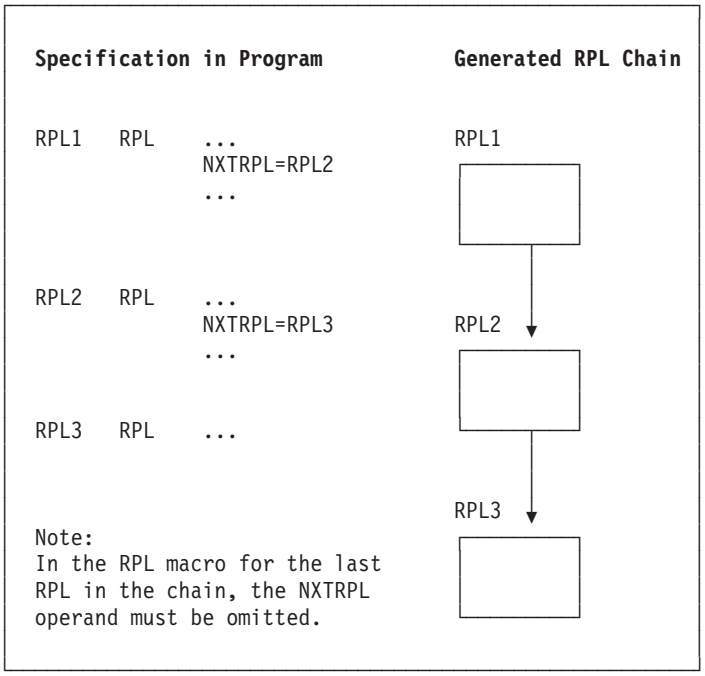
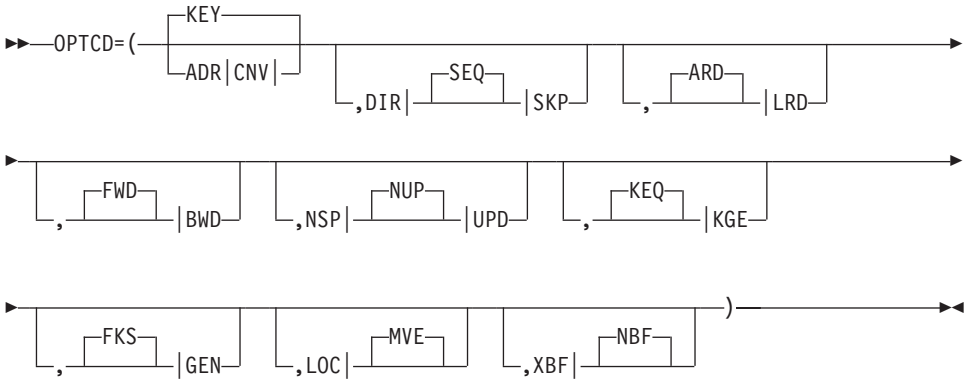


Figure 36. Example of an RPL Chain Built by Specifying the NXTRPL Operand

Except for the last RPL in a chain of RPLs, you cannot update or delete records, only retrieve records or store new records. Also you cannot process records in VSE/VSAM's I/O buffer with chained RPLs. So OPTCD=UPD and OPTCD=LOC are invalid for chained RPLs (except for the last RPL in a chain).

With chained RPLs, the following types of requests cause VSE/VSAM to position itself at the record following the one identified by the last RPL in the chain:

- POINT
- Sequential or skip sequential GET
- Direct GET with positioning requested (OPTCD=NSP)



specifies the type of access to be gained to the file through the requests defined by this RPL. Options are arranged in groups, and every group has a default value. You can specify only one option in every group; therefore, if your ACB indicates both sequential and direct processing for example, you must modify the RPL when you switch from one to the other. In other words you can use

the same RPL for different types of request (GET, PUT, POINT, for example) by modifying the RPL. Because VSE/VSAM ignores inapplicable option groups, there is no need for you to zero out options that are not required before you go from one type of request to another. For more information about modifying an RPL, see "The MODCB Macro" on page 226. The following list gives the options; they are arranged in groups, and every group has a default value (indicated by underlining):

KEY
Keyed access (for key-sequenced and relative-record files). For a key-sequenced file, you can change from keyed to addressed access without positioning. If you change from keyed to CI access, the results are unpredictable and no error code will be issued.

ADR
Addressed access (for key-sequenced and entry-sequenced files, not for relative-record files). If you change from addressed to keyed access, you must reestablish positioning or the request will terminate with an error. If you change from addressed to CI access, the results are unpredictable and no error code will be issued.

Note: Addressed access is not available for extended-addressed KSDS files. For more information, see "Extended-Addressed KSDS Files" in *VSE/VSAM Commands*.

CNV
CI access (provided for special applications such as utilities). If you change from CI to keyed access, you must reestablish positioning or the request will terminate with an error. If you change from CI to addressed access, the results are unpredictable and no error code will be issued.

Note: CI access is not available for extended-addressed KSDS files. For more information, see "Extended-Addressed KSDS Files" in *VSE/VSAM Commands*.

SEQ
Sequential processing.

DIR
Direct processing.

SKP
Skip sequential processing (for keyed access only).

FWD
Forward processing of a file.

BWD
Backward processing of a file (see "Specifying Processing Options for a Request" on page 238). Backward processing is only allowed for keyed (KEY) or addressed (ADR) access and for sequential (SEQ) or direct (DIR) processing.

ARD
The search argument given in your argument (ARG) field determines the record to be located, retrieved, or stored.

LRD
The last record of the file is to be located (POINT) or retrieved (GET direct). LRD can only be used in conjunction with OPTCD=BWD.

NUP
Request is not for update (you will not update or delete a record you are retrieving; a record you are storing is new). For a direct request, positioning will be released.

NSP
For direct processing only, request is not for update, and VSE/VSAM will be positioned at the next record for subsequent sequential processing.

UPD
Request is for update; you must issue a GET for update before you can issue a PUT for update or an ERASE. However, if you supply your own buffers for CI access, you can issue a PUT for update without a preceding GET.

KEQ
The search argument must equal the key of the data record (for keyed direct or skip sequential retrieval or keyed sequential pointing).

KGE
If the search argument does not equal the key of a record the request applies to the record with the next greater key (for keyed direct or skip sequential retrieval or keyed sequential pointing). If the search argument is a relative-record number, KEQ and KGE apply to a POINT request only. KGE is ignored if BWD is specified.

FKS
The entire key is to be used for a search argument (for keyed direct or skip sequential retrieval or keyed sequential pointing).

GEN
A generic key is to be used for a search argument (for keyed direct or skip sequential retrieval or keyed sequential pointing). You must specify the length of the generic key in the KEYLEN operand. GEN is ignored for relative-record files and if BWD is specified.

MVE
For retrieval and storage, VSE/VSAM moves a data record between the I/O buffer and your work area. MVE must also be specified when you supply your own buffers for CI access.

LOC
For retrieval, you can process the record in VSE/VSAM's I/O buffer. VSE/VSAM will pass you a pointer to the record in the buffer. If you want to update the record, you will have to move it to your work area before issuing a PUT macro (OPTCD=MVE). Do not specify LOC when processing spanned records.

NBF
Normal user buffering.

Each request as identified by an RPL is executed serially and independently. This is the conventional processing of user buffering and remains the default.

XBF
Extended user buffering.

VSE/VSAM will execute the chain of RPLs as a *single request*, thereby attempting to execute the requests with as few I/O requests as possible. All control intervals residing in the same control area will usually be

processed in a single I/O. For a description of extended user buffering see "How to Use Extended User Buffering: GET and PUT Macros" on page 287.

For more details on the options you can specify in the OPTCD operand of the RPL macro, refer to "Specifying Processing Options for a Request."

RECLEN=number
specifies the length, in bytes, of a data record stored by a PUT request. For fixed length records, the length need only be set once. For GET requests, VSE/VSAM indicates the length of the record in this field. To process a file with records of different lengths you can examine the field with the SHOWCB or TESTCB macro and modify it with the MODCB macro.

TRANSID=number
specifies a number that relates modified buffers in a buffer pool for a subsequent write operation (with the WRTBFR macro). It is used in shared resource applications and is described under "Sharing Resources Among Files and Displaying Catalog Information" on page 191.

Specifying Processing Options for a Request

The following deals mainly with keyed and addressed access as applied to the different types of processing (sequential, skip sequential, direct) and types of files. CI access and move/locate mode are described at the end of this section. See also "Examples of Request Macros" on page 267.

Keyed and Addressed Access

You can gain access to a record in:
- A KSDS file by keyed or addressed access.

  Note: Access to an extended-addressed KSDS file (> 4 GB) is keyed only. For more information, see "Extended-Addressed KSDS Files" in *VSE/VSAM Commands*.
- An RRDS file only by keyed access.
- An ESDS file only by addressed access.
- A VRDS file only by keyed access.

An alternate index or a path is treated like a KSDS, except that addressed access is not allowed for an alternate-index path. All key references in the RPL are to the alternate key (instead of the base cluster's prime key).

You can process spanned records in a KSDS by keyed (direct or sequential) access, and in an ESDS by addressed (direct or sequential) access. In either case, the entire record is returned. You cannot process spanned records in a KSDS file by addressed access, because the CIs that contain the spanned record may not be physically contiguous. You may process a file in backward direction by keyed or addressed access.

Figure 37 on page 239 summarizes the use of keyed and addressed access to retrieve, add (insert), update, or erase records in KSDS, ESDS, RRDS, and VRDS files. Sequential BWD means that the previous, instead of the next record in sequence (FWD) is to be accessed (see the BWD option of the OPTCD operand). Direct backward (BWD) is mainly used to prepare for a following GET sequential backward.

Type of File	Type of Access	Type of Processing	Records			
			Retrieve	Add	Update	Delete
key sequenced	keyed	sequential FWD sequential BWD skip sequential direct (FWD or BWD)	yes yes yes yes	yes no yes yes**	yes yes yes yes	yes yes yes yes
	addr***	sequential FWD sequential BWD direct (FWD or BWD)	yes yes yes	no no no	yes* yes* yes*	yes yes yes
entry sequenced	addr	sequential FWD sequential BWD direct (FWD or BWD)	yes yes yes	to end no to end**	yes* yes* yes*	no no no
relative record	keyed	sequential FWD sequential BWD skip sequential direct (FWD or BWD)	yes yes yes yes	yes no yes yes**	yes* yes* yes* yes*	yes yes yes yes
variable relative record	keyed	sequential FWD sequential BWD skip sequential direct (FWD or BWD)	yes yes yes yes	yes no yes yes**	yes yes yes yes	yes yes yes yes
* The length of the record cannot be changed. ** 'no' for backward (BWD) processing. *** Not available for extended-addressed KSDS files (> 4 GB)						

Figure 37. Summary of Processing Options for Keyed and Addressed Access

Sequential and Direct Processing

VSE/VSAM allows both sequential and direct processing for every of its types of files.

Sequential processing of a record depends on the position, with respect to the key, relative-record number, or address of the previously processed record; direct processing does not. With sequential access, records retrieved by key are in key sequence, records retrieved by relative-record number are in numerical order, and records retrieved by address are in entry (RBA) sequence. To retrieve or store records sequentially after initial positioning, you do not need to specify a key, relative-record number, or RBA. VSE/VSAM automatically retrieves or stores the next record in order. Apart from OPEN's positioning to the first record of a file, initial positioning can be established by:

- Pointing to the desired record, or
- Inserting a record into the file (keyed access with FWD only), or
- Using direct processing and:
  - Retrieving a record for update (UPD) or
  - Specifying OPTCD=NSP.

A variation of normal sequential retrieval is sequential backward processing. Instead of retrieving the next record in relation to current positioning in the file, the previous record is retrieved. Sequential backward processing is available for keyed and addressed access.

With direct processing, the retrieval or storage of a record is not dependent on the key, relative-record number, or address of any previously retrieved record. You must identify the record to be retrieved by key, or relative-record number, or RBA.

Keyed Access: Keyed access is for key-sequenced and relative-record files. The relative-record numbers of the records in a relative-record file are treated as keys. Keys or relative-record numbers are specified and returned in the area pointed to by the ARG operand of the RPL macro.

Keyed access provides for retrieval, update (including lengthening or shortening a record in a key-sequenced file, as well as altering its contents, except for the key), insertion, addition, and deletion. Each of these actions can be sequential, skip sequential, or direct.

With sequential processing, records are retrieved or stored in ascending key or relative-record sequence, starting from the beginning of the file or another position that you select. You do not have to supply a search argument for VSE/VSAM to process the records.

When you specify SEQ and BWD in the OPTCD operand of the RPL macro, VSE/VSAM returns the previous, instead of the next record in the file (in relation to current positioning). The previous record is the one which has the next lower key (or relative-record number). With the SEQ and BWD options, you can retrieve, update, or erase records, but you cannot insert or add records.

With direct processing, records are retrieved by the search argument (key or relative-record number) you supply. Records can be processed in any order, without regard to the sequence of records processed before or after.

With skip sequential processing, records are retrieved by search argument, but in *ascending* key or relative-record sequence (no backward processing). Thus, skip sequential combines functions of both sequential and direct processing.

The subject is discussed below in more detail for keyed retrieval, storage, and deletion.

Sequential (SEQ) Retrieval: If you specify KEY and SEQ for a *key-sequenced file*, the record to be retrieved depends on where VSE/VSAM is positioned in the file. When your program opens the file, VSE/VSAM is positioned at the first record in the file to begin sequential processing. However, if sequential processing is not to begin with the first record of the file, you can issue a POINT macro to position VSE/VSAM at the record whose key you specify. (If the specified key is generic, that is, a leading portion of the key field, then VSE/VSAM is positioned to the first of the records that have the same generic key.) A subsequent GET macro retrieves the record VSE/VSAM is positioned at and, at the same time, positions VSE/VSAM at the record with the next higher key. In the POINT macro you can also indicate the direction in which the file is to be processed subsequently, by specifying either FWD or BWD.

When you are accessing a base cluster through 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 several records contain the same (non-unique) alternate key, these records are retrieved in the order in which they were entered into the alternate index (even if BWD was specified). In addition, although Register 15 contains X'00', a warning code (duplicate key) is set in the FDBK field of the RPL if there is at least one more data record with the same alternate key value. For

example, if there are three data records with the alternate key "1234", the error code would be set during the retrieval of records one and two, and would be reset during retrieval of the third record.

Besides the error code, a function code is set in the RPL indicating whether the condition occurred during accessing the alternate index or the base cluster of a path or during upgrade processing (for a description of the function code, see "Return Codes of Request Macros" on page 289).

If a base cluster is accessed in a partition, once using a path and once not using a path, "a no record found" or "duplicate key error" can occur. These errors can be avoided by using Local Shared Resources (LSR).

The example in Figure 38 illustrates backward sequential retrieval through a path with non-unique alternate keys.

Alternate Index	Pointer	Record
CI 1: Alternate Key 10	1	T
	2	U
	3	E
CI 2: Alternate Key 20	1	S
	2	D
	3	A
	4	Y
Backward sequential retrieval result in the sequence: S, D, A, Y, T, U, E		

Figure 38. Example of Backward Sequential Retrieval through a Path with Non-Unique Alternate Keys

Keyed sequential retrieval for a *relative-record file* causes the records to be returned in ascending or, if you are retrieving the previous record, descending numerical order, based on the positioning for the file. Positioning is established in the same way as for a key-sequenced file, the relative-record number always treated as a full 4-byte key. If one or more empty slots are encountered during sequential retrieval, they are skipped and the next (or previous) record is retrieved. The relative-record number of the retrieved record is returned in the ARG field of the RPL.

Sequential Backward (SEQ BWD) Retrieval: To process a file in backward direction or to switch from forward to backward processing or vice versa, you must position VSE/VSAM and, at the same time, indicate the direction of subsequent processing. Open always establishes forward processing direction so that a GET sequential backward immediately after Open results in a positioning error.

To position VSE/VSAM to the end of the file, issue a POINT macro with OPTCD=(BWD,LRD) specified in the RPL. A subsequent GET sequential backward retrieves the last record of the file. To locate and retrieve any other record in the file and establish backward processing direction at the same time, issue a POINT with OPTCD=(BWD,ARD) and a subsequent GET sequential backward (or a direct GET with OPTCD=(BWD,NSP)).

A read error during a GET with:

```
OPTCD=(SEQ,BWD)
```

does not cause the positioning to be lost. An immediately following GET with OPTCD=(SEQ,BWD) will cause VSE/VSAM to skip the next logical record in backward direction that can be retrieved without a read error.

Direct (DIR) Retrieval: Keyed direct retrieval for a *key-sequenced file* does not depend on previous positioning; VSE/VSAM searches the index from the highest level down to the sequence set to retrieve a record. You must specify the record to be retrieved by supplying, in the ARG field of the RPL, one of the following:

- The exact key of the record (OPTCD=KEQ)
- A key less than or equal to the key field of the record (OPTCD=KGE)
- A leading portion of the key, or generic key (OPTCD=GEN)

You can specify OPTCD=KGE when you do not know the exact key. If a record actually has the specified key, VSE/VSAM retrieves it; otherwise, it retrieves the record with the next higher key. Generic-key specification for direct processing causes VSE/VSAM to retrieve the first record with a key whose leading portion is identical with the key in the ARG field. If you want to retrieve all the records with the generic key, specify NSP for your direct request, which causes VSE/VSAM to position itself at the next record in key sequence. You can then retrieve the remaining records with the same generic key sequentially.

If you use *generic keys* in conjunction with *direct* requests there is an additional aspect to consider. VSE/VSAM has to read a data CI to determine that it is empty. So the performance of direct requests with a generic key will decrease if you have many deleted records that match your generic key and precede the first existing record.

To retrieve a record in the file and indicate backward processing direction for a subsequent GET sequential backward, issue a direct GET with OPTCD=(BWD,NSP,ARD), or LRD instead of ARD if you want to retrieve the last record in the file. The search argument must always be a full key (FKS) and must be the same as that of the data record (KEQ); KGE and GEN are ignored. A direct GET or a POINT with OPTCD=(BWD,LRD) against an empty file results in a no-record-found condition.

When you are accessing a base cluster through a *path* with direct access, a record from the base cluster is returned according to the alternate key value you have specified in the ARG field of the RPL macro. If the alternate key is not unique, the record which was first entered with that alternate key is returned and a warning code (duplicate key) is set in the FDBK field of the RPL. To retrieve the remaining records with the same alternate key, specify the NSP option when retrieving the first record and then change to sequential processing.

If a base cluster is accessed in a partition, once using a path and once not using a path, a “no record found” or “duplicate key” error can occur. These errors can be avoided by using Local Shared Resources (LSR).

When you are processing a *relative-record file* with direct access, you must supply the 4-byte relative record number of the desired record in the ARG field of the RPL macro. If you request a deleted or non-existent record, the request will result in a no-record-found condition.

Skip Sequential (SKP) Retrieval: For skip sequential retrieval for a *key-sequenced file*, when you indicate the key of the next record to be retrieved, VSE/VSAM skips to its index entry by using horizontal pointers in the sequence set to get to the appropriate sequence-set index record to scan its entries. SKP is similar to direct processing, except that the key of the next record must always be higher in sequence than the key of the preceding record.

A *relative-record file* has no index. When you indicate the number of the next record to be retrieved, VSE/VSAM calculates the CI containing the requested record and the position of the requested record within that CI. As for a key-sequenced file, the relative-record numbers you specify must be ascending sequence for skip sequential retrieval.

For a *path*, skip sequential access is the same as direct access, except that the alternate key values have to be in ascending sequence. If a base cluster is accessed in a partition, once using a path and once not using a path, a "no record found" or "duplicate key" error can occur. These errors can be avoided by using Local Shared Resources (LSR).

Backward processing is not allowed for skip sequential retrieval.

Keyed Insertion: VSE/VSAM stores a record whenever you issue a PUT request against an RPL. A PUT request for update following a GET for update stores the record that the GET retrieved. To update a record, you must previously have retrieved it for update.

When you store records sequentially beyond the highest key in the file, VSE/VSAM automatically extends the file as though you were continuing to load records. VSE/VSAM does not use distributed free space for these records, but establishes new control areas at the end of the file. Free space is left in the new control areas and CIs according to the file's FREESPACE specification in the catalog.

To store records in key (or relative-record) sequence throughout the file, you can use sequential, skip sequential, or direct access.

When you insert records into a key-sequenced file, you never have to specify a search argument; VSE/VSAM always obtains the key from the record itself. With sequential insertion or skip sequential insertion of consecutive records, VSE/VSAM creates new CIs and control areas and free space is left in them according to the file's FREESPACE specification in the catalog. With direct insertion or skip sequential insertion of non-consecutive records, VSE/VSAM uses the free space.

For a *relative-record file*, sequential insertion causes a record to be inserted into the next slot (provided it is empty). The slot number is returned in the ARG field of the RPL. If the slot is not empty, a duplicate-record error condition will occur.

Direct or skip sequential insertion of a record into a relative-record file causes the record to be placed as specified by the relative-record number in the ARG field. You must insert the record into a slot which does not contain a record; otherwise, a duplicate-record error condition will occur.

If you insert a record after the current end-of-file of a relative-record file, the file is preformatted from the current end-of-file up to and including the control area that is to contain the inserted record. Preformatting mainly consists of inserting control information in the control areas and indicating that the slots are empty.

You can update and insert base data records via a *path*, provided the PUT request does not result in non-unique alternate-key values in an alternate index (in the upgrade set) which you have defined with the UNIQUEKEY parameter. The alternate indexes in the upgrade set are modified automatically when you insert or update a data record in the base cluster. When you update a previously retrieved base record via a path, you must not change the alternate key by which that record was retrieved or its prime key. If the updating of the alternate index results in an alternate index record with no pointers to the base cluster, that alternate index record is erased.

PUT insert requests with OPTCD=NUP or NSP are not allowed in backward direction.

Keyed Deletion: An ERASE macro instruction following a GET for update deletes the record that the GET retrieved. A record is physically erased in the file when you delete it. 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 file. The alternate indexes of the upgrade set are modified automatically when you erase a record. If the alternate key value of the erased record is unique, the alternate index data record with that alternate key is also deleted.

You can erase a record from a *relative-record file* after you have retrieved it for update. The record will be set to binary zeros and the control information for the slot will be updated to indicate an empty slot. You can reuse the vacated space by inserting another record of the same length in that location.

Addressed Access: Addressed access is the only form of access for an entry-sequenced file, using the RBA determined for a record when it was stored in the file. This form of access is also allowed for a key-sequenced file, but not for a path or for a relative-record file. For both key-sequenced and entry-sequenced files, addressed access allows processing in backward direction (by specifying OPTCD=BWD in the RPL macro). Positioning is established as for keyed retrieval. You cannot add or insert records in backward direction.

Addressed access can be either sequential or direct for both key-sequenced and entry-sequenced files, but the processing allowed for a key-sequenced file is different from that allowed for an entry-sequenced file.

With a key-sequenced file, addressed access can be used to retrieve records, update their contents, and delete records, but the length of a record and the contents of its key field cannot be changed. Records cannot be added because VSE/VSAM does not allow changes to the file which could cause the index to change. With an entry-sequenced file, addressed access can be used to retrieve records and to update their contents, but not to change their lengths. New records can be added to the end of the file. Records cannot be physically deleted because that would change the entry sequence of the records in the file (the RBAs of the records).

Keyed insertion, deletion, or update (length changing) of records can change the RBAs of these records. Therefore, to use addressed access to process a key-sequenced file, you may have to keep track of RBA changes. For this purpose VSE/VSAM passes back the RBA of every record retrieved, added, updated, or deleted. (See also “JRNAD Exit Routine to Journalize Transactions” on page 218.)

Note: Addressed access is not available for extended-addressed KSDS files (> 4 GB). For more information, see "Extended-Addressed KSDS Files" in *VSE/VSAM Commands*.

Addressed Retrieval: Positioning for addressed sequential retrieval is done by RBA rather than by key. When a processing program opens a file for addressed access, VSE/VSAM is positioned at the first record in the file in entry sequence to begin addressed sequential processing. A POINT positions VSE/VSAM for sequential access beginning at the record whose RBA you have indicated. A sequential GET causes VSE/VSAM to retrieve the data record at which it is positioned and positions VSE/VSAM at the next or previous record in entry sequence depending on whether you have specified forward (FWD) or backward (BWD) processing in the RPL. If you use addressed sequential retrieval for a key-sequenced file, records will not be in their key sequence if there have been CI or control-area splits.

Addressed direct retrieval requires that the RBA of every individual record be specified, because previous positioning is not applicable. The address specified for a GET or a POINT must correspond to the beginning of a data record; otherwise, the request is invalid.

With direct processing, you may optionally specify that GET position VSE/VSAM at the next record in forward (FWD,NSP) or backward (BWD,NSP) sequence. Your program can then process the following or preceding records sequentially.

Addressed Deletion: You can use the ERASE macro with a key-sequenced file to delete a record that you have previously retrieved for update.

With an entry-sequenced file, you are responsible for marking a record you want to delete. In other words, as far as VSE/VSAM is concerned, the record is not deleted. You can reuse the space occupied by a record marked for deletion by retrieving the record for update and storing in its place a new record of the same length.

Addressed Insertion: VSE/VSAM does not insert new records into the middle of an entry-sequenced file, but adds them at the end. With addressed access of a key-sequenced file, VSE/VSAM does not insert or add new records. You cannot add or insert new records in backward direction.

When you store records sequentially beyond the highest key in the file, VSE/VSAM automatically extends the file as though you were continuing to load records.

A PUT macro instruction stores a record. A PUT for update following a GET for update stores the record that the GET retrieved. To update a record, you must previously have retrieved it for update. You can update the contents of a record with addressed access, but you cannot alter the record's length. Neither can you alter the key field of a record in a key-sequenced file. To change the length of a record in an entry-sequenced file, you must store it either at the end of the file (as a new record) or in the place of a deleted record of the same length (as an update). You are responsible for marking the old version of the record as deleted.

CI Access

VSE/VSAM provides programmers of utilities and systems with CI access. They retrieve and store the contents of a CI, rather than a single record, by specifying CI access in the macros and (for direct processing) giving the RBA of the CI. They are

responsible for maintaining the control information at the end of the CI. The format of this information may change in future releases of VSE/VSAM.

CI access is allowed for relative-record files, provided the size of the file is not changed by insertions or additions. CI access is not allowed when you process an alternate-index path or access records in backward direction (with the BWD option).

Note: CI access is not available for extended-addressed KSDS files. For more information, see "Extended-Addressed KSDS Files" in *VSE/VSAM Commands*.

Processing a Record in a Work Area or in a Buffer: When your processing program retrieves a record, VSE/VSAM reads into virtual storage the contents of the entire CI in which the record is stored. VSE/VSAM de-blocks the records and either places the requested record in your program's work area (OPTCD=MVE) or leaves the record in VSE/VSAM's I/O buffer and gives you, in the AREA field, the address of the record in the buffer (OPTCD=LOC). VSE/VSAM indicates the length of the record to your program (in the RECLEN field) in both move mode and locate mode. You need not concern yourself with any physical attributes of stored records. Spanned records cannot be accessed in locate mode.

The SHOWCAT Macro

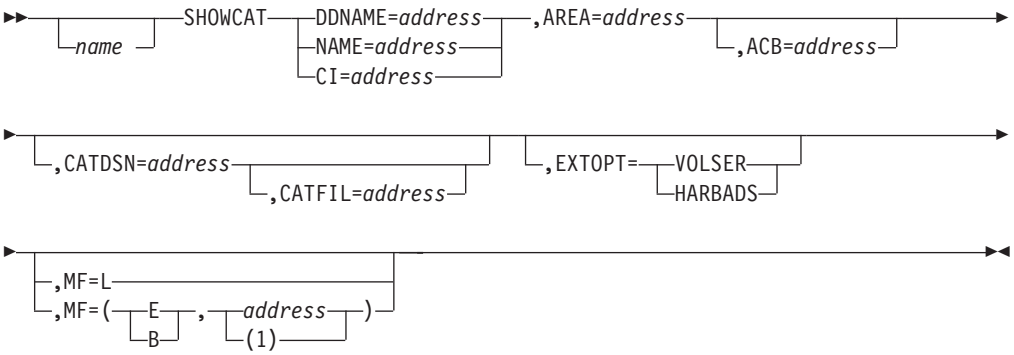
With the SHOWCAT macro, you can retrieve information from a catalog about any non-open file defined in the catalog.

For explanations on the *relationship* of the information that you can retrieve, refer to "Displaying Catalog Information" on page 189 (Figure 33 on page 190).

Format of the SHOWCAT Macro

When you issue a SHOWCAT macro, Register 13 must contain the address of a 72-byte save area that you are providing. When you issue a SHOWCAT macro from within one of your exit routines, such as LERAD or SYNAD, your program must provide a second 72-byte save area for use by VSE/VSAM, because the original save area is still in use by the external VSE/VSAM routine.

The SHOWCAT macro has the following format:



name
one to eight characters that provide a symbolic name.

DDNAME|NAME|CI=address
specifies the address of an area that identifies the catalog entry that contains the desired information.

DDNAME=address
specifies the address of a seven-byte area containing the file name of the object to be displayed. The object can be a cluster (C), an alternate index (G), or a path (R). Using the indicated file name, SHOWCAT first retrieves the corresponding name (file ID) of the object from the label cylinder and then the desired information from the catalog.

Either this parameter or the NAME or CI parameter must be provided. However, when issuing the first SHOWCAT for an object, specify DDNAME or NAME. VSE/VSAM then supplies the CI numbers of any associated objects for subsequent SHOWCATs (in the work area supplied through the AREA operand). See Figure 39 on page 250.

NAME=address
specifies the address of a 44-byte area containing the name (file ID) of the object to be displayed. The name is left-justified and padded with blanks on the right. The type of object named must be C, G, R, D, or I.

Either this parameter or the DDNAME or CI parameter must be specified. However, when issuing the first SHOWCAT for an object, specify DDNAME or NAME. VSE/VSAM then supplies the CI numbers of any associated objects for subsequent SHOWCATs (in the work area supplied through the AREA operand). See Figure 39 on page 250.

CI=address
specifies the address of a three-byte area that contains the CI number of the catalog entry for the object to be displayed. The entry type of the object must be C, G, R, D, I, or Y. (Y can only be retrieved via CI).

Either this parameter or the DDNAME or NAME parameter must be specified. However, when you have already issued a SHOWCAT request for an object (with the DDNAME or NAME parameter), you then issue any subsequent SHOWCATs for its associated objects by specifying their CI numbers (as returned to you via the previous SHOWCAT DDNAME or NAME request).

The three-byte area must be separate from the work area specified by the AREA operand, even though VSE/VSAM returns a CI number in the work area.

AREA=address
specifies the address of a work area in which the catalog information is to be displayed. The first two bytes of this area must contain the length of the area, including these two length bytes.

The minimum size of the area is 64 bytes, unless EXTOPT is specified. With EXTOPT, the minimum size is 28 bytes. If it is smaller than the minimum size, you get a return code of 4 in Register 15 and you can reissue the SHOWCAT macro with a larger size. The format of the work area is described in Figure 39 on page 250.

ACB=address
specifies the address of the ACB that defines the catalog containing the entry to be displayed. You issue the first SHOWCAT without ACB specified; VSE/VSAM searches for the specified objects and returns to you (in the work area supplied through the AREA operand) the address of the ACB that defines the correct catalog. The catalogs are searched in the following order: the catalog specified by the CATDSN parameter, the catalog specified by the CAT

parameter of the VSE/VSAM file, the job catalog, or if none of these exist, the master catalog. When you subsequently issue SHOWCAT, you can specify that ACB address, which causes VSE/VSAM to go directly to the correct catalog without searching other catalogs first. You should always include the ACB parameter when you specify CI instead of NAME.

CATDSN=address and CATFIL=address
CATDSN specifies the address of a 44-byte area containing the name (file ID) of the catalog to be searched.

CATFIL specifies the address of an 8-byte area containing the file name of the catalog to be searched. File ID and file name must be the same as those specified in the DLBL statement (if one is provided) for the catalog. CATDSN must always be specified if CATFIL is specified. CATFIL is always optional. You use these parameters to override the established order in which catalogs are searched. (VSE/VSAM always searches only one catalog for a specific entry.) That is, you must specify CATDSN if the object to be displayed is (1) not specified by the CAT parameter on the DLBL statement for the file, (2) not in the job or master catalog, or (3) in the master catalog and not the job catalog (if IJSYSUC and IJSYSCT are both specified).

EXTOPT=VOLSER|HARBADS
indicates that either the volume serial number of the file's primary allocation volume (VOLSER) or the high allocated RBA for the file (HARBADS) is to be returned to you. This operand can only be issued for a D or I type catalog record.

The data returned for the EXTOPT operand replaces the associated object information in the user return area. If you need the associated object information as well as the EXTOPT data, you must issue separate SHOWCAT macros.

MF=L
specifies that the list form of the SHOWCAT macro is required. The list form builds a parameter list when the macro is assembled; it is not executable. AREA and DDNAME|CI|NAME are optional in the list form; if you do not specify them in the list form they must be specified in the execute form. In the list form, the operand addresses cannot be expressed in register notation. The format of the SHOWCAT parameter list is described in "Parameter Lists for VSE/VSAM Macros" on page 303.

MF=(E|B,address|(1))
specifies that the execute form of the SHOWCAT macro is required.

E indicates that the parameter list, whose address is given in address or in a register, is to be passed to VSE/VSAM for processing.

B indicates that the parameter list is to be built or modified, but is not to be passed to VSE/VSAM. This form of the macro is similar to the list form, except that it works at execution time and can modify a parameter list, as well as build it.

To build a parameter list, first issue SHOWCAT with only MF=(B) specified, to zero out the area in which it will be built.

address gives the address of the parameter list. If you use register notation, you may use Register 1, as well as a register from 2 through 12. Register 1 is used to pass the parameter list to VSE/VSAM (if MF=E).

If the MF operand is omitted, the standard form of the SHOWCAT macro is assumed, which builds the parameter list, puts its address in Register 1, and passes control to VSE/VSAM to process the list.

Return Codes from SHOWCAT

When VSE/VSAM returns to your processing program after a SHOWCAT request, Register 15 contains one of the following return codes:

Return Code	Meaning
0 (0)	VSE/VSAM completed the request.
4 (4)	The area specified in the AREA operand is less than the minimum required (64 bytes) or is too small to display all associated objects (as many objects as possible are displayed).
8 (8)	Either the ACB address is invalid or the VSE/VSAM master catalog does not exist or could not be opened.
12 (0C)	The request was not executed because an error was found while VSE/VSAM routines were loaded (see Note).
20 (14)	The named object or CI does not exist (see Note).
24 (18)	An I/O error occurred in gaining access to the catalog (see Note).
28 (1C)	The specified CI number is invalid.
32 (20)	The specified object is not a C, D, G, I, R, or Y type (see Note).
36 (24)	The information in the catalog is at a different level than that in the CRA.
40 (28)	An unexpected error code was returned from catalog management to the SHOWCAT processor (see Note).
44 (2C)	An error occurred in searching the label area for the file ID corresponding to the specified file name (See Note).
48 (30)	EXTOPT field name is not valid for SHOWCAT.
52 (34)	EXTOPT specified, but record type not D or I.

If a return code of 0 was passed in Register 15, the requested catalog information is returned in the work area which you have supplied through the AREA operand. The format is shown in Figure 39 on page 250.

Note: In case the SHOWCAT return code in Register 15 is 12, 20, 24, 28, 36, 40, 44, or 52, the work area contains the return code and reason code issued by VSE/VSAM catalog management as well as the module ID of the catalog management module in which the error was detected. The format of the work area is then as follows:

Offset	Length	Description
0	2	Length of work area
2	2	VSE/VSAM catalog return code[1] or (for return code 44) VSE/VSAM error code
4	2	VSE/VSAM catalog reason code[1]
6	2	VSE/VSAM catalog management module ID

[1] For the codes, see the "VSE/VSAM return and error codes" in the *z/VSE Messages and Codes*.

Offset Length
Description

0(0) 2 Length of the work area, including the length of this field (provided by you).

2(2) 2 Length of the work area actually used by VSE/VSAM, including the length of this field and the preceding field.

4(4) 4 The address of the ACB that defined the catalog that contains the entry which is to be displayed.

8(8) 1 Type of object about which information is returned:

C Cluster
D Data component
G Alternate index
I Index
R Path
Y Upgrade set

The following fields contain one set of information for C, G, R, and Y types, and another set for D and I types.

For C, G, R, and Y types:

9(9) 1 For Y type: Reserved. For C type:

x... .... The SHOWCAT output for the D type record will provide the VSAM file type (1).

.xxx xxxx
Reserved

For G types:

x... .... The alternate index might (1) or might not (0) be a member of an upgrade set. The way to find out for sure is to display information for the upgrade set of the base cluster and check whether it contains CI numbers of entries that describe the components of the alternate index. Figure 33 on page 190 shows you how to get from the alternate index's catalog entry to the entries that describe its components (G to C to D to Y to D and I).

.xxx xxxx
Reserved.

For R type:

x... .... The path is (1) or is not (0) defined with the UPDATE attribute (for upgrading alternate indexes).

.xxx xxxx
Reserved.

10(A) 2 The number of pairs of fields that follow. Every pair of fields identifies another catalog entry that describes an object associated with this C, G, R or Y object. The possible types of associated objects are:

With C: D, G, I, R.
With G: C, D, I, R.
With R: C, D, G, I.
With Y: D, I.

Figure 33 on page 190 shows how the catalog entries for all these objects are related.

12(C) 1 Type of associated object the entry describes.

13(D) 3 The CI number of its first record.

16(10) Next pair of fields, and so on. If the area is too small to display a pair of fields for every associated object, VSE/VSAM displays as many pairs as possible and returns a code of 4 in Register 15.

Every pair of fields occupies 4 bytes, except Y-type entries which require 8 bytes (4 for the data component and 4 for the index component of the alternate index in the upgrade set).

Figure 39. Format of the SHOWCAT Work Area (Part 1 of 2)

For D and I types:

9(9) 1 For I type: Reserved. For D type:
..xx .... .1x. -> The file is a SAM file (NOCIFORMAT data set, DTFPH access only). .01. -> The file is a SAM ESDS file (ACB access allowed). .00. -> The file is a native VSAM file, defined as:
.x00 x... 0000 -> ESDS (Entry-Sequenced Data Set) 1000 -> KSDS (Key-Sequenced Data Set) 0001 -> RRDS (Relative Record Data Set) 1001 -> VRDS (Variable-length Relative Record Data Set)

For SAM ESDS (invalid for native VSE/VSAM file):
x... .... File definition by implicit (1) or explicit (0) DEFINE
.... .x.. The SAM record format is blocked (1)
.... ..x. The SAM record format is variable records (1)
....x The SAM record format is fixed records (1)

10(A) 2 Relative position of the prime key in records in the data component. For the data component of an entry-sequenced or a relative record file there is no prime key, and this field is 0.

12(C) 2 Length of the prime key, or length of logical record for fixed-blocked SAM files.

14(E) 4 CI size of the data or index component.

18(12) 4
Maximum record size of the data or index component, or block size for blocked SAM files.

22(16) 2
The number of pairs for fields that follow. Every pair of fields identifies another catalog entry that describes an object associated with this D or I object. The possible types of associated objects are:

With D: C, G, Y.
With I : C, G.

Figure 33 on page 190 shows how the catalog entries for all these objects are related.

24(18) 1
Type of associated object the entry describes.

25(19) 3
The CI number of its first record.

28(1C) Next pair of fields, and so on. Fields for all associated objects can always be displayed (with the minimum AREA size specified).

Figure 39. Format of the SHOWCAT Work Area (Part 2 of 2)

The SHOWCB Macro

The SHOWCB macro displays fields in an ACB, EXLST or RPL. VSE/VSAM places these fields in an area that you provide. They are independent of the format of the control block or list you are displaying. The fields are displayed in the order that you specify the keywords for them.

The operands of the SHOWCB macro are specified as absolute numeric expressions, as character strings, as codes, as expressions that generate valid relocatable A-type address constants, in ordinary z/VSE register notation, as S-type address constants, and as indirect S-type address constants. “Operand Notation for VSE/VSAM Macros” on page 295 gives all the ways of coding every operand for the macros that work at execution.

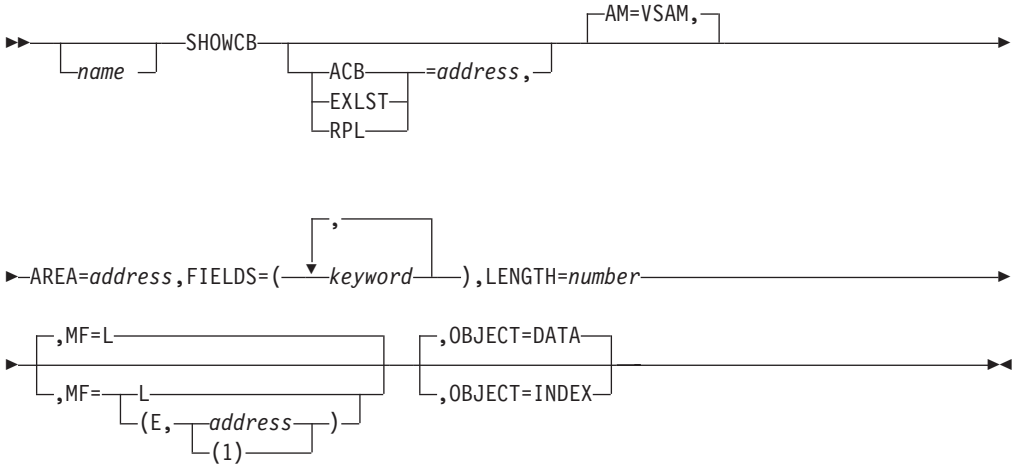
When you issue a standard SHOWCB macro (not the short form described below), Register 13 must contain the address of a 72-byte save area that you are providing. When you issue a SHOWCB macro from within one of your exit routines such as LERAD or SYNAD, your program must provide a second 72-byte save area for use by VSE/VSAM because the original save area is still in use by the external VSE/VSAM routine.

If you want to display only the length of a data record (the RECLEN field of the corresponding RPL), you can do so without any call to a VSE/VSAM routine by issuing the SHOWCB macro in the following short form:

SHOWCB RPL=(1),RECLEN=(0)

The address of the RPL must be contained in Register 1. The record length will be put into Register 0. No parameter list is created. For other SHOWCB functions, you must use the standard form of the SHOWCB macro.

Format of the SHOWCB Macro



name
one to eight characters that provide a symbolic name.

ACB|EXLST|RPL=address
This operand specifies whether you want to display an ACB, an EXLST, or an RPL and specifies its address.

In the standard and list forms of SHOWCB, you can omit this operand if you are displaying only the standard length of a control block or list (see "Length of a Control Block or List" on page 254). With the execute form of SHOWCB, you can change the address of the block or list to be displayed, but not the type.

AM=VSAM
specifies that this is a VSE/VSAM control block. You may want to specify this operand for documentation purposes if your installation also uses VTAM.

AREA=address
specifies the address of the area in virtual storage that you are providing for VSE/VSAM to display the items you specify in the FIELDS operand. The items are in the area in the order you specify the keywords. The area must begin on a fullword boundary.

FIELDS=(keywords)
There are three groups of keywords you can code for the FIELDS operand of the ACB.
- The keywords that you can code with the ACB, EXLST, RPL, and GENCB macros.
- The length of an ACB, RPL, or EXLST.

- The attributes of an open file or index indicated by the ACB.

For details, refer to:
- "Keywords of the ACB, EXLST, and RPL Macros"
- "Length of a Control Block or List" on page 254
- "Attributes of an Open File" on page 254

LENGTH=number
specifies the length of the display area you are providing (by way of the AREA operand). Every field in the ACB and RPL takes a fullword, except for DDNAME and STMST in the ACB, which take two fullwords. Every EXLST operand takes only one fullword, because you cannot display the codes A, N, and L.

MF=
For information on specifying this operand, refer to "List, Execute, and Generate Forms of the Control Block Manipulation Macros" on page 291.

OBJECT=DATA|INDEX
specifies, for the open ACB of a key-sequenced file, whether the fields displayed are for the data or the index. VSE/VSAM will display the same values for KEYLEN regardless of your specification in the OBJECT operand. The same is true for field RKP.

If you specify INDEX, VSE/VSAM's display is all zeros for the following fields:

FS	NINSR	NUPDR
NCIS	NRETR	
NDELR	NSSS	

Keywords of the ACB, EXLST, and RPL Macros

The keywords in this group require one fullword every for display, except DDNAME which requires two fullwords. The keywords are identical to those of the ACB, EXLST, and RPL macros, except that:

- You can code the operands in more ways, as shown in "Operand Notation for VSE/VSAM Macros" on page 295.
- You do not code the address, value, option, or name to which the keyword is equal.
- In relation to the ACB macro, you cannot display the MACRF options and the RMODE31 keyword.

  With the keyword **ERROR**, you can display the error code (in the rightmost byte of the display word) from the Open or Close routine (see the OPEN and CLOSE macros); you can test the **MACRF** options with the TESTCB macro.

  Also in relation to the ACB macro, you cannot display the ABEND CLOSE disposition, that is, the second KEEP or DELETE keyword of the PARMS= parameter.
- In relation to the EXLST macro, you cannot display the codes that indicate whether an exit address is active or not active or is the address of the name of a routine to be loaded; you can test them with the TESTCB macro.
- In relation to the RPL macro, you cannot display the OPTCD options, but you can code the keyword **FDBK** to display error codes (in the rightmost byte of the display word) from the request macros and the keyword **RBA** to display the relative byte address of the last record processed; you can test the **OPTCD** options with the TESTCB macro.

You can code the keyword **AIXPC** to display the number of key or RBA pointers in the most recently processed alternate index record.

You can code the keyword **FTNCD** to display, after a logical or physical error, the *function code* which indicates whether the respective condition occurred during processing of the base cluster or the alternate index of a path or during upgrade processing. (For details, see "Return Codes of Request Macros" on page 289.)

Length of a Control Block or List

You can code the keyword **ACBLEN**, **EXLLEN**, or **RPLLEN** to display either the standard length of an ACB, EXLST, or RPL, or the actual length of a particular block or list. You display a standard length by omitting the ACB|EXLST|RPL operand and coding only one (or more) of these length keywords and no other keywords. You display the actual length of a block or list by specifying the ACB|EXLST|RPL operand and the corresponding length keyword.

Attributes of an Open File

After a file is opened, the ACB contains information that it does not contain before it is opened or after it is closed. Whether you are displaying the attributes of the data or the index of a key-sequenced file is determined by the OBJECT operand. Every item displayed requires one fullword in your work area, except STMST which requires two fullwords. You can display the following items:

Attribute
: **Meaning**

AVSPAC
: Number of bytes of available space in the data or index component.

BFRFND
: Number of requests for retrieval that could be satisfied without an I/O operation; that is, the data was found in the buffer. Applies for LSR only.

BUFNO
: Number of buffers used for the data or index component.

BUFRDS
: Number of requests for retrieval that require I/O operation; that is, the data was *not* found in the buffer. Applies for LSR only.

CINV Size of a CI in the data or index component.

ENDRBA
: Ending RBA of the space used by the data component or the index component.

FS Percent of free CIs in every data CA of a key-sequenced file.

HALCRBA
: High allocated RBA. The relative byte address (1 fullword) of the end of the data component (OBJECT=DATA) or the index component (OBJECT=INDEX) of the cluster opened by the related ACB.

KEYLEN
: Full length of the prime key or alternate key field in every logical record (depending on whether or not you access the base cluster via a path).

LRECL
: Maximum length of a logical record, or for an index, the index CI size minus seven bytes.

NCIS Number of CI splits in the file.

NDELR
Number of data records deleted from the file.

NEXCP
Number of times EXCP was issued by VSE/VSAM I/O routines.

NEXT Number of logical extents, data spaces, or portions of data spaces, now allocated to the data or index component.

NINSR
Number of data records inserted into the file. For a relative-record file, number of valid records (non-empty slots in the file). For a key-sequenced file, number of records inserted between the records, not records initially loaded or added to the end of the file.

NIXL Number of levels in the index of a key-sequenced file.

NLOGR
Number of data records in the file. For a relative-record file, total number of slots (empty or non-empty) in the used CIs.

NRETR
Number of data records retrieved from the file.

NSSS Number of data control-area splits in a key-sequenced file.

NUIW Number of write requests that VSE/VSAM was forced to do because buffers were not available for reading the contents of a control interval (CI). (NUIW is the number of write requests that were *not* initiated by the user.) Applies for LSR with DFR only.

NUPDR
Number of data records updated in the file.

RKP Displacement of the prime key or alternate key field from the beginning of a data record (depending on whether or not you access the base cluster via a path); the same value is displayed whether the object is index or data.

STMST
System time stamp; the time and day (in microseconds) when the data or index component was last closed. Bits 52 through 63 of the field are unused.

STRMAX
Maximum number of requests which were concurrently active since the resource pool was built. Used in shared resource applications (see "The BLDVRP Macro" on page 207).

UIW Number of all other write requests (those that are not counted in NUIW). Applies for LSR only.

Example: The SHOWCB Macro

Figure 40 on page 256 is an example of how to display information from VSE/VSAM control blocks using the SHOWCB macro. Continuation characters required in column 72 are not shown in the example.

The SHOWCB macro is used to display statistics about an open file:

```
         SHOWCB ACB=(2),AREA=DISPLAY,LENGTH=12,                        *
                FIELDS=(KEYLEN,LRECL,RKP)
         LTR    15,15                        SHOWCB successful?
         BNZ    SHOWERR                      No, go to error routine
         .
         .
DISPLAY  DS     0F                           Align on fullword boundary
KEYLEN   DS     F
LRECL    DS     F
RKP      DS     F
```

The SHOWCB macro is used to display the length and RBA of a record that has been retrieved:

```
         GET    RPL=(4)
         LTR    15,15
         GNZ    GETRR
         SHOWCB RPL=(4),AREA=DISPLAY,LENGTH=8,                         *
                FIELDS=(RECLEN,RBA)
         LTR    15,15                        SHOWCB successful?
         BNZ    SHOWERR                      No, go to error routine
         .
         .
DISPLAY  DS     0F                           Align on fullword boundary
RECLEN   DS     F
RBA      DS     F
```

Figure 40. SHOWCB Macro Example

Example: Statistics on Use of LSR Buffer Pools

This example shows what to specify in SHOWCB to obtain statistics about the usage of buffer pools for local shared resources (LSR). The information can help you to determine how to improve both, a previous definition of a resource pool, and the mix of data sets that use a pool.

The statistics:

- Are available through an ACB that describes an open data set that uses a buffer pool.
- Reflect the use of the buffer subpool from the time it was built up to the time you issue SHOWCB.
- Are for a single buffer subpool. To get statistics for all buffer subpools, issue a SHOWCB for each of the subpools.

The example specifications for displaying the statistics are:

```
SHOWCB ACB=(R6),AREA=SHOW,FIELDS=(BFRFND,BUFRDS,NUIW,UIW),  *
       LENGTH=16,OBJECT=INDEX
```

Figure 41. SHOWCB Macro Example

where:

- R6 must point to an ACB for an open data set.
- SHOW must be 16 bytes long. After processing of SHOWCB, the field SHOW will contain all four counters (each being four bytes long).
- INDEX specifies that the statistics are to be taken from the LSR sub-pool that is used by the index component of the data set.

The TCLOSE Macro

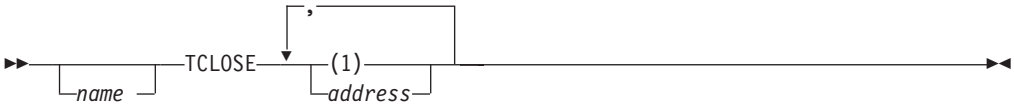
A TCLOSE macro completes outstanding I/O operations and updates the catalog. Processing can continue without reopening the file. You use the TCLOSE macro to protect data while the file is loaded or extended and the SPEED option was specified when the file was defined. When TCLOSE is issued, the close routine formats the last CA in the file to ensure that all of the data that has been loaded is accessible.

The TCLOSE macro cannot be used to change the processing mode for a file from sequential load to retrieve in the same run.

The TCLOSE macro has no effect when the local shared resources (LSR) option is in the ACB macro together with DFR (deferred write).

The return codes and error codes are identical to those of the CLOSE macro.

Format of the TCLOSE Macro



name
one through eight characters that provide a symbolic name.

address
specifies up to 16 addresses of ACBs. You can specify address:

- In register notation, using a register from 1 through 12. Specify within parentheses.

  Or
- With an expression that generates a valid relocatable A-type address constant.

You cannot specify the address of DTFs with TCLOSE.

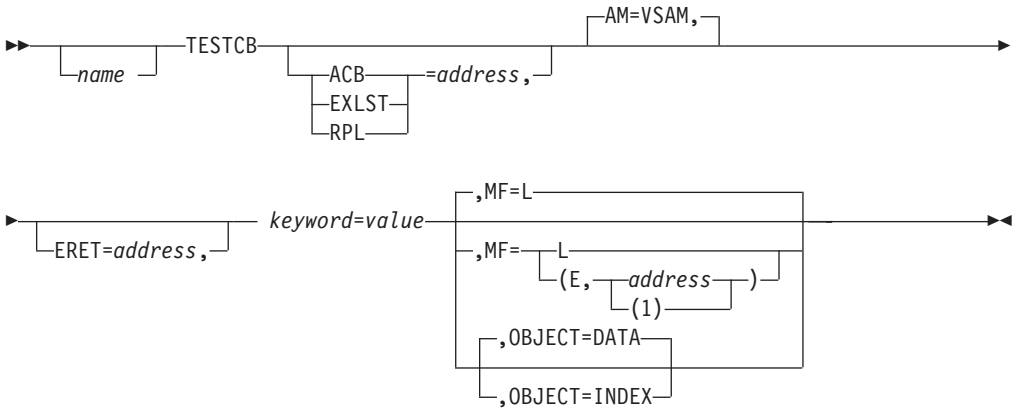
The TESTCB Macro

The TESTCB macro tests values in an ACB, EXLST, or RPL against values that you specify in the macro.

You can examine the condition code after issuing a TESTCB macro and examining the return code in Register 15. For keywords specified as an option (such as A for an operand of the EXLST macro), a test is for an equal or unequal comparison; for keywords specified as an address or value, a test is for an equal, unequal, high, low, not-high, or not-low comparison. In the comparison, A to B, B is the address, value, or option that you specify in the TESTCB macro. For example, if you test for a value in an ACB, a high comparison means the value in the block is higher than the value you specified in the TESTCB macro.

When you issue a TESTCB macro, Register 13 must contain the address of a 72-byte save area that you are providing. When you issue a TESTCB macro from within one of your exit routines such as LERAD or SYNAD, your program must provide a second 72-byte save area for use by VSE/VSAM because the original save area is still in use by the external VSE/VSAM routine.

Format of the TESTCB Macro



name
one to eight characters that provide a symbolic name.

ACB|EXLST|RPL=address
This operand specifies whether you want to test an ACB, an EXLST, or an RPL and specifies its address.

In the standard and list forms of TESTCB, you can omit this operand if you are testing only the standard length of a control block or list (see "Length of a Control Block or List" on page 259). With the execute form of TESTCB, you can change the address of the block or list to be tested, but not the type.

AM=VSAM
specifies that this is a VSE/VSAM control block. You may want to specify this operand for documentation purposes if your installation also uses VTAM.

ERET=address
specifies the address of a user-written routine that VSE/VSAM gives control if, because of an error, it is unable to test for the condition you specified (return code in Register 15 is not X'00'). When the ERET routine receives control, it should inspect the return code. If the return code is X'04', an error code will be tested in Register 0. See "Return Codes from the Control Block Manipulation Macros" on page 291 for the error codes that can be tested by TESTCB.

After completing its processing, the ERET routine can terminate the job or pass control to a point in the processing program that it determines. It cannot return to VSE/VSAM.

keyword=value
specifies a field and a value. The contents of the field are compared with the value and the condition code is set. You can specify only one keyword at a time. There are THREE groups of operands that you can code with the TESTCB macro:

- The addresses, values, options, and names that you can code with the ACB, EXLST, RPL, and GENCB macros
- The length of a control block or list
- The attributes of an open file or index indicated by the ACB.

If you code more than one operand, every one of them must compare equal to the corresponding value in the block or list for you to get an equal condition.

For details, refer to:

- "Operands of the ACB, EXLST, and RPL Macros"
- "Length of a Control Block or List"
- "Attributes of an Open File or Index" on page 260

MF=
For information on specifying this operand, refer to "List, Execute, and Generate Forms of the Control Block Manipulation Macros" on page 291.

OBJECT=DATA|INDEX
specifies, for the open ACB of a key-sequenced file, whether the field tested is for the data or the index. KEYLEN and RKP will contain the same value, no matter whether the data or the index is tested. FS, NCIS, NDELR, NINSR, NIXL, NLOGR, NRETR, NSSS, and NUPDR will contain zeros if the index is tested.

Operands of the ACB, EXLST, and RPL Macros

The operands in this group are identical to those of the ACB, EXLST, and RPL macros.

- You can code the operands in more ways, as shown "Operand Notation for VSE/VSAM Macros" on page 295.
- In relation to the ACB macro, you can test for error codes from the Open and Close routines by coding ERROR=*code* (as any absolute expression, except for a self-defining character term). When an ACB is opened for a path, the base cluster ACB is tested. However, you can test the alternate index ACB by specifying MACRF=AIX in the ACB macro. For the ACB, you cannot test the RMODE31.
- In relation to the EXLST macro, you can test whether an EXLST has an exit of a certain type by coding keyword=0.
- In relation to the EXLST macro, you can test whether an address in an EXLST is active or not active or is the address of the name of a routine to be loaded by coding: keyword=,A|N or keyword=,A|N,L.
- In relation to the RPL macro, you can code the operand **FDBK=code** (as any absolute expression, except for a self-defining character term) to test for error codes from the request macros (see "Return Codes of Request Macros" on page 289). You can code the operand RBA=number to test the relative byte address of the last record processed.
- In relation to the RPL macro, you can code the operand **AIXPC=number** to find out the number of key or RBA pointers in the most recently processed alternate index record.
- You can code the operand **AIXFLAG=AIXPKP** to test whether the alternate index record just processed contains prime key pointers (or, if not, RBA pointers).
- You can code the operand **FTNCD=number** to test (after a logical or physical error) the *function code*. The function code indicates whether the respective condition occurred during processing of the base cluster or the alternate index of a path or during upgrade processing. (For details, see "Return Codes of Request Macros" on page 289.)

Length of a Control Block or List

You can code the operand EXLLEN=length, ACBLEN=length, or RPLLEN=length to test either the standard length of an EXLST, ACB, or RPL; or the actual length of a particular ACB, RPL, or EXLST. You test for a standard length by omitting the ACB|EXLST|RPL operand and coding only one (or more) of these length operands

and no other operands. You can test the actual length of a control block or list by specifying the ACB|EXLST|RPL operand and the corresponding length operand.

Attributes of an Open File or Index

After a file is opened, the ACB contains information that it does not contain before it is opened or after it is closed. Whether you are testing for the attributes of the data or the index of a key-sequenced file is determined by the OBJECT operand. By coding **OFLAGS=OPEN**, you can test whether the file is open.

You can test the following fields:

Attribute
Meaning

AVSPAC
Number of bytes of available space in the data or index component

BUFNO
Number of buffers used for the data or index component

CINV Size of a CI in the data or index component

ENDRBA
Ending RBA of the space used by the data component or the index component.

FS Percent of free CIs in every data CA of a key-sequenced file

KEYLEN
Full length of the prime key or alternate key field in every logical record (depending on whether or not you access the base cluster via a path)

LRECL
Maximum length of a logical record or, for an index, the index CI size minus seven bytes

NCIS Number of CI splits in the file

NDELR
Number of data records deleted from the file

NEXCP
Number of EXCP commands issued since the data or the index was opened

NEXT Number of logical extents, data spaces or portions of data spaces, now allocated to the data or index component

NINSR
Number of records inserted into the file. For a relative-record file, number of valid records, that is, non-empty slots in the file.

NIXL Number of levels in the index of a key-sequenced file

NLOGR
Number of data records in the file. For a relative-record file, total number of slots (empty or non-empty) in the used CIs.

NRETR
Number of data records retrieved from the file

NSSS Number of control-area splits in a key-sequenced file

NUPDR
Number of data records updated in the file

RKP Displacement of the prime key or alternate key field from the beginning of a data record (depending on whether or not you access the base cluster via a path); the same value is displayed whether the object is index or data.

STMST
System time stamp; the time and day (in microseconds) when the data or index component was last closed. Bits 52 through 63 of the fields are unused.

You can also test for these attributes:

Specification
Meaning
ATRB=COMP
File is defined with the COMPRESSED attribute
ATRB=ESDS
Entry-sequenced file
ATRB=KSDS
Key-sequenced file
ATRB=RRDS
Relative-record file
ATRB=VRDS
Variable-length Relative record Data Sets
ATRB=WCK
VSE/VSAM is verifying write operations
ATRB=SSWD
Sequence set of the index is adjacent to the file
ATRB=REPL
Index records are replicated
ATRB=SPAN
File contains spanned records
ATRB=UNQ
Unique alternate keys in alternate index
ATRB=XLKSDS
Extended-addressed KSDS

Furthermore, you can determine whether the opened object is a path, a base cluster, or an alternate index by coding:
OPENOBJ=PATH
Alternate index/base cluster pair (path)
OPENOBJ=BASE
Base cluster
OPENOBJ=AIX
Alternate index

Examples of the TESTCB Macro

Figure 42 on page 262 shows examples of how the TESTCB macro can be used to test values in a VSE/VSAM control block.

Continuation characters required in column 72 are not shown in the example.

Example 1: Uses TESTCB to determine whether or not a file is open.

```
            TESTCB ACB=(2),OFLAGS=OPEN,
                   ERET=TESTERR
            BE     OPEN
            B      UNOPEN
            .
            .
TESTERR     .....
```

Example 1: Uses TESTCB to determine whether the LERAD exit routine was entered because of an end-of-file condition or a processing error. (The example assumes that no EODAD exit routine was provided.)

```
LOGERR      TESTCB RPL=(4),FDBK=4,
                   ERET=TESTERR
            BE     EODATA
            B      ERROR
            .
            .
TESTERR     ......
```

Figure 42. TESTCB Macro Examples

The WRTBFR Macro

Managing I/O Buffers

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.
- Writing out buffers whose writing has been deferred.

Deferring Write Requests

VSE/VSAM normally writes out the contents of a buffer immediately for direct PUT requests. With shared resources, however, you can cause write operations for direct PUT requests to be deferred. Buffers are finally written out:

- When you issue the WRTBFR macro.
- When VSE/VSAM needs a buffer to satisfy a GET request.
- When a file using a buffer pool is closed. (Temporary CLOSE is ineffective against a file that is sharing buffers; nor does ENDREQ cause buffers in a resource pool to be written.)

Deferring writes saves I/O operations in cases where subsequent requests can be satisfied by the data in the buffer pool. Processing speed improves if CIs are updated several times.

You indicate that writes are to be deferred by coding the MACRF DFR option in the ACB, along with MACRF=LSR:

```
ACB  MACRF=(LSR,DFR,...)
```

NDF, the default, indicates that writes are not to be deferred for direct PUTs.

The DFR option is incompatible with SHAREOPTIONS(4). (SHAREOPTIONS is a parameter of the IDCAMS command DEFINE.) A request to open a file with SHAREOPTIONS(4) for deferred writes is rejected.

Relating Deferred Requests by Transaction ID

You can relate action requests (GET, PUT, etc.) 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 of the modified buffers used for this transaction to be written out together. When the WRTBFR request is complete, the transaction is physically complete. To relate requests, specify:

```
RPL TRANSID=number
```

TRANSID=number

specifies a number from 0 to 31. A number from 1 to 31 relates the request(s) defined by this RPL to the requests defined by other RPLs with the same transaction ID. The number 0, which is the default, indicates that the request defined by this RPL is not associated with other requests.

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
```

You can also change the transaction ID of an RPL by issuing the MODCB macro:

```
MODCB   TRANSID=number
```

Writing Buffers Whose Writing Has Been Deferred

If DFR is specified in the ACB of any file that is using a resource pool, you can use the WRTBFR (write buffer) macro to write:

- All modified buffers for a given file
- All modified buffers in the resource pool
- The least recently used modified buffers in every buffer pool in the resource pool
- All buffers that have been modified by requests with the same transaction ID. (See “Relating Deferred Requests by Transaction ID”).

You can specify the DFR option in an ACB without using the WRTBFR to write buffers. A buffer will be written when VSE/VSAM needs one to satisfy a GET request, or all modified buffers will be written when the last of the files that uses them is closed.

Using WRTBFR can improve performance, if you schedule WRTBFR to overlap other processing.

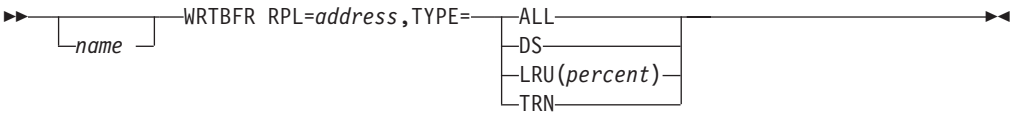
VSE/VSAM notifies the processing program when there are no more unmodified buffers into which to read the contents of a CI. (VSE/VSAM would be forced to write buffers when another GET or PUT request required an I/O operation.) VSE/VSAM sets Register 15 to 0 and puts 12 (X‘0C’) in the feedback (FDBK) field of the RPL that defines the request that detects the condition.

VSE/VSAM also notifies the processing program when there are no buffers available to process your request. This is a logic error. Register 15 contains 8, unless an exit is taken to a LERAD routine. The feedback (FDBK) field in the RPL contains 152 (X'98'). You may retry the request and it will get a buffer if one has been freed.

In addition, VSE/VSAM will notify the processing program when the number of active requests exceeds the STRNO value specified in the BLDVRP macro (Register 15=X'08'; RPL FDBK=X'40').

Format of the WRTBFR Macro

When you issue a WRTBFR macro, Register 13 must contain the address of a 72-byte save area that you are providing. When you issue a WRTBFR macro from within one of your exit routines such as LERAD or SYNAD, your program must provide a second 72-byte save area for use by VSE/VSAM, because the original save area is still in use by the external VSE/VSAM routine.



name
one to eight characters that provide a symbolic name.

RPL=address
specifies the address of the request parameter list that defines the WRTBFR request. An RPL need not be built especially for the WRTBFR; WRTBFR may use an inactive RPL that defines other request(s) (GET, PUT, etc.) for a file that is using the resource pool.

Only the ACB and the TRANSID operands of the RPL are meaningful for WRTBFR; all other RPL operands are ignored. Unlike the other action macros (GET, PUT, etc.), WRTBFR assumes that RPLs are not chained.

TYPE=ALL|DS|LRU(percent)|TRN
specifies what buffers are to be written.

ALL
specifies that all modified buffers in every buffer pool in the resource pool are to be written. (Closing all of the files that use a resource pool has the same effect.)

DS
specifies that, for the file defined by the ACB to which the WRTBFRs RPL is related all modified buffers are to be written.

LRU(percent)
specifies the percentage of the total number of buffers in every buffer pool in the resource pool that are to be examined for possible writing. The least recently used buffers are examined. (If percent is coded in register notation, only Registers 1 and 13 may not be used.)

When using the DFR option it is possible for the buffer pool to become filled with modified buffers. VSE/VSAM would then be forced to write out a buffer before satisfying any other GET or PUT request. To ensure that buffers are always available for GET or PUT requests (without having to wait for buffers to be written) you can periodically force out the least recently used part of every buffer pool through the LRU option. To help

determine when to do this, VSE/VSAM sets a non-error return code of 12 (X'0C') in the FDBK field of the RPL whenever it is forced to write out a deferred buffer because of insufficient free buffers.

TRN
specifies that all buffers in a buffer pool are to be written that have been modified by requests with the same transaction ID as the one specified in the WRTBFR's RPL. Transaction IDs are no longer associated with these buffers.

Examples: ACB, EXLST, and RPL Macros

Specifying VSE/VSAM Control Blocks

Figure 43 shows an example of how you can specify VSE/VSAM control blocks by using the ACB, EXLST, and RPL macros. These control blocks are generated during assembly of your program. Default values will be provided for those parameters that are omitted.

```
ACBADR      ACB       EXLST=EXISTS,
                      PASSWD=PASS,
                      BUFND=4,BUFNI=3,
                      BUFSP=11264,
                      MACRF=(KEY,SEQ,
                      DIR,OUT),
                      STRNO=2
EXITS       EXLST     EODAD=(ENDUP,N),
                      LERAD=LOGERR,
                      SYNAD=(IOERR,L),
                      EXCPAD=(OVERLP,A)
RETRVE      RPL       ACB=ACBADR,
                      AREA=WORK
                      ARG=SEARCH,
                      AREALEN=125,
                      OPTCD=(DIR,NSP)
            .
            .
PASS        DC        FL1'6',C'CHANGE'
WORK        DS        CL125
SEARCH      DS        CL4
IOERR       DC        C'PHYSICAL'
ENDUP                 End-of-file routine
            .
            .
LOGERR                Logical-error routine
            .
            .
OVERLP                I/O-overlap routine
            .
            .
```

Figure 43. Example of Specifying Control Blocks for a File

Explanations to Figure 43:

ACB Macro

Because the DDNAME operand is not specified, VSE/VSAM uses the name, ACBADR, of the ACB as the name (file name) of the associated file.

BUFND:
Four I/O buffers for data CIs.

BUFNI:
Three I/O buffers for index CIs.

BUFSP:
The size of the buffer space is sufficient to accommodate four data control intervals of 2048 bytes each and three index CIs of 1024 bytes each.

EXLST:
Specifies that the label of the exit list associated with this ACB is named EXITS.

PASSWD:
Specifies the location of the password. The DC at PASS gives the password's length in the first byte and the password itself in the subsequent six bytes.

MACRF:
Specifies keyed-sequential and keyed-direct processing for both insertion and update.

STRNO:
Specifies that two requests will require concurrent positioning.

EXLST Macro

EODAD:
The end-of-file routine is located at ENDUP and is not active.

LERAD:
The logic error routine is located at LOGERR and is active.

SYNAD:
The physical I/O error routine's name is located at IOERR.

EXCPAD:
The I/O-overlap routine is located at OVERLP and is active.

RPL Macro

ACB: Associates the RPL with the ACB named ACBADR.

AREA:
Address of work area is WORK.

AREALEN:
Length of work area is 125 bytes.

ARG: The search argument is defined at SEARCH. Because the KEYLEN operand is omitted, VSE/VSAM uses the full key as search argument.

OPTCD:
Specifies direct processing with positioning at the next record for subsequent sequential processing.

JCL to Open and Process a File

Figure 44 on page 267 shows the JCL needed to open and process a file identified in an ACB macro (file ACBADR in the example). Continuation characters required in column 72 are not shown in the example.

```
// JOB
// DLBL IJSYSCT,'AMASTCAT',,VSAM
// DLBL ACBADR,'FILE1',,VSAM
// EXEC progname,SIZE=AUTO
   .
   .
   .
   OPEN ACBADR
   .
   .
   .
   GET RPL=RETRVE
   .
   .
   .
   CLOSE ACBADR
   .
   .
   .
/*
/&
```

FILE1 is the name of the file under which it is entered in the VSE/VSAM master catalog.

Figure 44. Example of JCL Needed to Open and Process a File

Examples of Request Macros

The following examples show the essential macros and operands required to perform the operations indicated in the headings of the examples. The examples illustrate the relationship between the ACB MACRF operand, the RPL OPTCD operand, and the request macros themselves. They show how to use the other operands as required by the assumptions for every example.

For your convenience in reading them, the examples show macros that generate control blocks at assembly (ACB, EXLST, and RPL) at the beginning of the example rather than at the end where they would normally be placed with program constants. Every example assumes that the file has been opened and that it will be closed. Only nominal checks for errors are shown. Exit routines to analyze errors are not indicated.

Note: The continuation characters required in column 72 are not shown in the examples, nor are the asterisks required in column 1 of the comment cards shown.

The examples relate to:

- "How to Retrieve a Record: GET Macro" on page 268
- "How to Position for Subsequent Sequential Access: GET and POINT Macros" on page 273
- "How to Chain Request Parameter Lists and Terminate a Request: ENDREQ Macro" on page 276
- "How to Store a Record: PUT Macro" on page 278
- "How to Update a Record: GET and PUT Macros" on page 282
- "How to Delete a Record: GET and ERASE Macros" on page 285

How to Retrieve a Record: GET Macro

Examples 1, 2, 3, 4, and 5 illustrate keyed and addressed, direct sequential, and skip sequential retrieval.

Example 1: Keyed-Sequential Retrieval

Assumptions: Records moved to a work area. Fixed-length records, 100 bytes. Control blocks generated at assembly.

```
INPUT    ACB     MACRF=(KEY,SEQ,IN)       All MACRF and OPTCD options
                                          specified are defaults and
                                          could have been omitted.
RETRVE   RPL     ACB=INPUT,
                 AREA=IN,AREALEN=100,
                 OPTCD=(KEY,SEQ,NUP,MVE)
         .
         .
         .
LOOP     GET     RPL=RETRVE               This GET or identical GETs
                                          can be issued, with no change
                                          in the request parameter list,
                                          to retrieve subsequent records
                                          in key sequence.
         LTR     15,15
         BNZ     ERROR
         .
         .
         .
         B       LOOP
ERROR    ...                              Request was not accepted or
         .                                failed.
         .
         .
IN       DS      CL100                    IN contains a data record
                                          after GET is completed.
```

Figure 45. Request Macro Example 1: Keyed-Sequential Retrieval

Discussion: The records are retrieved in key sequence. No search argument has to be specified; VSE/VSAM is positioned at the first record in key sequence when the file is opened, and the next record is retrieved automatically as every GET is issued. The branch to ERROR will also be taken if the end of the file is reached.

Example 2: Skip-Sequential Retrieval

Assumptions: Variable-length records: they are processed in the I/O buffer. The search argument is a full key, compared greater than or equal. Control blocks are generated at the time of execution. Key length is eight bytes.

```
         GENCB       BLK=ACB,                VSE/VSAM gets an area in virtual
                     DDNAME=INPUT,           storage to generate the
                     MACRF=(KEY,SKP,IN)      access method control block
                                             and returns the address in
                                             Register 1.
         LTR         15,15
         BNZ         CHECKO
         LR          2,1                     Address of ACB
         GENCB       BLK=RPL,ACB=(2),
                     AREA=RCDADDR,AREALEN=4,
                     ARG=SCRHKEY,
                     OPTCD=(KEY,SKP,NUP,KGE,FKS,LOC)
         LTR         15,15
         BNZ         CHECK0
         LR          3,1                     Address of the request
         .                                   parameter list.
         .
         .
LOOP     MVC         SRCHKEY,table           Search argument for
                                             retrieval, moved in from
                                             a table or a
                                             transaction record.
         GET         RPL=(3)
         LTR         15,15
         BNZ         ERROR
         LR          1,3                      Put RPL address in Register 1.
         SHOWCB      RPL=(1),                 Display the length of the
                     RECLEN=(0)               record.
         LTR         15,15
         BNZ         CHECK0
         ST          0,RCDLEN                 Save the record length.
         .
         .
         .
         B           LOOP
ERROR    ...                                  Request was not accepted
                                              or failed.
CHECKO   ...                                  Generation or display failed
         .
         .
         .
RCDADDR  DS          F                        Work area into which VSE/VSAM
                                              puts the address of a data
                                              record within the I/O buffer
                                              (OPTCD=LOC).
SRCHKEY  DS          CL8                      Search argument for
                                              retrieval.
RCDLEN   DS          F                        For retrieving variable
                                              record lengths.
```

Figure 46. Request Macro Example 2: Skip-Sequential Retrieval

Discussion: The records are retrieved in key sequence, but some records are skipped. Skip sequential retrieval is very similar to keyed direct retrieval (see Example 4), except that you must retrieve records in ascending sequence (with skips) rather than in a random sequence.

Internally, with skip sequential retrieval, VSE/VSAM uses only the sequence set of the index to skip ahead; with direct retrieval it searches the index from top to bottom to locate a record.

Example 3: Addressed-Sequential Retrieval

Assumptions: Many records are retrieved with one GET request. Records are moved to work areas (only option); they are of fixed length, 20 bytes long. Chain of RPLs is generated during execution.

```
BLOCK    ACB      DDNAME=INPUT,
                  MACRF=(ADR,SEQ,IN)
         .
         .
         .
         GENCB    BLK=RPL,
                  COPIES=10,
                  ACB=BLOCK,
                  OPTCD=(ADR,SEQ,NUP,MVE)
         LTR      15,15
         BNZ      CHECKO
         LA       5,WORKAREA             Address of the first
                                         work area.
         LA       3,10                   Number of lists(10).
         LR       2,1                    Address of the first list.
         LR       1,0                    Length of all of the lists.
                                         Registers 0 and 1 contain
                                         length and address of the
                                         generated control blocks when
                                         VSE/VSAM returns control after
                                         GENCB.
         SR       0,0                    Prepare for following
                                         division.
         DR       0,3                    Divide number of lists into
                                         length of all the lists.
         LR       3,1                    Save the resulting length of a
                                         sing le list for an offset.
         LR       4,2                    Save address of the first list.
```

Figure 47. Request Macro Example 3: Addressed-Sequential Retrieval (Part 1 of 2)

Do the following 6 instructions 10 times to set up all of the request parameter lists. The tenth time register 4 must be set to 0 to indicate the last request parameter list in the chain.

```
         AR       4,3                    Address of the next list.
         MODCB    RPL=(2),               In every request parameter
                  NXTRPL=(4),            list, indicate the address
                  AREA=(5),AREALEN=20    of the next list and the
                                         address and length of the
                                         work area.

         LTR      15,15
         BNZ      CHECKO
         AR       2,3                    Address the next line.
         LA       5,20(5)                Address the next work area.
         .                               Restore Register 2 to address
         .                               the first list before
         .                               continuing to process.
LOOP     GET      RPL=(2)
         LTR      15,15
         BNZ      ERROR
         .                               Process the ten records that
         .                               have been retrieved by the
         .                               GET.
         B        LOOP
CHECKO   ...
ERROR    ...                             Display the feedback field
                                         (FIELDS=FDBK) of every
                                         request parameter list to
                                         find out which one had an
                                         error.
WORKAREA DS       CL200                  Space for a work area for
                                         each of the 10 request
                                         parameter lists.
```

Figure 47. Request Macro Example 3: Addressed-Sequential Retrieval (Part 2 of 2)

Discussion: The records are retrieved in entry sequence. In a key-sequenced file that has had CI or control-area splits, it is likely that the entry sequence of the records is no longer the same as their key sequence. Each of the ten RPLs in the chain identifies a record to be retrieved by the GET. VSE/VSAM moves every record into the work area provided for the request parameter list that identifies the record.

If an error occurred for one of the RPLs in the chain and you have supplied error analysis routines, VSE/VSAM takes a LERAD or SYNAD exit before returning to your program. Register 15 indicates the status of the request. A code of 0 indicates that no error was associated with any of the RPLs. Any other code indicates that an error occurred for one of the RPLs. Issue a SHOWCB for every RPL in the chain to find out which one had an error. VSE/VSAM does not process any of the RPLs beyond the one with an error.

Example 4: Keyed-Direct Retrieval

Assumptions: Fixed-length records are processed in the I/O buffer. Key length is 15 bytes. The search argument is a 5-byte generic key, compared equal. Control blocks are generated during assembly.

```
INPUT    ACB      MACRF=(KEY,DIR,IN)

RETRVE   RPL      ACB=INPUT,              You specify all parameters
                  AREA=IN,AREALEN=4       for the request in the RPL
                  OPTCD=(KEY,DIR,         macro.
                  NUP,KEQ,GEN,LOC),
                  ARG=KEYAREA,KEYLEN=5
         .
         .
         .
LOOP     MVC      KEYAREA,table           Search argument for retrieval,
                                          moved in from a table or a
                                          transaction record.
         GET      RPL=RETRVE              This GET or identical GETs can
                                          be issued with no change in the
                                          RPL: just specify every new
                                          search argument in the field
                                          KEYAREA.
         LTR      15,15
         BNZ      ERROR
         .
         .                                Process the record.
         .
         B        LOOP
ERROR    ...                              Request was not accepted
         .                                or failed.
         .
         .
IN       DS       CL4                     VSE/VSAM stores the address
         .                                of the record here.
         .
         .
KEYAREA  DS       CL5                     You specify the search
                                          argument here.
```

Figure 48. Request Macro Example 4: Keyed-Direct Retrieval

Discussion: The generic key specifies a class of records. For example, if you search on the first third of employee number, you get the first of presumably several records that start with the specified characters. To retrieve all of the records in that class, either switch to sequential access (see Example 7) or to a full-key search with greater-than-or-equal comparison (Example 2), increasing the key of every record you retrieve to the next possible key value.

Example 5: Addressed-Direct Retrieval

Assumptions: Fixed-length records, 20 bytes long, are moved to a work area.

```
BLOCK    ACB      DDNAME=INPUT,          Access-method control block
                  MACRF=(ADR,DIR,IN)     generated at assembly.
         GENCB    BLK=RPL,COPIES=1,      Request parameter list
                  ACB=BLOCK,             generated at execution.
                  OPTCD=(ADR,DIR,
                  NUP,MVE),
                  ARG=SRCHADR,
                  AREA=IN,AREALEN=20
         LTR      15,15
         BNZ      CHECKO
         LR       2,1                    Address of the list.
         .
         .
         .
LOOP     MVC      SRCHADR,table          Search argument for retrieval,
                                         calculated or moved in from a
                                         table or a transaction record.
         GET      RPL=(2)
         LTR      15,15
         BNZ      ERROR
         .
         .                               Process the record.
         .
         B        LOOP
CHECKO   ...                             Generation failed.
ERROR    ...                             Request was not accepted
         .                               or failed.
         .
         .
IN       DS       CL20                   VSE/VSAM puts a record here
                                         for every GET request.
SRCHADR  DS       CL4                    You specify the RBA search
                                         argument here for every request.
```

Figure 49. Request Macro Example 5: Addressed-Direct Retrieval

Discussion: The RBA provided for a search argument must match the RBA of a record. Keyed insertion and deletion of records in a key-sequenced file will probably cause the RBAs of some records to change. Therefore, if you process a key-sequenced file by addressed direct access (or by addressed sequential access using POINT), you need to keep track of changes. You can use the JRNAD exit for this purpose.

How to Position for Subsequent Sequential Access: GET and POINT Macros

Examples 6 and 7 illustrate positioning both with the POINT macro and with direct access followed by sequential access.

Example 6: Keyed Positioning with POINT

Assumptions: Sequential access. The search argument (for positioning) is a full key of 5 bytes, compared equal. Records are 50 bytes long. Control blocks are generated during assembly.

```
BLOCK    ACB      DDNAME=10              Default MACRF options
                                         sufficient.
POSITION RPL      ACB=BLOCK,             ARG operand and KEQ and
                  AREA=WORK,AREALEN=50,  FKS OPTCD options define
                  ARG=SRCHKEY,           the POINT request.
                  OPTCD=(KEY,SEQ,
                  KEQ,FKS)
         .
         .
         .
LOOP     MVC      SRCHKEY,table          Search argument for
                                         positioning, moved in from a
                                         table or transaction record.
         POINT    RPL=POSITION
         LTR      15,15
         BNZ      ERROR
LOOP1    GET      RPL=POSITION
         LTR      15,15
         BNZ      ERROR
         .                               Process the record.  Decide
         .                               whether to skip another
         .                               position (forward or
                                         backward).
         BE       LOOP                   Yes, skip.
         B        LOOP1                  No, continue in consecutive
                                         sequence.
ERROR    ...                             Request was not accepted or
         .                               failed.
         .
         .
SRCHKEY  DS       CL5                    Search-argument field for
                                         POINT request.
WORK     DS       CL50                   VSE/VSAM puts a record here
                                         for every GET request.
```

Figure 50. Request Macro Example 6: Keyed Positioning with POINT

Discussion: No access is gained to a record with POINT. POINT causes VSE/VSAM to be positioned ahead or back to the specified record for a subsequent sequential GET request, which retrieves the record. If, after positioning, you issue a direct request by way of the same RPL, VSE/VSAM does not remember the position established by the POINT. VSE/VSAM would then either be positioned somewhere else or not positioned at all, depending on whether OPTCD=NSP or UPD was specified or OPTCD=NUP (see Example 7).

Positioning by address is identical to positioning by key, except that the search argument is an RBA, which must match with the RBA of a record in the file.

Example 7: Switching from Direct to Keyed-Sequential Retrieval

Assumptions: Records are moved to a work area. The search argument (for the direct request preceding sequential requests) is a generic key, 8 bytes long, compared equal. Records are of fixed-length, 100 bytes long. Positioning is requested for direct requests. Control blocks are generated during assembly.

```
INPUT    ACB      MACRF=(KEY,DIR,          Both direct and sequential
                  SEQ,IN)                  access specified.
RETRVE   RPL      ACB=INPUT,
                  AREA=IN,AREALEN=100,
                  OPTCD=(KEY,DIR,
                  NSP,KEQ,GEN,MVE),        NSP specifies that VSE/VSAM is
                  ARG=KEYAREA,KEYLEN=8     to remember its position.
         .
         .
         .
LOOP     MVC      KEYAREA,table            Search argument for direct
                                           retrieval, moved in from a
                                           table or transaction record.
LOOP1    GET      RPL=RETRVE
         LTR      15,15
         BNZ      ERROR
         .                                 Decide whether to switch from
         .                                 one type of access to the
         .                                 other.  If now sequential:
                                           To remain sequential, branch
                                           to LOOP1
                                           To switch to direct, branch
                                           to DIR
                                           If now direct:
                                           To remain direct, branch
                                           to LOOP
                                           To switch to sequential,
                                           branch to SEQ
SEQ      MODCB    RPL=RETRVE,              Alter request parameter list
                  OPTCD=SEQ                for sequential access.
         LTR      15,15
         BNZ      CHECKO
         B        LOOP1                    No search argument required.
DIR      MODCB    RPL=RETRVE,              Alter request parameter list
                  OPTCD=DIR                for direct access.
         LTR      15,15
         BNZ      CHECK0
         B        LOOP                     Prepare new search argument.
ERROR    ...                               Request was not accepted or
                                           failed.
CHECKO   ...                               Modification failed.
         .
         .
         .
IN       DS       CL100                    VSE/VSAM puts retrieved records
                                           here.
KEYAREA  DS       CL8                      Specify the generic key for
                                           a direct request here.
```

Figure 51. Request Macro Example 7: Switching from Direct to Keyed-Sequential

Discussion: Positioning is associated with an RPL; thus to switch from direct to sequential access without independently establishing positioning for the sequential access, modify a single RPL that alternately defines requests for both types of access rather than use a different RPL for every type.

With direct retrieval, VSE/VSAM does not remember its position for subsequent retrieval unless you explicitly requested this (OPTCD=NSP). After a direct GET for update (OPTCD=UPD), VSE/VSAM is positioned for a subsequent PUT,ERASE, or sequential GET (if you modify OPTCD(DIR,UPD) to OPTCD=(SEQ,UPD)). If you modify OPTCD=(DIR,NUP) to OPTCD=SEQ, you must issue a POINT to get VSE/VSAM positioned for sequential retrieval, as NUP indicates that no positioning is desired with a direct GET.

If you have chained many RPLs together, one position is remembered for the whole chain. For example, if you issue a GET that gives the address of the first RPL in the chain, the position of VSE/VSAM when the GET request is complete is at the record following the one defined by the last RPL in the chain. Therefore, modifying OPTCD=(DIR,NSP) in every RPL in a chain to OPTCD=SEQ implies continuing with sequential access relative to the last of the direct RPLs.

How to Chain Request Parameter Lists and Terminate a Request: ENDREQ Macro

Example 8 illustrates how to chain RPLs. Example 9 illustrates the use of ENDREQ to cause VSE/VSAM to give up its position for a request to be able to remember its position for another request.

Example 8: Chaining Request Parameter Lists

Assumptions: Records are 50 bytes long. Retrieved records are moved to a work area. Three RPLs are chained.

```
FIRST    RPL      ACB=BLOCK,
                  AREA=AREA1,AREALEN=50,
                  NXTRPL=SECOND

SECOND   RPL      ACB=BLOCK,
                  AREA=AREA2,AREALEN=50,
                  NXTRPL=THIRD
THIRD    RPL      ACB=BLOCK               Last list does not indicate
                  AREALEN=50              a next list.
                  AREALEN=50
         .
         .
         .
LOOP     GET      RPL=FIRST               Request gives the address of
                                          the first request parameter
                                          list.
         LTR      15,15
         BNZ      ERROR
         .                                Process the three records
         .                                retrieved by the GET.
         .
         B        LOOP
ERROR    ...                              Display the feedback field
                                          (FIELD=FDBK) of every request
                                          parameter list to find out
                                          which one had an error.
AREA1    DS       CL50                    A single GET request causes
                                          VSE/VSAM to put a record in
                                          each one of AREA1, AREA2,
                                          and AREA3.
AREA2    DS       CL50
AREA3    DS       CL50
```

Figure 52. Request Macro Example 8: Chaining Request Parameter Lists

Discussion: If an error occurred for one of the RPLs in the chain and you have supplied error-analysis routines, VSE/VSAM takes a LERAD or SYNAD exit before it returns control to your program. Register 15 is set to indicate the status of the request. A code of 0 indicates that no error was associated with any of the RPLs. Any other code indicates that an error occurred for one of the RPLs. You should issue a SHOWCB macro for every RPL in the chain to find out which one had an error. VSE/VSAM does not process any of the RPLs beyond the one with an error.

Example 9: Giving up Positioning for Another Request

Assumptions: There are three RPLs, all of which require VSE/VSAM to remember its position, one only temporarily and two until VSE/VSAM is explicitly requested to forget its position. VSE/VSAM can remember only two positions concurrently (STRNO=2).

```
BLOCK    ACB      MACRF=(SEQ,DIR),
                  STRNO=2
SEQ      RPL      ACB=BLOCK,              VSE/VSAM must remember its
                  OPTCD=SEQ               position.
DIRUPD   RPL      ACB=BLOCK,              VSE/VSAM must remember its
                  OPTCD=(DIR,UPD)         position until explicitly
                                          requested to forget it by
                                          PUT or ENDREQ.
DIRNUP   RPL      ACB=BLOCK,              VSE/VSAM must be able to
                  OPTCD=(DIR,NUP)         temporarily remember its
         .                                position.
         .
         .
LOOP     GET      RPL=SEQ                 VSE/VSAM now remembers its
                                          position for this request.
         LTR      15,15
         BNZ      ERROR
         GET      RPL=DIRNUP              VSE/VSAM remembers its position
                                          for this request only while
                                          it is processing the request.
         LTR      15,15
         BNZ      ERROR
         GET      RPL=DIRUPD              VSE/VSAM can therefore remember
                                          its position for this request,
                                          even that STRNO=2.
         LTR      15,15
         BNZ      ERROR
         .
         .                                Decide whether to update the
         .                                record.
         BE       UPDATE
         B        FORGET                  No.
UPDATE   PUT      RPL=DIRUPD              Yes, update the record,
                                          causing VSE/VSAM to forget its
                                          position for DIRUPD.
         LTR      15,15
         BNZ      ERROR
         B        LOOP
FORGET   ENDREQ   RPL=DIRUPD              Cause VSE/VSAM to forget its
                                          position for DIRUPD.
         LTR      15,15
         BNZ      ERROR
         B        LOOP
ERROR    ...                              Request was not
                                          accepted or failed.
```

Figure 53. Request Macro Example 9: Giving up Positioning for Other Request

Discussion: The use of ENDREQ illustrated here is to cause VSE/VSAM to forget its position for one RPL so a request defined by another RPL can be issued. When PUT is issued after a GET RPL=DIRUPD request, ENDREQ need not be issued, because PUT causes VSE/VSAM to forget its position (the next GET with RPL=DIRUPD does not depend on VSE/VSAM's remembering its position). You need to cause VSE/VSAM to forget its position when you have issued requests for as many RPLs requiring concurrent positioning as the number you specified for STRNO (in the ACB macro) and you want to issue a request for yet another RPL.

In the example, a GET with RPL=DIRNUP cannot be reissued unless VSE/VSAM is freed from remembering its position for either SEQ or DIRNUP. VSE/VSAM must be allowed to remember its position for SEQ because requests against this RPL are sequential and depend on VSE/VSAM's remembering its position.

To cause VSE/VSAM to give up its position associated with a chain of RPLs, specify the first RPL in the chain in your ENDREQ macro.

Because VSE/VSAM remembers its position after a direct GET with OPTCD=UPD, if no PUT or ENDREQ follows, you can switch to sequential access (OPTCD=(SEQ,UPD) or OPTCD=SEQ) and use the positioning for a GET.

How to Store a Record: PUT Macro

Examples 10, 11, 12, and 13 illustrate the storage of records: keyed and addressed, sequential, skip sequential, and direct.

Example 10: Keyed-Sequential Insertion

Assumptions: Records of variable length are moved from a work area (only option). These records are up to 250 bytes long. Key length is 15 bytes. Some records are inserted between existing records, others are added at the end of the file.

```
BLOCK    ACB      DDNAME=OUTPUT,
                  MACRF=(KEY,SEQ,OUT)

LIST     RPL      ACB=BLOCK,
                  AREA=BUILDRCD,AREALEN=250,
                  OPTCD=(KEY,SEQ,NUP,MVE)
         .
         .
LOOP     L        0,length              Put length of record to be
                                        inserted into Register 0.
         LA       1,LIST                Put RPL address into Register 1.
         MODCB    RPL=(1),              Modify record length in
                  RECLEN=(0)            request parameter list.
         LTR      15,15
         BNZ      CHECKO
         PUT      RPL=LIST
         LTR      15,15
         BNZ      ERROR
         B        LOOP
CHECKO   ...                            Modification failed.
ERROR    ...                            Request was not accepted
         .                              or failed.
         .
         .
BUILDRCD DS      CL250                 Work area for building record.
```

Figure 54. Request Macro Example 10: Keyed-Sequential Insertion

Discussion: Sequential insertion does not require VSE/VSAM to be positioned at the point of insertion. VSE/VSAM automatically skips ahead (never back) to that point, as though you were using skip sequential insertion (see Example 11). The difference between sequential and skip sequential insertion is that sequential insertion leaves free space in CIs and CAs according to the file's FREESPACE specification in the catalog (which is entered by the IDCAMS command DEFINE). Skip sequential insertion (and direct insertion) uses the free space.

You must use sequential storage (as opposed to skip sequential or direct storage) when you load records into a file for the first time. Thereafter, you may use skip sequential and direct storage, but you should use sequential storage when you are inserting large numbers of records between two existing records or at the end of the file.

When you store records sequentially beyond the highest key in the file, VSE/VSAM automatically extends the file as though you were continuing to load records. VSE/VSAM does not use distributed free space for these records, but establishes new CAs at the end of the file.

Example 11: Skip-Sequential Insertion

Assumptions: Several records are inserted with one PUT request. The records are moved from a work area (only option). They are fixed-length, 100 bytes long.

```
OUTPUT   ACB      MACRF=(KEY,SKP,OUT)
         .
         .
         .
         GENCB    BLK=RPL,COPIES=5,       Generate 5 request parameter
                  ACB=OUTPUT,             lists at execution.
                  AREALEN=100,
                  OPTCD=(KEY,SKP,NUP,MVE)
         LTR      15,15
         BNZ      CHECKO
```

Calculate the length of every list and use register notation with the MODCB macro to complete each list. See Example 3.

```
         MODCB    RPL=(2),
                  AREA=(3),
                  NXTRPL=(4)
         LTR      15,15
         BNZ      CHECKO
```

Increase the value in every register and repeat the MODCB until all five requests have been completed. The last time, Register 4 must be set to 0.

```
         .
         .
         .
LOOP     ...                              Restore address of first list
                                          Register 2.  Build 5 records
                                          in WORK.
         PUT      RPL=(2)                 Register 2 points to the first
                                          request parameter list in the
                                          chain.  The five records in
                                          WORK are stored with this one
                                          PUT request.
         LTR      15,15
         BNZ      ERROR
         .
         .
         .
         B        LOOP
CHECKO   ...                              Generation or modification
                                          failed.
ERROR    ...                              Display the feedback field
                                          in every request parameter
                                          list to find out if it had
                                          an error (see discussion
                                          for Example 8).
WORK     DS       CL500                   Contains 5 100-byte work
                                          areas.
```

Figure 55. Request Macro Example 11: Skip-Sequential Insertion

Discussion: You give no search argument for storage. VSE/VSAM knows the position of the key field in every record and extracts the key from it. Skip sequential insertion differs from keyed direct insertion in the sequence in which records may be inserted (ascending non-consecutive sequence versus random sequence) and in performance. With skip sequential insertion, VSE/VSAM uses only the sequence set of the index to find the point of insertion; with keyed direct insertion, VSE/VSAM searches from the top level of the index down to the sequence set.

With skip sequential insertion, if you insert two or more records into a CI, VSE/VSAM does not write the contents of the buffer to direct-access storage until you have inserted all records. With direct insertion, VSE/VSAM writes the contents of the buffer after you have inserted each record.

Example 12: Keyed-Direct Insertion

Assumptions: Records are moved from a work area (only option.) They have a fixed length of 100 bytes.

```
OUTPUT   ACB      MACRF=(KEY,DIR,OUT)

DIRECT   RPL      ACB=OUTPUT,
                  AREA=WORK,AREALEN=100,
                  OPTCD=(KEY,DIR,NUP,MVE),
                  RECLEN=100
         .
         .
         .
LOOP     PUT      RPL=DIRECT
         LTR      15,15
         BNZ      ERROR
         .
         .
         .
         B        LOOP
ERROR    ...                              Request failed
         .
         .
         .
WORK     DS       CL100                   Work area
```

Figure 56. Request Macro Example 12: Keyed-Direct Insertion

Discussion: VSE/VSAM extracts the key from every record's key field. You give no search argument. Using keyed direct access is very similar to using skip sequential access. About the only differences are specifying DIR instead of SKP in the MACRF and OPTCD operands and being able to process records randomly instead of in ascending key sequence (with skips).

Example 13: Addressed-Sequential Addition

Assumptions: Records are moved from work area (only option). They are of variable-length, up to 100 bytes long.

```
BLOCK    ACB      MACRF=(ADR,SEQ,OUT)

LIST     RPL      ACB=BLOCK,
                  AREA=NEWRCD,AREALEN=100,
                  OPTCD=(ADR,SEQ,MVE)
         .
         .
         .
LOOP     ...                            Build the record.
         L        0,length              Put the length of the record
                                        into Register 0.
         LA       1,LIST                Put RPL address into Register 1.
         MODCB    RPL=(1),              Indicate length of new record.
                  RECLEN=(0)
         LTR      15,15
         BNZ      CHECKO
         PUT      RPL=LIST
         LTR      15,15
         BNZ      ERROR
         B        LOOP
CHECKO   ...                            Modification failed.
ERROR    ...                            Request was not accepted
         .                              or failed.
         .
         .
NEWCRD   DS       CL100                 Build record in this work
                                        area.
```

Figure 57. Request Macro Example 13: Addressed-Sequential Addition

Discussion: With addressed access, you cannot insert records into or add records to a key-sequenced file, because the index is not used and VSE/VSAM cannot locate the CI into which to insert the record. You can add records to, but not insert records into, an entry sequenced-file. Every record is stored in the next position after the last record in the file. You do not have to specify an RBA or do any explicit positioning (with the POINT macro). Addressed addition of records is always identical to loading a file. When the last CA is filled up, VSE/VSAM extends the file and establish new CAs.

Actually, there is no difference between addressed sequential and addressed direct addition. Every method stores a record in the next position after the last record in the file. However, you cannot use direct processing to load records into a file for the first time; you must use sequential processing.

How to Update a Record: GET and PUT Macros

Examples 14, 15, and 16 illustrate updating a record by first retrieving it and then storing it back with changes. (You cannot update a record without first retrieving it for update.)

Example 14: Keyed-Sequential Update

Assumptions: Records are updated in a work area (only option). They are fixed-length, 50 bytes long. Not every record retrieved is also updated.

```
UPDATA   ACB      MACRF=(KEY,SEQ,OUT)

LIST     RPL      ACB=UPDATA,             UPD indicates the record
                  AREA=WORK,AREALEN=50,   may be stored back(or
                  OPTCD=(KEY,SEQ,         deleted).
                  UPD,MVE)
         .
         .
         .
LOOP     GET      RPL=LIST
         LTR      15,15
         BNZ      ERROR
         .                                Decide whether to update
         .                                the record.
         .
         BE       UPDATE
         B        LOOP                    Do not update it; retrieve
                                          another.

UPDATE   .                                Update the record and store
         .                                it back.
         .
         PUT      RPL=LIST
         LTR      15,15
         BNZ      ERROR
         B        LOOP
ERROR    ...                              Request was not accepted
         .                                or failed.
         .
         .
         .
WORK     DS      CL50                     VSE/VSAM places the retrieved
                                          record here.
```

Figure 58. Request Macro Example 14: Keyed-Sequential Update

Discussion: A GET update (OPTCD=UPD) must precede a PUT for update. Besides retrieving the record to be updated, GET positions VSE/VSAM at the record retrieved in anticipation of the succeeding update (or deletion). It is not necessary to store back (or delete) the record that you retrieved for update. VSE/VSAM's position at the record previously retrieved allows you to issue another GET to retrieve the following record (OPTCD=(SEQ,UPD) or OPTCD=SEQ). Then, however, the position for update was not maintained because of the following GET.

This example requires the use of a work area because you cannot update a record in the I/O buffer. Skip sequential retrieval (with OPTCD=UPD) can be used to update. Compare this example with Example 2.

Example 15: Keyed-Direct Update

Assumptions: Records are moved to and from a work area (only option). They are of variable-length, up to 120 bytes (with some lengths changed by update). The search argument is a full key of five bytes, compared equal.

```
INPUT    ACB      MACRF=(KEY,DIR,OUT)

UPDATE   RPL      ACB=INPUT,              UPD indicates the record
                  AREA=IN,AREALEN=120,    may be stored back (or
                  OPTCD=(KEY,DIR,         deleted).
                  UPD,KEQ,FKS,MVE),
                  ARG=KEYAREA,KEYLEN=5
         .
         .
         .
LOOP     GET      RPL=UPDATE
         LTR      15,15
         BNZ      ERROR
         LA       1,UPDATE                Put RPL address in Register 1.
         SHOWCB   RPL=(1),                Display the length of the
                  RECLEN=(0)              record.
         LTR      15,15
         BNZ      CHECKO
         ST       0,RLNGTH                Save the record length.
         .                                Update the record.  Does the
         .                                update change the record's
         .                                length?

         B        STORE                   No, length not changed.
         L        0,length                Yes, load new length into
                                          Register 0.
         LA       1,UPDATE                Put RPL address in Register 1.
         MODCB    RPL=(1),                Modify length indication in
                  RECLEN=(0)              the request parameter list.
         LTR      15,15

          BNZ      CHECKO
STORE    PUT      RPL=UPDATE
         LTR      15,15
         BNZ      ERROR
         B        LOOP
ERROR    ...                              Request was not accepted
                                          or failed.
CHECKO   ...                              Display or modification
         .                                failed.
         .
         .
IN       DS       CL120                   Work area for retrieving,
                                          updating, and storing a
                                          record
KEYAREA  DS       CL5                     Search argument for
                                          retrieving a record.
RLNGTH   DS       F                       Area for displaying the
                                          length of a retrieved record.
```

Figure 59. Request Macro Example 15: Keyed-Direct Update

Discussion: You cannot update records in the I/O buffer. A direct GET for update positions VSE/VSAM at the record retrieved, in anticipation of storing back (or deleting) the record. This positioning also allows you to switch to sequential access to retrieve the next record.

You do not have to store back a record that you retrieve for update, but if you do another retrieval, (using the same RPL). Or else, use two RPLs with STRNO=2. One RPL is used solely for GET DIR with UPD.

Example 16: Addressed-Sequential Update

Assumptions: Entry-sequenced file. Records are processed in a work area. They are of variable-length, up to 200 bytes long (lengths are not changed by updates; the length of a record can never be changed if addressed access is used).

```
ENTRY    ACB      MACRF=(ADR,SEQ,OUT)

ADRUPD   RPL      ACB=ENTRY,              UPD indicates update
                  AREA=WORK,              (or deletion).
                  AREALEN=200,
                  OPTCD=(ADR,SEQ,UPD,MVE)
         .
         .
         .
LOOP     GET      RPL=ADRUPD
         LTR      15,15
         BNZ      ERROR
         LA       1,ADRUPD                Put RPL address in Register 1.
         SHOWCB   RPL=(1),                Find out how long the record is.
                  RECLEN=(0)
         LTR      15,15
         BNZ      CHECKO
         ST       0,RLNGTH                Save the record length.
         PUT      RPL=ADRUPD
         LTR      15,15
         BNZ      ERROR
         B        LOOP
ERROR    ...                              Request was not accepted
                                          or failed.
CHECKO   ...                              Display failed.
         .
         .
         .
WORK     DS       CL200                   Record-processing work area.
RLNGTH   DS       F                       Display area for length of
                                          records.
```

Figure 60. Request Macro Example 16: Addressed-Sequential Update

Discussion: The RBAs of records in an entry-sequenced file are fixed and free space is not distributed. Therefore, it is not possible to change the length of records in an entry-sequenced file.

If you have inactive records in your entry-sequenced file, you may reuse the space they occupy by retrieving the records for update and restoring a new record in their place.

With a key-sequenced file, it is also impossible to change the length of records by addressed update because the index is not used and VSE/VSAM could not split a CI if required because of changing record length.

Addressed direct update differs from sequential update in the specification of an RBA for a search argument.

How to Delete a Record: GET and ERASE Macros

Examples 17 and 18 illustrate deleting a record from a key-sequenced file.

Example 17: Keyed-Direct Deletion

Assumptions: Records are processed in a work area (only option). They are fixed-length, 50 bytes long. Not every record retrieved for deletion is deleted. The search argument is a full key, 5 bytes long, compared equal.

```
DELETE   ACB      MACRF=(KEY,DIR,OUT)
LIST     RPL      ACB=DELETE,             UPD indicates deletion.
                  AREA=WORK,AREALEN=50,
                  ARG=KEYFIELD,
                  OPTCD=(KEY,DIR,UPD,MVE,FKS,KEQ)
         .
         .
         .
LOOP     MVC      KEYFIELD,table          Search argument for retrieval,
                                          from a table or transaction
                                          record.
         GET      RPL=LIST
         LTR      15,15
         BNZ      ERROR
         .
         .
         .                                Decide whether to delete
                                          the record.
         BE       DELET
         B        LOOP                    No, retrieve the next record.
DELET    ERASE    RPL=LIST                Yes, delete the record.
         LTR      15,15
         BNZ      ERROR
         B        LOOP
ERROR    ...                              Request was not accepted
                                          or failed.
WORK     DS       CL50                    Examine the data record here.
KEYFIELD DS       CL5                     Search argument.
```

Figure 61. Request Macro Example 17: Keyed-Direct Deletion

Discussion: When you retrieve a record for deletion (OPTCD=UPD, same as retrieval for update), VSE/VSAM is positioned at the record retrieved, in anticipation of a following ERASE (or PUT) request for that record. You are not required to issue such a request, however. Another GET request nullifies any previous positioning for deletion or update.

Keyed sequential retrieval for deletion varies from direct in not using a search argument (except for possible use of the POINT macro). Skip sequential retrieval for deletion (OPTCD=(SKP,UPD) has the same effect as direct, but it is faster or slower depending on the number of CIs separating the records to be retrieved.

Example 18: Addressed-Sequential Deletion

Assumptions: Records are processed in a work area. They are fixed-length, 100 bytes long. Not every record that is retrieved is deleted. Skipping is effected by issuing the POINT macro.

```
DELETE   ACB      MACRF=(ADR,SEQ,OUT)

REQUEST  RPL      ACB=DELETE,             UPD indicates deletion.
                  AREA=WORK,
                  AREALEN=100,
                  ARG=ADDR,
                  OPTCD=(ADR,SEQ,UPD,MVE)
         .
         .
         .
LOOP     ...                              Decide whether you need to
                                          skip to another position
                                          (forward or backward).
         B        RETRIEVE                No, bypass the POINT.
         MVC      ADDR,RBA value          Yes, move search argument for
                                          POINT into search-argument
                                          field.
         POINT    RPL=REQUEST             Position VSE/VSAM to the record
                                          to be retrieved next.
         LTR      15,15
         BNZ      ERROR
RETRIEVE GET      RPL=REQUEST
         LTR      15,15
         BNZ      ERROR
         .
         .
         .                                Decide whether to delete
                                          the record.
         BE       DELET
         B        LOOP                    No, skip ERASE.
DELET    ERASE    RPL=REQUEST             Yes, delete the record.
         LTR      15,15
         BNZ      ERROR
         B        LOOP
ERROR    ...                              Request was not accepted
         .                                or failed.
         .
         .
ADDR     DS       F                       RBA search argument for
                                          POINT.
WORK     DS       CL100                   Work area.
```

Figure 62. Request Macro Example 18: Addressed-Sequential Deletion

Discussion: Addressed deletion is allowed only for a key-sequenced file. The records of an entry-sequenced file are fixed, both in their existence and in their location.

How to Use Extended User Buffering: GET and PUT Macros

User buffering is mostly used by database systems such as DL/I and SQL/DS. Extended user buffering reduces the number of I/O requests and contributes to an increase in performance. Extended user buffering is provided for VSE/VSAM ESDS files and can be requested via the VSE/VSAM RPL control block. The support became available for the first time with VSE/ESA 2.1.2.

Current User Buffering Support

User buffering is only possible in conjunction with the *control interval mode*, that is, the ACB specifies: `MACRF=(CNV,UBF,MVE,FWD)`. With user buffering, the RPL identifies the buffer address via the `AREA=` parameter and, in case of direct access, the RBA of the control interval to be read or written via the RPL `ARG=` parameter.

Since a READ request needs to be processed immediately and (on a PUT request) the buffer cannot be copied from the user buffer, each VSE/VSAM request with user buffering results in an immediate I/O for each single control interval.

If several RPLs are chained via the NXTRPL= option of the RPL, the situation is unchanged, because each request as identified via each RPL is executed independently. In addition, certain restrictions exist for use with RPL chaining.

Extended User Buffering Support

Extended user buffering improves performance by handling related I/O requests, as far as possible, as a **single** I/O request. This reduces the number of I/O requests required by a factor that is usually the number of buffers per extended user buffering request.

To support extended user buffering, the options NBF and XBF are added to the RPL macro OPTCD options:

```
OPTCD=(...,NBF,...)
OPTCD=(...,XBF,...)
```

NBF and XBF are also added as operands to the macros GENCB, MODCB, and TESTCB.

OPTCD

Meaning

NBF Normal user buffering.

Each request as identified by an RPL is executed serially and independently. This is the conventional processing of user buffering and remains the default.

XBF Extended user buffering.

VSE/VSAM will execute the chain of RPLs as a *single request*, thereby attempting to execute the requests with as few I/O requests as possible. All control intervals residing in the same control area will usually be processed in a single I/O.

Using Extended User Buffering

When using extended user buffering, the following must be observed:

1. OPTCD=UBF and OPTCD=XBF can be used interchangeably.
2. To perform an actual request, an application must:
   a. Set up a chain of as many RPLs as control intervals are to be read or written.
   b. Store the RBA and buffer address (AREA, AREALEN) information for **each** control interval into the associated RPL.
   c. Execute the appropriate request macro against the first RPL of the chain.
3. With OPTCD=XBF, the following request macros are valid:
   GET
   PUT
4. All requests will be executed as if OPTCD=(DIR,NUP,MVE) were specified. OPTCD=SEQ is allowed but it would only affect the cache handling (DEFINE EXTENT, global attributes) for ECKD devices. OPTCD=SEQ should only be set for an application that has to process large portions of data sequentially in forward direction.

5. There is no *exclusive control* handling. The consequences are:
   - It is possible to update a control interval without having it read previously.
   - There is no protection by VSE/VSAM that different strings update a control interval concurrently.
6. With `OPTCD=XBF`, it is not possible to extend a data set. Hence pre-formatting would still need to be done with either `OPTCD=NBF`, or with VSE/VSAM buffering (`MACRF=NUB`).

Errors

If extended user buffering was incorrectly used the request will be rejected as logical error (R15=8) with error code 106 (X'6A').

Return Codes of Request Macros

When VSE/VSAM returns to your processing program, a return code in Register 15 indicates what happened. If an error occurred, the RPL contains additional information. Your processing program can examine the feedback field of the RPL with the SHOWCB or TESTCB macro. Register 1 contains the address of the RPL which defines the request that caused the error.

Control is returned to the instruction following your action macro when:

1. The request was completed normally
2. The request was not accepted because another request was using the RPL
3. An error occurred and you did not have an active exit routine
4. An error occurred and you had an active exit routine.

   The routine returns control to VSE/VSAM after processing the error. VSE/VSAM then returns control to the instruction following the action macro.

When you gain control after a request, Register 15 contains one of the following return codes:

Return Code	**Meaning**
X'00'	Request completed successfully; the RPL might contain additional (non-error) information about the request.
X'04'	The request was not accepted because a request from another task is active on the same RPL; no additional information is in the RPL.
X'08'	Logic error; the error code in the RPL identifies the specific error. End-of-file is considered a logic error (error code X'04').
X'0C'	Uncorrectable I/O error; the error code in the RPL identifies the specific error.

Note: For information on return and error codes, refer to see the "VSE/VSAM return and error codes" in the *z/VSE Messages and Codes*.

As applicable, also refer to the descriptions of the request macros (GET, PUT, POINT, ERASE, and ENDREQ).

Depending on the return code in Register 15 and your specification in the EXLST macro, VSE/VSAM takes one of the following actions:

- When the RPL is in use (return code X'04'), retry the request.

- If the request is completed with a logic error (return code X'08') other than end-of-file, your LERAD exit routine is entered if you specified the LERAD exit in the EXLST and if it is active. If no LERAD exit routine is specified or if it is inactive, control is returned to the instruction following the request macro that raised the logic error condition with return code X'08' set.

  When you reach end-of-file, your request completes with a logic error (return code X'08' and error code X'04') and your EODAD exit routine is entered. If you have no EODAD exit routine or if it is inactive, your LERAD exit routine is entered. If no LERAD exit routine is specified or if it is inactive, control is returned to the instruction following the request macro that raised the end-of-file condition with return code X'08' set. Note, too, that if the EODAD exit is taken, the LERAD exit is not.
- If the request completed with an I/O (physical) error (return code X'0C'), your SYNAD exit routine is entered if you specified the SYNAD exit in the EXLST and if it is active. If no SYNAD exit routine is specified or if it is inactive, control is returned to the instruction following the request macro that raised the I/O error condition with return code X'0C' set.

After your EODAD, LERAD, or SYNAD exit returns to VSE/VSAM, VSE/VSAM returns control to the instruction following the request macro that raised the error condition with a non-zero return code set.

The feedback field in the RPL (FDBK operand in SHOWCB and TESTCB) is a three-byte field with the following format:

```
0000xx
```

where:

xx is an error code that describes the error or, if the return code is zero, additional information about the request.

Besides the return code (set in Register 15) and the error code (which you may obtain by specifying FDBK in the SHOWCB macro) a function code is provided for alternate-index processing. This function code is set on logical or physical errors detected by VSE/VSAM and indicates whether the respective error condition occurred during accessing the base cluster or the alternate index. In addition, the function code indicates whether or not the upgrade set is still correct after the request that failed. The function codes and their meanings are:

Function Code
Meaning

X'00' Condition occurred during accessing the base cluster. Upgrade set is correct.

X'01' Condition occurred during accessing the base cluster. Upgrade set may be incorrect as a consequence of this request.

X'02' Condition occurred during accessing the AIX over a base cluster. Upgrade set is correct.

X'03' Condition occurred during accessing the AIX over a base cluster. Upgrade set may be incorrect as a consequence of this request.

X'04' Condition occurred during upgrade processing. Upgrade set is correct.

X'05' Condition occurred during upgrade processing. Upgrade set may be incorrect as a result as a consequence of this request.

You can display or test the function code by specifying the keyword FTNCD in the SHOWCB or TESTCB macro, respectively.

Return Codes from the Control Block Manipulation Macros

When VSE/VSAM returns to your processing program after a GENCB, MODCB, SHOWCB, or TESTCB request, Register 15 contains one of the following return codes:

Return Code	Meaning
X'00'	Operation successfully completed.
X'04'	An error occurred.
X'08'	The execute form of the macro was used in an attempt to change a non-existent entry in the referenced parameter list.
X'0C'	Request was not executed because an error was encountered while VSE/VSAM routines were loaded.

If Register 15 contains X'04', an error code is set in Register 0, which indicates the type of error. Make sure that, before issuing the macro, you save the contents of Register 0 if you want to use its contents later on. For an explanation of the error codes, see the "VSE/VSAM return and error codes" in the *z/VSE Messages and Codes*.

List, Execute, and Generate Forms of the Control Block Manipulation Macros

The list and execute forms of the control block manipulation macros (GENCB, MODCB, SHOWCB, and TESTCB) allow you to save virtual storage by using one parameter list for two or more macros. You can also make your program reenterable, that is, executable by more than one task at a time. While the generate form of the macros enables you to make programs reenterable it does not allow shared parameter lists.

List and Execute Forms

The list form of GENCB, MODCB, SHOWCB, and TESTCB has the same parameters as the standard form, except that it includes the parameter MF=L or MF=(L,adress...).

The parameter list of the macro is created inline when MF=L is coded. This version is not reenterable and register notation cannot be used for macro parameter addresses.

When MF=(L,address...) is coded, the parameter list of the macro is created in the area specified by address. This form is reenterable. You must supply the area by a GETVIS macro when your program is executed. You can determine the size of the parameter list by coding the third operand label. VSE/VSAM equates label to the length of the list.

The execute form produces the executable code of the macros. The execute form is also identical to the standard form, except that it includes the operand MF=(E,address), where address points to the parameter list created by the list form of the macro. All of the other operands of the macro are optional and are coded

only to change entries in the parameter list before the list is used. However, you cannot use the execute form to add or delete entries from the parameter list or to change the type of list.

Generate Form

The generate form of the macros allows you to make your program reenterable, but it does not create shared parameter lists. The generate form is the same as the standard form, except that you code MF=(G,address...). The parameter list is created in an area pointed to by address. To ensure that the parameter list is reenterable, address should be coded in register notation. You must obtain this area by a GETVIS macro when the program is executed. You can determine the size of the parameter list by coding the third operand label. VSE/VSAM equates label to the length of the list.

Examples of the List, Execute, and Generate Forms

Figure 63 and Figure 64 show the use of the list, execute, and generate forms of the control block manipulation macros.

In Figure 63, MODCB is used to place the length of a record in the RPL before the record is written. The list and execute forms are used so that only one parameter list is created (though the macro is issued several times). This list form is not reenterable.

In Figure 64, the generate form is used to create an ACB. It is reenterable because both the ACB itself and the parameter list of the GENCB macro are created in areas obtained through a GETVIS macro.

Continuation characters required in column 72 are not shown in the examples.

```
        MODCB  MF=(E,LENMOD),RECLEN=(7)      Current length in register 7
        LTR    15,15                        MODCB successful?
        BNZ    MODERR                       No, go to error routine
        PUT    RPL=LIST                     Yes, write record
        .
        .
        .
        MODCB  MF=(E,LENMOD)                Length is 100 bytes
        LTR    15,15                        MODCB successful?
        BNZ    MODERR                       No, go to error routine
        PUT    RPL=LIST                     Yes, write record
        .
        .
        .
LENMOD MODCB  RPL=LIST,RECLEN=100,MF=L      List form has default
```

Figure 63. Examples of the List and Execute Form

```
        LA     10,PARMLEN                   Load length for GETVIS
        GETVIS ADDRESS=(8),LENGTH=(10)      Get area for parm, list
        LTR    15,15                        GETVIS successful?
        BNZ    VISERR                       No, go to error routine
        GENCB  BLK=ACB,MF=(G,(8),PARMLEN),
                EXLST=(3),BUFND=4,BUFNI=3,
                DDNAME=VFILENM,
                MACRF=(KEY,SEQ,DIR,OUT),
                PASSWD=PASS
        LTR    15,15                        GENCB successful?
        BNZ   GENERR                        No, go to error routine
        LR    2,1                           Yes, save ACB address
        .
        .
        .
PASS    DC    FL1'6',C'CHANGE'
```

Figure 64. Example of the Generate Form

Appendix A. Operand Notation and Parameter Lists for VSE/VSAM Macros

This Appendix...

- Documents Programming Interface information. For a definition of this category of interface information refer to "Notices" on page xiii.
- Lists the macro operands and parameter lists for VSE/VSAM macros.

Operand Notation for VSE/VSAM Macros

The addresses, names, numbers, and options required with operands in GENCB, MODCB, SHOWCB, TESTCB, BLDVRP, WRTBFR, and SHOWCAT can be expressed in a variety of ways:

- An absolute numeric expression, for example, RECLEN=400, as in the following sample job stream:

```
         LA     1,RPL                Set RPL address in register 1.
         MODCB  RPL=(1),RECLEN=400   Set record length field in
*                                    RPL to value of 400.
```

- A character string, for example, DDNAME=DATASET
- A code or a list of codes separated by commas and enclosed in parentheses, for example, OPTCD=KEY or OPTCD=(KEY,DIR,IN)
- A register from 2 through 12 that contains an address or numeric value. Equated labels can be used to designate a register, for example, SYNAD=(ERR), where the following equate statement has been included in the program: ERR EQU 3

  Example of register notation for an operand taking numeric value:

```
         LA     6,400               Set length desired in register 6.
         MODCB  RPL=RPL,RECLEN=(6)  Set record length field in RPL
*                                   to value specified in register 6.
```

  Example of register notation for an operand that takes an address value:

```
         LA     2,RCDAREA           Set address of record area in
*                                   register 2.
         MODCB  RPL=RPL,AREA=(2)    Set area operand in RPL according
*                                   to contents of register 2.
         .
         .
         .
RCDAREA  DS  CL400
```

- An expression of the form (S,scon), where scon is any expression valid for an S-type address constant, including the base-displacement form.

  The use of the S-type notation for numeric-value operands is usually equivalent to either absolute-numeric-expression notation or register notation (see the following example).

  Example of S-type address notation for an operand that takes a numeric value:

© Copyright IBM Corp. 1979, 2005

```
         MODCB  RPL=RPL,RECLEN=(S,400)    Set record length field
*                                         in RPL to value of 400.
```

Example of S-type address notation for an operand that takes an address value:

```
         MODCB  RPL=RPL,AREA=(S,RCDAREA1)   Set area operand in RPL
*                                           to address of RCDAREA1.
         .
         .
         .
RCDAREA1 DS  CL400
```

- An expression of the form (*,scon); where scon is any expression valid for an S-type address constant, including the base-displacement form. The address specified by scon is indirect, that is, it points to the location that contains the value of the keyword.

  If indirect S-type address notation is used, the value it points to must meet either of the following criteria:

  – If the value is a numeric quantity, a bit string representing codes, or a pointer, it must occupy a fullword of storage.
  – If the value is an alphameric character string, it must occupy two words of storage, be left aligned, and be padded on the right with blanks, if necessary.

  Example of indirect S-type address notation for an operand that takes a numeric value:

```
         MODCB  RPL=RPL,RECLEN=(*,RECLEN1)  Set record length field
*                                           in RPL to value specified
*                                           in the fullword RECLEN1.
         .
         .
         .
RECLEN1  DC  F'400'
```

  Example of indirect S-type address notation for an operand that takes a name value:

```
         MODCB  ACB=ACB,DDNAME=(*,DDNAME1)  Set ddname field in ACB
*                                           to value specified in the
*                                           8-byte field DDNAME1.
         .
         .
         .
DDNAME1  DC  CL8'FILENAME'
```

  Example of indirect S-type notation for an operand that takes an address value:

```
          MODCB RPL=RPL,AREA=(*,ARCDAREA)  Set area operand in RPL
*                                          to the address pointed to
*                                          by the pointer ARCDAREA.
          .
          .
          .
ARCDAREA  DC  A(RCDAREA1)
```

  Example of an expression valid for a relocatable A-type address constant:

```
          MODCB RPL=RPL,AREA=RCDAREA        Set area operand in RPL
*                                           to address of RCDAREA.
```

The expressions that can be used depend on the keyword specified. Register and S-type address notations cannot be used when MF=L is specified.

GENCB Macro Operands

GENCB Keywords

Keyword	Absolute Numeric	Code	Character String	Register	Address		
					S-Type	Indirect S-Type	A-Type
AM	-	x	-	-	-	-	-
BLK	-	x	-	-	-	-	-
COPIES	x	-	-	x	x	x	-
LENGTH	x	-	-	x	x	x	-
WAREA	-	-	-	x	x	x	x

ACB Keywords (BLK=ACB)

Keyword	Absolute Numeric	Code	Character String	Register	Address		
					S-Type	Indirect S-Type	A-Type
BSTRNO	x	-	-	x	x	x	-
BUFND	x	-	-	x	x	x	-
BUFNI	x	-	-	x	x	x	-
BUFSP	x	-	-	x	x	x	-
DDNAME	-	-	x	-	-	x	-
EXLST	-	-	-	x	x	x	x
MACRF	-	x	-	-	-	-	-
MAREA	-	-	-	x	x	x	x
MLEN	x	-	-	x	x	x	-
PARMS=CLOSDSP	-	x	-	-	-	-	-
PASSWD	-	-	-	x	x	x	x
RMODE31	-	x	-	-	-	-	-
SHRPOOL	x	-	-	x	x	x	-
STRNO	x	-	-	x	x	x	-

EXLST Keywords (BLK=EXLST)

Keyword	Absolute Numeric	Code	Character String	Register	Address		
					S-Type	Indirect S-Type	A-Type
EODAD	-	-	-	x	x	x	x
EXCPAD	-	-	-	x	x	x	x
JRNAD	-	-	-	x	x	x	x
LERAD	-	-	-	x	x	x	x
SYNAD	-	-	-	x	x	x	x

RPL Keywords (BLK=RPL)

Keyword	Absolute Numeric	Code	Character String	Register	Address		
					S-Type	Indirect S-Type	A-Type
ACB	-	-	-	x	x	x	x
AREA	-	-	-	x	x	x	x
AREALEN	x	-	-	x	x	x	-
ARG	-	-	-	x	x	x	x
KEYLEN	x	-	-	x	x	x	-
NXTRPL	-	-	-	x	x	x	x
OPTCD	-	x	-	-	-	-	-
RECLEN	x	-	-	x	x	x	-
TRANSID	x	-	-	x	x	x	-

MODCB Macro Operands

MODCB Keywords

Keyword	Absolute Numeric	Code	Character String	Register	Address		
					S-Type	Indirect S-Type	A-Type
ACB	-	-	-	x	x	x	x
AM	-	x	-	-	-	-	-
EXLST	-	-	-	x	x	x	x
RPL	-	-	-	x	x	x	x

ACB Keywords

Keyword	Absolute Numeric	Code	Character String	Register	Address		
					S-Type	Indirect S-Type	A-Type
BSTRNO	x	-	-	x	x	x	-
BUFND	x	-	-	x	x	x	-
BUFND	x	-	-	x	x	x	-
BUFNI	x	-	-	x	x	x	-
BUFSP	x	-	-	x	x	x	-
DDNAME	-	-	x	-	-	x	-
EXLST	-	-	-	x	x	x	x
MACRF	-	x	-	-	-	-	-
MAREA	-	-	-	x	x	x	x
MLEN	x	-	-	x	x	x	-
PARMS=CLOSDSP	-	x	-	-	-	-	-
PASSWD	-	-	-	x	x	x	x
RMODE31	-	x	-	-	-	-	-
SHRPOOL	x	-	-	x	x	x	-
STRNO	x	-	-	x	x	x	-

EXLST Keywords

Keyword	Absolute Numeric	Code	Character String	Register	Address		
					S-Type	Indirect S-Type	A-Type
EODAD	-	-	-	x	x	x	x
EXCPAD	-	-	-	x	x	x	x
JRNAD	-	-	-	x	x	x	x
LERAD	-	-	-	x	x	x	x
SYNAD	-	-	-	x	x	x	x

RPL Keywords

Keyword	Absolute Numeric	Code	Character String	Register	Address		
					S-Type	Indirect S-Type	A-Type
ACB	-	-	-	x	x	x	x
AREA	-	-	-	x	x	x	x
AREALEN	x	-	-	x	x	x	-
ARG	-	-	-	x	x	x	x
KEYLEN	x	-	-	x	x	x	-
NXTRPL	-	-	-	x	x	x	x
OPTCD	-	x	-	-	-	-	-
RECLEN	x	-	-	x	x	x	-
TRANSID	x	-	-	x	x	x	-

SHOWCB Macro Operands

SHOWCB Keywords

Keyword	Absolute Numeric	Code	Character String	Register	Address		
					S-Type	Indirect S-Type	A-Type
ACB	-	-	-	x	x	x	x
AM	-	x	-	-	-	-	-
AREA	-	-	-	x	x	x	x
EXLST	-	-	-	x	x	x	x
FIELDS*	-	x	-	-	-	-	-
LENGTH	x	-	-	x	x	x	-
OBJECT	-	x	-	-	-	-	-
RPL	-	-	-	x	x	x	x

* For a list of the operands you can specify in the FIELDS parameter, see "The SHOWCB Parameter List" on page 308.

TESTCB Macro Operands

TESTCB Keywords

Keyword	Absolute Numeric	Code	Character String	Register	Address		
					S-Type	Indirect S-Type	A-Type
ACB	-	-	-	x	x	x	x
AM	-	x	-	-	-	-	-
ERET	-	-	-	x	x	x	x
EXLST	-	-	-	x	x	x	x
OBJECT	-	x	-	-	-	-	-
RPL	-	-	-	x	x	x	x

ACB Keywords

Keyword	Absolute Numeric	Code	Character String	Register	Address		
					S-Type	Indirect S-Type	A-Type
ACBLEN	x	-	-	x	x	x	-
ATRB	-	x	-	-	-	-	-
AVSPAC	x	-	-	x	x	x	-
BSTRNO	x	-	-	x	x	x	-
BUFND	x	-	-	x	x	x	-
BUFNI	x	-	-	x	x	x	-
BUFNO	x	-	-	x	x	x	-
BUFSP	x	-	-	x	x	x	-
CINCV	x	-	-	x	x	x	-
DDNAME	-	-	x	-	-	-	-
ERROR	x	-	-	x	x	x	-
EXLST	-	-	-	x	x	x	x
FS	x	-	-	x	x	x	-
KEYLEN	x	-	-	x	x	x	-
LRECL	x	-	-	x	x	x	-
MACRF	-	x	-	-	-	-	-
MAREA	-	-	-	x	x	x	x
MLEN	x	-	-	x	x	x	-
NCIS	x	-	-	x	x	x	-
NDELR	x	-	-	x	x	x	-
NEXCP	x	-	-	x	x	x	-
NEXT	x	-	-	x	x	x	-
NINSR	x	-	-	x	x	x	-
NIXL	x	-	-	x	x	x	-
NLOGR	x	-	-	x	x	x	-
NRETR	x	-	-	x	x	x	-
NSSS	x	-	-	x	x	x	-
NUPDR	x	-	-	x	x	x	-
OFLAGS	-	x	-	-	-	-	-
OPENOBJ	-	x	-	-	-	-	-
PARMS=CLOSDSP	-	x	-	-	-	-	-
PASSWD	-	-	-	x	x	x	x
RKP	x	-	-	x	x	x	-
SHRPOOL	x	-	-	x	x	x	-
STMST	-	-	-	-	-	x	-
STRNO	x	-	-	x	x	x	-

EXLST Keywords

Keyword	Absolute Numeric	Code	Character String	Register	Address		
					S-Type	Indirect S-Type	A-Type
EODAD	-	-	-	x	x	x	x
EXCPAD	-	-	-	x	x	x	x
EXLLEN	x	-	-	x	x	x	-
JRNAD	-	-	-	x	x	x	x
LERAD	-	-	-	x	x	x	x
SYNAD	-	-	-	x	x	x	x

RPL Keywords

Keyword	Absolute Numeric	Code	Character String	Register	Address		
					S-Type	Indirect S-Type	A-Type
ACB	-	-	-	x	x	x	x
AIXFLAG	x	-	-	-	-	-	-
AIXPC	x	-	-	x	x	x	-
AREA	-	-	-	x	x	x	x
AREALEN	x	-	-	x	x	x	-
ARG	-	-	-	x	x	x	x
FDBK	x	-	-	x	x	x	-
FTNCD	x	-	-	x	x	x	-
KEYLEN	x	-	-	x	x	x	-
NXTRPL	-	-	-	x	x	x	x
OPTCD	-	x	-	-	-	-	-
RBA	x	-	-	x	x	x	-
RECLEN	x	-	-	x	x	x	-
RPLLEN	x	-	-	x	x	x	-
TRANSID	x	-	-	x	x	x	-

BLDVRP Macro Operands

Keyword	Absolute Numeric	Code	Character String	Register	Address		
					S-Type	Indirect S-Type	A-Type
BUFFERS	x	-	-	x	-	-	-
KEYLEN	x	-	-	x	-	-	-
RMODE31		x	-	-	-	-	-
SHRPOOL	x	-	-	x	-	-	-
STRNO	x	-	-	x	-	-	-
TYPE	-	x	-	-	-	-	-

DLVRP Macro Operands

Keyword	Absolute Numeric	Code	Character String	Register	Address		
					S-Type	Indirect S-Type	A-Type
SHRPOOL	X	-	-	X	-	-	-
TYPE	-	X	-	-	-	-	-

SHOWCAT Macro Operands

Keyword	Absolute Numeric	Code	Character String	Register	Address S-Type	Address Indirect S-Type	Address A-Type
ACB	-	-	-	x	-	-	x
AREA	-	-	-	x	-	-	x
CI	-	-	-	x	-	-	x
CATDSN	-	-	-	x	-	-	x
CATFIL	-	-	-	x	-	-	x
DDNAME	-	-	-	x	-	-	x
NAME	-	-	-	x	-	-	x

WRTBFR Macro Operands

Keyword	Absolute Numeric	Code	Character String	Register	Address S-Type	Address Indirect S-Type	Address A-Type
percent	x	-	-	x	-	-	-
RPL	-	-	-	x	-	-	x
TYPE=ALL	-	x	-	-	-	-	-
TYPE=DS	-	x	-	-	-	-	-
TYPE=LRU	-	x	-	-	-	-	-
TYPE=TRN	-	x	-	-	-	-	-

Parameter Lists for VSE/VSAM Macros

The VSE/VSAM control block (CB) manipulation macros (GENCB, MODCB, SHOWCB, and TESTCB) use an internal parameter list to describe the actions that you specify when you code the macros. The BLDVRP macro (for building a VSE/VSAM resource pool) and the SHOWCAT macro (which displays catalog information) also use an internal parameter list to indicate the addresses and values that you specify when you code the macros.

Depending on the *form* of the macro, the *internal parameter list* is built as follows:

- The *standard* form of these macros builds a parameter list in-line and processes it.
- The *list* form builds a parameter list in an area you specified.
- The *execute* form processes a previously built parameter list.
- The *generate* form (not for BLDVRP and SHOWCAT) builds a parameter list in an area you specify and also processes it.

(Use of the different forms are discussed under “List, Execute, and Generate Forms of the Control Block Manipulation Macros” on page 291.)

For special purposes, such as developing high-level programming languages, you may want to build and process parameter lists without using the macros. The following describes the format of the parameter lists and gives the codes used for the operands of each of the macros. The formats and codes are fixed, so that you can build and alter them by your own methods. For the control block manipulation macros, a parameter list contains a variable number of entries of three types:

Type 1: At the beginning of the list, *addresses* of entries of Type 2 and Type 3. The addresses are fullwords, and the high-order bit of the last fullword is 1.
Type 2: A *header* entry containing general information about the block or list that you want to generate, modify, display or test.
Type 3: At the end of the list, *keyword entries* describing each field that you want to generate, modify, display, or test.

In the following, entries of *Type 2* and *Type 3* are described separately for GENCB, MODCB, SHOWCB, and TESTCB. When VSE/VSAM receives control, register 1 must point to your parameter list.

The format of the BLDVRP and SHOWCAT parameter lists is different from the above scheme. Refer to "The BLDVRP Parameter List" on page 312 and "The SHOWCAT Parameter List" on page 313.

The GENCB Parameter List

Header Entry

Offset Dec (HEX)	0	1	2
0 (0)	Block or list See "(1) list"	X'01'	Number of copies
4 (4)	Address of the area you are providing, or zeros		
8 (8)	Length of the area, or zeros		(reserved)

(1) list explanation:

X'A0' indicates ACB
X'B0' indicates EXLST
X'C0' indicates RPL

Keyword Entries

The parameter list for GENCB contains no keyword entries if you are generating a default ACB, EXLST, or RPL.

Offset Dec (HEX)	0	2
0 (0)	Keyword code See "(1) keyword code"	(reserved)
4 (4)	(value|address|option|name) of the keyword See "(2) option"	
8 (8)	(Required for some keywords) See "(3) keywords"	

(1) keyword code explanation: Defined by AL2(value).

For an ACB		For an EXLST		For an RPL	
Keyword	Value	Keyword	Value	Keyword	Value
BUFND	4	EODAD	37	ACB	43
BUFNI	5	EXCPAD	38	AREA	44
BUFSP	7	JRNAD	39	AREALEN	45
DDNAME	9	LERAD	40	ARG	46
EXLST	12	SYNAD	41	KEYLEN	48
MAREA	14			NXTRPL	51
MLEN	15			OPTCD	52
MACRF	18			RECLEN	53
PASSWD	30			TRANSID	95
STRNO	32				
BSTRNO	36				
SHRPOOL	129				
CLOSDSP	151				
MACRF3	163				

(2) option explanation: Indicates the options for MACRF, MACRF3, OPTCD, and CLOSDSP with a 1 in a bit of the fullword:

MACRF Option	Bit	MACRF3 Option	Bit	OPTCD Option	Bit	CLOSDSP Option	Bit
KEY	0	DDN	0	KEY	0		
ADR	1	DSN	1	ADR	1	First CLOSDSP Option:	
CNV	2	NCM	8	CNV	2		
SEQ	3	CMP	9	SEQ	3		
SKP	4	RMODE31	15	DIR	4		
DIR	5			SKP	5	KEEP	0
IN	6			NUP	8	DELETE	1
OUT	7			UPD	9	DATE	2
NUB	8			NSP	10		
UBF	9			KEQ	11		
NRM	15			KGE	12		
AIX	16			FKS	13		
NSR	17			GEN	14	Second CLOSDSP Option:	
LSR	18			MVE	15		
NDF	22			LOC	16		
DFR	23			FWD	17		
RST	28			BWD	18	KEEP	3
NRS	29			ARD	19	DELETE	4
NFY	30			LRD	20		
VFY	31						

(3) keywords explanation: The third fullword is required for the ACB operand DDNAME, and for all of the EXLST operands, for which the third fullword indicates A, N, and L:

Bit	Meaning when Set to 1
0	Address is active (A).
1	Address is not active (N).
2	Address is of a field containing the name of an exit routine to be loaded (L).
3	Address is specified in the preceding fullword of this entry.
4-31	Unused.

The MODCB Parameter List

Header Entry

Offset Dec (HEX)	0	1	2
0 (0)	Block or list See "(1) list"	X'02'	(reserved)
4 (4)	Address of the block or list to be modified		

(1) list explanation:

X'A0' indicates ACB
X'B0' indicates EXLST
X'C0' indicates RPL

Keyword Entries

Offset Dec (HEX)	0	2
0 (0)	Keyword code See "(1) keyword code"	(reserved)
4 (4)	(value|address|option|name) of the keyword See "(2) option"	
8 (8)	(Required for some keywords) See "(3) keywords"	

(1) keyword code explanation: Defined by AL2(value):

For an ACB		For an EXLST		For an RPL	
Keyword	Value	Keyword	Value	Keyword	Value
BUFND	4	EODAD	37	ACB	43
BUFNI	5	EXCPAD	38	AREA	44
BUFSP	7	JRNAD	39	AREALEN	45
DDNAME	9	LERAD	40	ARG	46
EXLST	12	SYNAD	41	KEYLEN	48
MAREA	14			NXTRPL	51
MLEN	15			OPTCD	52
MACRF	18			RECLEN	53
PASSWD	30			TRANSID	95
STRNO	32				
BSTRNO	36				
SHRPOOL	129				
CLOSDSP	151				
MACRF3	163				

(2) option explanation: Indicates the options for MACRF, MACRF3, OPTCD, and CLOSDSP with a 1 in a bit of the fullword:

MACRF Option	Bit	MACRF3 Option	Bit	OPTCD Option	Bit	CLOSDSP Option	Bit
KEY	0	DDN	0	KEY	0		
ADR	1	DSN	1	ADR	1	First CLOSDSP Option:	
CNV	2	NCM	8	CNV	2		
SEQ	3	CMP	9	SEQ	3		
SKP	4	RMODE31	15	DIR	4		
DIR	5			SKP	5	KEEP	0
IN	6			NUP	8	DELETE	1
OUT	7			UPD	9	DATE	2
NUB	8			NSP	10		
UBF	9			KEQ	11		
NRM	15			KGE	12		
AIX	16			FKS	13		
NSR	17			GEN	14	Second CLOSDSP Option:	
LSR	18			MVE	15		
NDF	22			LOC	16		
DFR	23			FWD	17		
RST	28			BWD	18	KEEP	3
NRS	29			ARD	19	DELETE	4
NFY	30			LRD	20		
VFY	31						

With the MODCB macro, there are no defaults for these options. When you code a bit for the OPTCD operand, the contrary bit that was previously set is turned off. For example, if KEY was previously set, and you set ADR, KEY is turned off, because a request parameter list can be set for only one type of access.

(3) keywords explanation: The third fullword is required for the ACB operand DDNAME, and for all of the EXLST operands, for which the third fullword indicates A, N, and L:

Bit	Meaning when Set to 1
0	Address is active (A).
1	Address is not active (N).
2	Address is of a field containing the name of an exit routine to be loaded (L).
3	Address is specified in the preceding fullword of this entry.
4-31	Unused.

The SHOWCB Parameter List

Header Entry

Offset Dec (HEX)	0	1	2
0 (0)	Block or list See "(1) list"	X'03'	Type of object to be displayed See "(2) displayed"
4 (4)	Address of the block or list to be displayed		
	Address of the display area you are providing		
8 (8)	Length of the display area		(reserved)

(1) list explanation:

X'00' indicates that no block or list is specified
to display the standard length of the block(s)
or list(s) specified by the keywords
ACBLEN, EXLLEN, or RPLLEN
X'A0' indicates ACB
X'B0' indicates EXLST
X'C0' indicates RPL

(2) displayed:

AL2(0) indicates the data of a file
AL2(1) indicates the index of a file

Keyword Entries

Offset Dec (HEX)	0	2
0 (0)	Keyword code See "(1) keyword code"	(reserved)

(1) keyword code explanation: Defined by AL2(value):

For an ACB				For an EXLST		For an RPL	
Keyword	Value	Keyword	Value	Keyword	Value	Keyword	Value
AVSPAC	2	NEXT	22	EODAD	37	ACB	43
ACBLEN	3	NINSR	23	EXCPAD	38	AREA	44
BUFND	4	NIXL	24	JRNAD	39	AREALEN	45
BUFNI	5	NLOGR	25	LERAD	40	ARG	46
BUFNO	6	NRETR	26	SYNAD	41	KEYLEN	48
BUFSP	7	NSSS	27	EXLLEN	42	NXTRPL	51
CINV	8	NUPDR	28			RECLEN	53
DDNAME	9	PASSWD	30			RPLLEN	55
ENDRBA	10	RKP	31			FDBK	56
ERROR	11	STRNO	32			FDBK	56
EXLST	12	STMSG	35			RBA	57
FS	13	BSTRNO	36			AIXPC	58
MAREA	14	BFRFND	124			TRANSID	95
MLEN	15	BFRNDS	125			FTNCD	99
KEYLEN	16	NUIW	126				
LRECL	17	UIW	127				
NCIS	19	STRMAX	128				
NDELR	20	SHRPOOL	129				
NEXCP	21	HALCRBA	148				

The TESTCB Parameter List

Header Entry

Offset Dec (HEX)	0	1	2
0 (0)	Block or list See "(1) list"	X'04'	Type of object to be tested See "(2) tested"
4 (4)	Address of the block or list to be tested		
8 (8)	Address or the routine to return to from unequal comparisons, or zeros		
12 (C)	(reserved)		

(1) list explanation:

X'00' indicates that no block or list is specified
to test the standard length of the block(s)
or list(s) specified by the keywords
ACBLEN, EXLLEN, or RPLLEN
X'A0' indicates ACB
X'B0' indicates EXLST
X'C0' indicates RPL

(2) tested explanation:

AL2(0) indicates the data of a file
AL2(1) indicates the index of a file

Keyword Entries

Offset Dec (HEX)	0	2 -->
0 (0)	Keyword code See "(1) keyword code"	(reserved)
4 (4)	(value|address|option|name) of the keyword See "(2) option" See "(3) code"	
8 (8)	(Required for some keywords) See "(4) keywords"	

(1) keyword code explanation: Defined by AL2(value):

For an ACB				For an EXLST		For an RPL	
Keyword	Value	Keyword	Value	Keyword	Value	Keyword	Value
ATRB	1*	NEXT	22	EODAD	37	ACB	43
ATRB	162**	NINSR	23	EXCPAD	38	AREA	44
AVSPAC	2	NIXL	24	JRNAD	39	AREALEN	45
ACBLEN	3	NLOGR	25	LERAD	40	ARG	46
BUFND	4	NRETR	26	SYNAD	41	KEYLEN	48
BUFNI	5	NSSS	27	EXLLEN	42	NXTRPL	51
BUFNO	6	NUPDR	28			OPTCD	52
BUFSP	7	OFLAGS	29			RECLEN	53
CINV	8	PASSWD	30			RPLLEN	55
DDNAME	9	RKP	31			FDBK	56
ERROR	11	STRNO	32			RBA	57
EXLST	12	OPENOBJ	33			AIXPC	58
FS	13	STMST	35			AIXFLAG	59
MAREA	14	BSTRNO	36			FTNCD	99
MLEN	15	SHRPOOL	129				
KEYLEN	16	CLOSDSP	151				
LRECL	17	MACRF3	163				
MACRF	18						
NCIS	19						
NDELR	20						
NEXCP	21						

*) Option bits 0 to 7 only (see (2)).

**) Option bits 8 to 15 only (see (2)).

(2) option explanation: Indicate the options for MACRF, MACRF3 OPTCD, CLOSDSP, ATRB, OFLAGS, AIXFLAG, and OPENOBJ: with a 1 in a bit of the fullword:

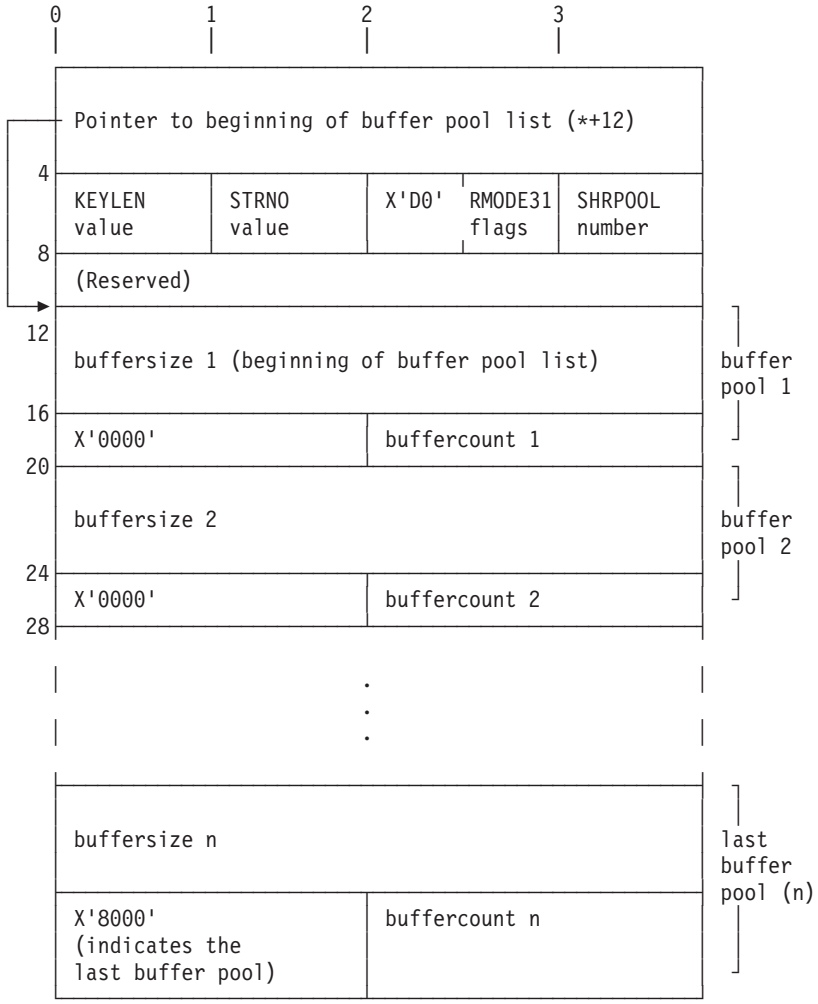
MACRF Option	Bit	MACRF3 Option	Bit	OPTCD Option	Bit	CLOSDSP Option	Bit
KEY	0	DDN	0	KEY	0		
ADR	1	DSN	1	ADR	1	First CLOSDSP Option:	
CNV	2	NCM	8	CNV	2		
SEQ	3	CMP	9	SEQ	3		
SKP	4	RMODE31	15	DIR	4		
DIR	5			SKP	5	KEEP	0
IN	6			NUP	8	DELETE	1
OUT	7			UPD	9	DATE	2
NUB	8			NSP	10		
UBF	9			KEQ	11		
NRM	15			KGE	12		
AIX	16			FKS	13		
NSR	17			GEN	14	Second CLOSDSP Option:	
LSR	18			MVE	15		
NDF	22			LOC	16		
DFR	23			FWD	17		
RST	28			BWD	18	KEEP	3
NRS	29			ARD	19	DELETE	4
NFY	30			LRD	20		
VFY	31						

ATRB Option	Bit	OFLAGS Option	Bit	AIXFLAG Option	Bit	OPENOBJ Option	Bit
KSDS	0	OPEN	0	AIXPKP	0	AIX	0
ESDS	1					PATH	1
WCK	2					BASE	2
SSWD	3						
REPL	4						
RRDS	5						
SPAN	6						
UNQ	7						
COMP	8						
XLKSDS	11						
VRDS	14						

(3) code explanation: The codes for ERROR and for FDBK are documented with the appropriate macro instructions. **(4) keywords** explanation: The third fullword is required for the ACB operands DDNAME and STMST, and for all of the EXLST operands, for which the third fullword indicates A, N, and L:

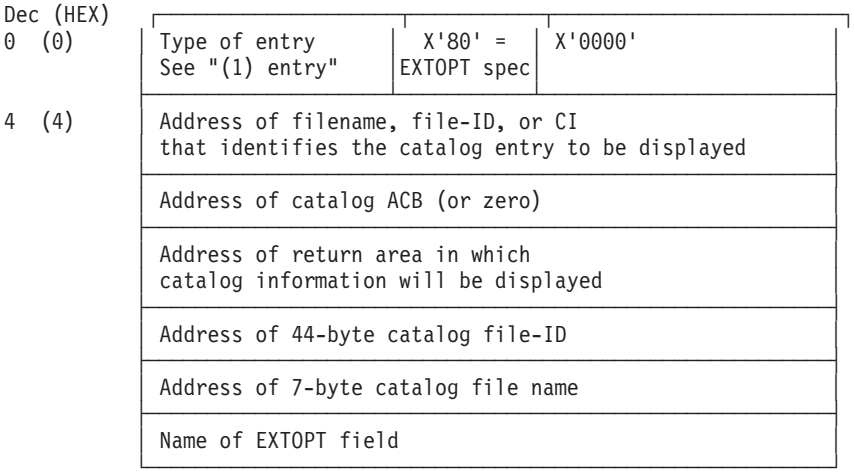
Bit	Meaning when Set to 1
0	Address is active (A).
1	Address is not active (N).
2	Address is of a field containing the name of an exit routine to be loaded (L).
3	Address is specified in the preceding fullword of this entry.
4-31	Unused.

The BLDVRP Parameter List



The SHOWCAT Parameter List

Header Entry



(1) entry explanation:

X'80' = The field at offset X'04' points to a 44-byte file ID
X'40' = The field at offset 'X04' points to a 7-byte file name
X'00' = The field at offset X'04' points to a 3-byte CI number

Appendix B. Invoking IDCAMS from a Program

This Appendix...

Shows how IDCAMS can be invoked by a program through the use of the CDLOAD macro instruction.

Describes how the dynamic invocation of IDCAMS enables re-specification of selected processing defaults as well as the ability to manage input/output operations for selected files.

Invoking Macro Instructions

IDCAMS may be invoked from a program by loading the root segment of IDCAMS into virtual storage and then doing a branch entry to the module. To load IDCAMS, the program should be invoked with the SIZE=AUTO parameter on the EXEC statement, and should issue a CDLOAD macro of the form:

```
CDLOAD | address
```

```
where: address
       specifies the address of an 8-byte,
       left justified character string 'IDCAMS'.
```

CDLOAD returns the starting address of the module in Register 1.

The invoking program should branch to the address returned by CDLOAD plus 6.

Because IDCAMS uses MVS linkage conventions, the invoking program must provide an 18 fullword area to be used as a save area by IDCAMS. On entry to IDCAMS, Register:

- 1 should point to the argument list described in Figure 65 on page 316.
- 13 should point to the save area.
- 14 must contain the return address.
- 15 must contain the address of IDCAMS plus 6.

On return, all registers except Register 15 are restored by IDCAMS. Register 15 contains the final return code from the processor. The following table contains the possible values of Register 15:

Code	Meaning
0	The function was executed as directed and expected. Informational messages may have been issued.
4	Some disturbance in executing the complete function was met, but it was possible to continue. The results might not be exactly what the user wants, but no permanent harm appears to have been done by continuing. A warning message was issued.
8	A function could not perform all that was asked of it. The function was completed, but specific details were bypassed.
12	The entire function could not be performed.

© Copyright IBM Corp. 1979, 2005

16 Severe problem encountered. Remainder of command stream is not flushed but processor returns code 16 to the system.

Figure 65 describes the argument list as it exists in the user's area that is passed to the IDCAMS processor.

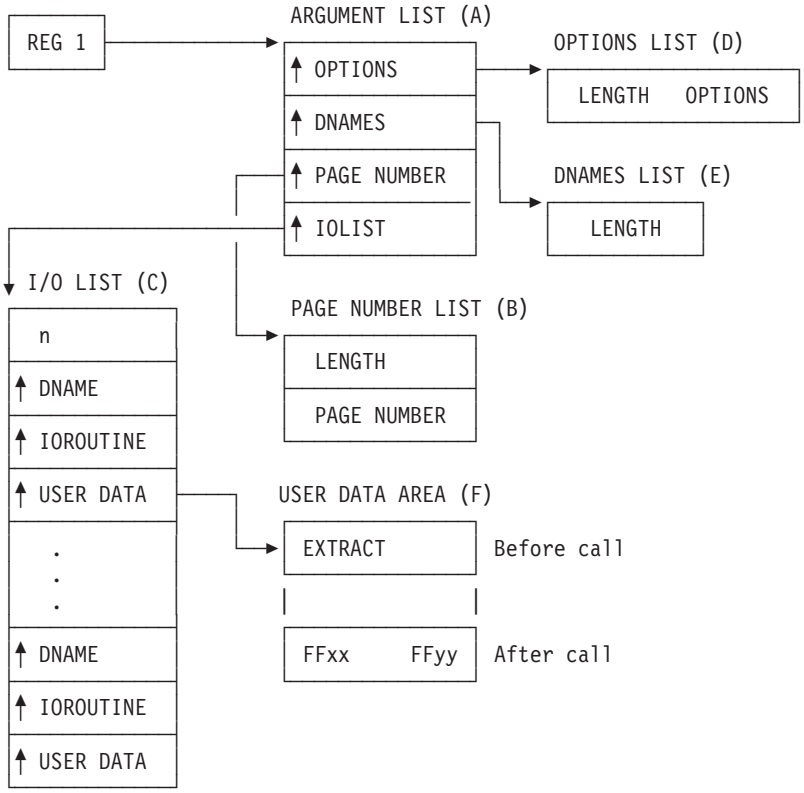


Figure 65. Processor Invocation Argument List from a Program

Explanation

The following explains Figure 65.

(A) The Argument List

A maximum of four fullword addresses pointing to the various arguments. The high-order bit of the last address must be set to one. Any argument you do not wish to specify that precedes an argument you are specifying must be an address pointing to a half word of binary zeros. If you do not specify IOLIST, turn on the high-order bit in PAGE NUMBER.

(B) The Page Number List

LENGTH: Halfword that specifies the number of bytes in the PAGE NUMBER field.

PAGE NUMBER; Optional: Provides a way to specify the starting page number for system output listing. If you do not wish to specify a starting page number, you must set the length field to binary zeros.

PAGE NUMBER is a 1-4 byte character string that may specify the starting page number of system output listing. This value is reset to the current page number upon completion of the present invocation of the IDCAMS processor.

(C) The Input/Output List

Optional. Provides the means of identifying those files for which the invoker wishes to manage all I/O operations.

n: A fullword that specifies the number of groups of three fields that follows. Every group consists of a DNAME address, an IOROUTINE address, and a USER DATA address.

DNAME: Address of a character string that identifies a file that causes the invocation of the associated IOROUTINE for all I/O operations (including OPEN and CLOSE) against the file. The character string identifies the data set as follows: A 10-byte character string, the first two characters are 'DD', the next 8 characters are the DNAME field left-justified (padded with blanks if necessary), which may appear in the FILE, INFILE, or OUTFILE parameters of any IDCAMS command. The SYSIPT (DDSYSIPT) and SYSLST (DDTSYSLST) DLBL names may also appear if the Invoker wishes to manage these files.

IOROUTINE: Address of the program that is to be invoked to process I/O operations upon the file associated with DNAME. This routine, instead of the processor, is invoked for all operations against the file. For information on linkage and interface conventions between the IOROUTINE and IDCAMS, see "User I/O Routines," below.

USER DATA: Address of a data area that the user may use for any purpose.

(D) The Options List

Required. Provides a way to specify processing options. If you do not wish to specify any options, you must set the length field to binary zeros.

LENGTH: Halfword that specifies the number of bytes in the options field.

OPTIONS: Character string that contains the processing options of the Access Method Services (AMS) PARM command. The options must comply to the parameter syntax of the IDCAMS PARM command.

(E) The DNAMES List

Optional. This value must be a halfword of binary zeros.

(F) USER DATA AREA

As long as no return or reason codes are inserted, the *user data area* (8 bytes) contains the character string **EXTRACT**.

If return or reason codes are inserted, the meaning is as follows:

FF = Valid return or reason code follows.
xx = Catalog return code in hexadecimals.
yy = Catalog reason code in hexadecimals.

User I/O Routines

User I/O routines enable a user to perform all I/O operations for a file that would normally be handled by the IDCAMS processor. This makes it possible, for instance, to control the command input stream by providing an I/O routine for SYSIPT.

A user I/O routine is invoked by IDCAMS for all operations against the selected files. The identification of the files and their associated I/O routines is via the Input/Output list of the processor invocation parameter list (Figure 65).

When writing a user I/O routine, the user must be aware of three things. First, the processor handles the user file as if it were a nonVSAM file that contains undefined records (maximum record length is 32760 bytes) with a physical

sequential organization. The processor does not test for the existence of the file. Second, the user must know the data format so that the user's routine can be coded to handle the correct type of input and format the correct type of output. Third, every user routine must handle errors encountered for files it is managing and provide to the processor a return code in register 15. The processor uses the return code to determine what to do next.

The permissible return codes are:
- 0 – operation successful
- 4 – end of data for a GET operation
- 8 – error encountered during GET/PUT operation, but continue processing
- 12 – do not allow any further calls (except CLOSE) to this routine

Figure 66 shows the argument list used in communication between the user I/O routine and the IDCAMS processor. The user I/O routine is invoked by the processor for OPEN, CLOSE, GET and PUT routines. The type of operation to be performed is indicated via the IOFLAGS. The IOINFO field indicates, for OPEN and CLOSE operations, the filename of the DLBL or TLBL statements for the file. For GET and PUT operations, the IOINFO field is used to communicate the record length and address.

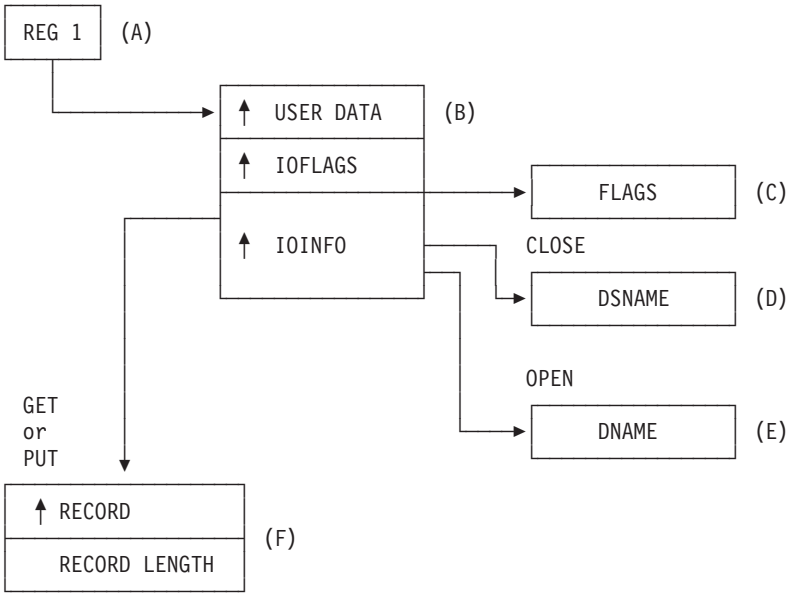


Figure 66. Arguments Passed to and from a User I/O Routine

Explanation

The following explains Figure 66.

(A) Register 1: When IDCAMS gives control to the PUT I/O routine pointed to by the IOROUTINE, Register 1 points to an IDCAMS argument list. Refer to Figure 65 on page 316.

(B) User Data: The user data pointer is obtained from the input/output list of the processor invocation parameter list. The user data area contains the character string EXTRACT, or return or reason codes. Refer to Figure 65 on page 316.

(C) Flags: The following explains the fullword of FLAGS:

Byte	Value or Bit Pattern	Meaning
1 (1)	0 4 8 12	OPEN CLOSE GET PUT
2	1..... .1.... ..1...	OPEN for Input OPEN for Output On OPEN, indicates that IOINFO contains the address of a DLBL or TLBL file name.
3, 4 (2)	0 n	A normal data record is to be written. If an IDC message is to be written, the message serial number converted to binary.

(1) Operation only.
(2) Record type for PUT only.

(D) DSNAME: A 44-byte field, left justified, and padded with blanks if necessary. It contains the name of the data set to be closed.

(E) DNAME: An 8-byte field, left justified, and padded with blanks if necessary. It contains the DLBL or TLBL file name.

(F) RECORD and RECORD LENGTH:

- For a GET: The information is returned to the processor by the user's I/O routine in the 8-byte area passed to the routine. Where:
  RECORD: Address of the retrieved record.
  RECORD LENGTH: Fullword length of the retrieved record.
- For a PUT: The processor gives the information to the user's I/O routine. Where:
  RECORD: Address of the record to be written.
  RECORD LENGTH: Fullword length of record to be written.

Appendix C. Advantages of the ISAM Interface Program (IIP)

This Appendix ...

Is for users of ISAM who want to **convert** from ISAM to VSE/VSAM.

The information helps you to decide whether your existing ISAM processing programs can use the *ISAM Interface Program (IIP)* to process files that have been converted from ISAM format to VSE/VSAM format.

The IIP minimizes your conversion costs and scheduling problems by permitting ISAM programs to process VSE/VSAM files. Also, through IIP, ISAM programs can process ISAM files and VSE/VSAM files concurrently.

Comparison of VSE/VSAM and ISAM

In most cases, you can get better performance with VSE/VSAM while achieving essentially the same results that can be achieved with ISAM. Furthermore, VSE/VSAM can achieve results that cannot be achieved with ISAM.

The extent to which you can use your existing ISAM processing programs to process key-sequenced files relates to the similarities between ISAM and VSE/VSAM, as well as to limitations of the IIP.

The following describes the similarities and differences between VSE/VSAM and ISAM in the areas that you are familiar with from using ISAM, and outlines the functions of VSE/VSAM that have no counterpart in ISAM.

Differences Between ISAM and VSE/VSAM

A number of things that ISAM does are done differently or not at all by VSE/VSAM, even though similar results are achieved. The following describes the areas in which VSE/VSAM and ISAM differ.

Index Structure

Both a VSE/VSAM key-sequenced file and an indexed-sequential file have an index that consists of levels, with a higher level controlling a lower level. In ISAM, either all or none of the index records of a higher level can be kept in storage. VSE/VSAM keeps individual index records in storage during processing, the number depending on the amount of buffer space provided.

Relation of Index to Data

The relation of a VSE/VSAM index to the direct access storage space whose records it controls is quite different from the corresponding relation for ISAM, in particular with regard to overflow areas for record insertion.

ISAM keeps a two-part index entry for every primary track on which a file is stored. The first part of the entry indicates the highest-keyed record on the primary track. The second part indicates the highest-keyed record from that primary track that is in the overflow area, and gives the physical location in the overflow area of the lowest-keyed overflow record from that primary track. All the records in the overflow area from a primary track are chained together, from the lowest-keyed to

© Copyright IBM Corp. 1979, 2005

the highest-keyed, by pointers that ISAM follows to locate an overflow record. Overflow records are unblocked, even if primary records are blocked.

VSE/VSAM does not distinguish between primary and overflow areas. A control interval (CI), whether used or free, has an entry in the sequence set, and after records are stored in a free CI, it is processed in exactly the same way as other used CIs. Data records are blocked in all CIs and addressed, without chaining, by way of an index entry that contains the key (in compressed form) of the highest-keyed record in a CI.

Defining and Loading a File

All VSE/VSAM files are defined in a catalog. Records are loaded into a file with IDCAMS or with the processing program, in one execution or in stages. When loading new records into an empty key-sequenced file, the index is built automatically. IDCAMS does not merge input files. For a key-sequenced file, however, input records are merged in key sequence with existing records of the output file.

Deletion of Records

With ISAM, records cannot be deleted until the file is reorganized; you must mark the records you want to delete.

VSE/VSAM automatically reclaims the space in a key-sequenced file and combines it with any existing free space in the affected CI. VSE/VSAM's use of distributed free space for insertions and deletions requires less file reorganization than ISAM does.

VSE/VSAM Functions That Go Beyond ISAM

VSE/VSAM Functions Available through IIP

Secondary Allocation of Storage Space: When you define a VSE/VSAM file, you can specify the amount of direct access storage space that is to be allocated automatically, when required, beyond the primary space allocation. You can specify the amount of secondary space in number of data records, or in number of blocks (for FBA), or tracks or cylinders (for CKD).

Automatic File Reorganization: VSE/VSAM partially reorganizes a key-sequenced file by splitting a control area (CA) when it has no more free control intervals (CIs) and if one is needed to insert a record. VSE/VSAM allocates a new CA and gives it the contents of approximately half of the CIs of the old CA; about half of the CIs of every CA are then free.

Key Range Allocation: With a multiple volume key-sequenced file, you can assign data to various volumes according to the ranges of key values in the data records. For example, for a file that resides on three volumes, you might assign keys A-E to the first volume, F-M to the second volume, and N-Z to the third.

Automatic CLOSE: Because it is essential for the integrity of a file that it be closed properly, VSE/VSAM attempts to close all open VSE/VSAM files within the partition at both normal or abnormal termination of the job step. It also restores control blocks to their status before the file was opened, and it frees storage that open routines used for VSE/VSAM control blocks.

Job Control: ASSGN or EXTENT statements are not required for file access. The IIP supports disposition processing (DISP parameter on DLBL statement) for reusable and dynamic files.

VSE/VSAM Functions Requiring Conversion from ISAM

If you convert your ISAM programs to VSE/VSAM, the following additional VSE/VSAM functions become available to you.

Addressed Sequential Access: With VSE/VSAM, you can retrieve and store the records of a key-sequenced file by relative byte address (RBA), as well as by key. With ISAM you can position by physical address, but you must retrieve in a separate request.

Direct Retrieval by Generic Key: With VSE/VSAM, you can retrieve a record directly, not only with a full-key search argument, but also with a generic search argument. ISAM can only position a record by generic argument; you must retrieve the record separately.

Concurrent Request Processing: A processing program can issue concurrent requests for a single ACB. The requests can be sequential or direct, or both, for the same part or different parts of the file. VSE/VSAM maintains a position in the file for every concurrent request.

No Abnormal Terminations by OPEN: The VSE/VSAM OPEN routine does *not* abnormally terminate the user program, but returns an explanatory message in all cases where it cannot carry out a request to open a file.

Alternate Indexes for Key-Sequenced and Entry-Sequenced Files: Instead of only one index, you can build several indexes (called alternate indexes) for a single data file. Every index can access the file in a different way so that you need not keep multiple copies of the same information organized differently for different applications.

Variable-Length and Spanned Records: In addition to fixed-length records, VSE/VSAM supports variable-length and spanned records.

Skip Sequential Access: You can process a key-sequenced file sequentially and skip records automatically, as though you were using direct access.

Preparations and Using the ISAM Interface Program

Programming Interface Information

Before you can use the IIP, you have to:

- Consider restrictions in the use of the IIP and VSE/VSAM
- Convert ISAM files to VSE/VSAM files, that is:
  - Define a VSE/VSAM file
  - Load ISAM files into the VSE/VSAM file

  For a summary on converting files and processing them, refer to Figure 67 on page 327.
- Change ISAM job control statements

Step 1: Consider Restrictions in the Use of IIP and VSE/VSAM

Most programs that use ISAM require little or no modification for using the IIP to process VSE/VSAM files. It is suggested that you evaluate your existing ISAM programs in terms of your requirements, and in terms of their suitability to use the IIP.

The following lists prerequisites for using the IIP, and those ISAM functions for which there is no VSE/VSAM equivalent or which cannot be simulated by the IIP.

- The program must run successfully under ISAM. IIP does not check for parameters that are invalid for ISAM.
- The program must use standard ISAM interfaces.
- Record ID processing of ISAM cannot be used because VSE/VSAM does not use the record ID functions.
- VSE/VSAM does not return device-dependent information or the storage or disk address of the record containing the error in the ERREXT parameter list.
- VSE/VSAM always assumes EXTEND mode when loading a file. If you try to reload an existing file, VSE/VSAM returns a sequence error code to you. You must DELETE and DEFINE the file (or specify DISP=NEW to reset a reusable file) before reloading it.
- The ISAM program cannot open a DTF while another ISAM DTF or VSE/VSAM ACB is already open for output processing for the same file unless VSE/VSAM SHAREOPTIONS(3) was specified for the file. If you select SHAREOPTIONS (3), you must accept the responsibility of maintaining file integrity. SHAREOPTIONS(4) may also be valid if the records accessed concurrently are not in the same CA.
- Files defined with SHAREOPTIONS(4) cannot be shared between IIP users in different systems because IIP always opens a file for output. Note that another system can open the file for input using native VSE/VSAM access.

Ensure that your existing ISAM programs comply with the restrictions described above. If they comply, there is no need to assemble or link-edit these programs again.

Step 2: Define a VSE/VSAM File

Define a key-sequenced VSE/VSAM file by using the IDCAMS command "DEFINE CLUSTER", described in the book *VSE/VSAM Commands*. Note the following information for specifying the command:

Data Space

You may define the file on a volume that already contains enough free VSE/VSAM data space for it, or you may define data space when you define the file (unique file).

Buffer Space

The BUFFERSPACE parameter in the DEFINE command specifies how much space VSE/VSAM will have for I/O buffers. If you do not specify the BUFFERSPACE parameter, the default is at least two data buffers and one index buffer. For better performance, however, you can specify space for more than two data buffers and one index buffer.

Reusable File

If you have a file that requires rebuilding, initially specify the REUSE parameter in the DEFINE command. When reloading the file, specify DISP=NEW in the DLBL statement.

DTFIS Parameters and DEFINE Command Relationship

For VSE/VSAM, some of the information given in the DTFIS parameters must be specified correctly in the DEFINE command, because the value specified in the command overrides the DTF. These parameters and the corresponding DEFINE command options are:

DTFIS Parameter	DEFINE Option
HOLD=YES	SHAREOPTIONS(4).
KEYLEN=n and KEYLOC=n	KEYS (length, offset) o length should always be set to KEYLEN. o offset should be set to: KEYLOC-1 if DTFIS RECFORM=FIXBLK 0 if RECFORM=FIXUNB
RECSIZE=n	RECORDSIZE (average, maximum) The average and maximum values must be equal. If (in the DTFIS): o RECFORM=FIXBLK, you should set RECORDSIZE to RECSIZE. o RECFORM=FIXUNB, you should set RECORDSIZE to RECSIZE + KEYLEN.
VERIFY=YES	WRITECHECK.

The IIP uses the following DTFIS parameters (all other parameters are ignored):

```
   ERREXT=YES   (for a description of the ERREXT parameter
                 with IIP, see
Table 11 on page 327)
   IOAREAL=name (used when IOROUT=LOAD)
   IOAREAS=name (used if SETL BOF is issued)
   IOREG=(r)
   IOROUT=LOAD, ADD, RETRVE, ADDRTR
   KEYARG=name
   RECFORM=FIXUNB, FIXBLK
   WORKL=name
   WORKR=name
   WORKS=YES
```

Step 3: Load the VSE/VSAM File

After you have defined the VSE/VSAM file, load the file by copying your existing ISAM file into it. To do so, you may use one of the following:

- Your ISAM load program, by way of the IIP
- The IDCAMS command "REPRO", described in the book *VSE/VSAM Commands*.

Notes:

1. Do not move files from ISAM to tape and then from tape to VSE/VSAM.
2. The REPRO procedure must be from disk to disk.
3. If you have records marked for deletion in the ISAM file and do not want them copied into the VSE/VSAM file, you should use your ISAM load program, because the REPRO command copies all records from the ISAM file, including those marked for deletion.

4. REPRO of a fixed, unblocked ISAM file creates records consisting of the original record preceded by its key. The IIP strips this leading key when a program that specifies fixed unblocked ISAM is executed, and returns only the original record to you. The leading key is returned with the record, however, when the file is accessed in native VSE/VSAM mode.

Step 4: Changing ISAM Job Control Statements

To satisfy the requirements of VSE/VSAM, you have to replace the job control statements for ISAM by job control statements for VSE/VSAM.

The following is an example of VSE/VSAM job control statements used with an ISAM program:

```
// JOB    PROCESS A VSE/VSAM FILE
// DLBL   IFN,'MSTRFILE',,VSAM
// EXEC   ISAMPGM,SIZE=nK
 .
 .   SYSIPT data for the program ISAMPGM
 .
/*
/&
```

One DLBL statement is required for the file; it connects the ISAM filename (IFN) to the VSE/VSAM cluster name (MSTRFILE) stored in the catalog. The DLBL type code parameter (VSAM) causes the *ISAM Interface Program* to be called. The same VSE/VSAM job control statements are required regardless of the type of ISAM program.

End of Programming Interface Information

What the ISAM Interface Program Does

Programming Interface Information

When a processing program that uses ISAM opens a VSE/VSAM file, the VSE open routine detects the need for the IIP by the type code "VSAM" specified in the DLBL statement. The processing program calls the IIP OPEN routine to:

- Build control blocks required by VSE/VSAM
- Load the ISAM command processor
- Flag the DTFIS for the IIP to intercept ISAM requests

Figure 67 on page 327 summarizes the steps required to convert indexed sequential files to key-sequenced files and processing them either with programs that have been converted from ISAM to VSE/VSAM, or with programs that still use ISAM.

Most existing processing programs that use ISAM can process VSE/VSAM files through the *ISAM Interface Program* (IIP) with little or no change.

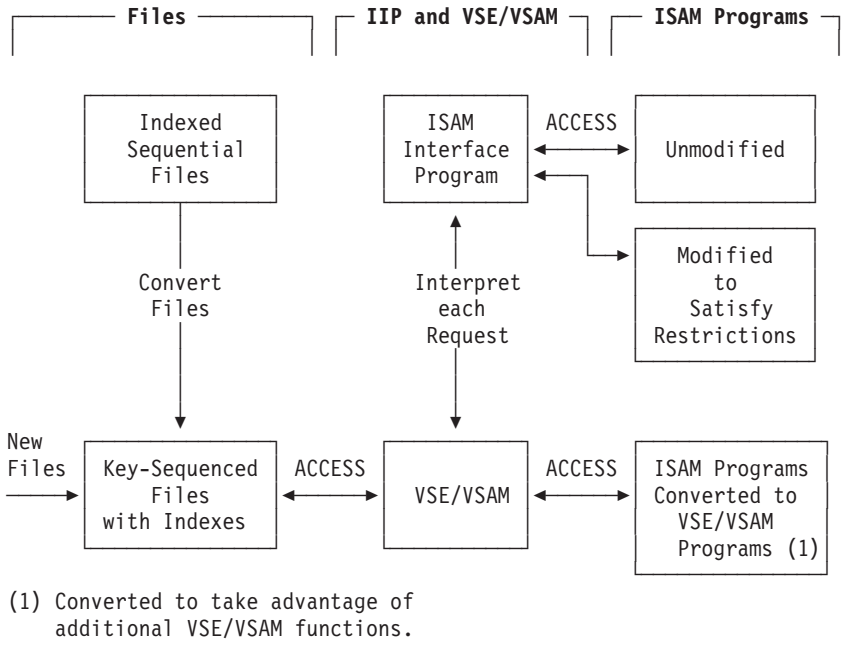


Figure 67. Using the ISAM Interface Program

The IIP intercepts every subsequent ISAM request, analyzes it to determine the equivalent keyed VSE/VSAM request, which it defines in the RPL constructed by OPEN, and then initiates the request.

The IIP interprets VSE/VSAM's return codes and, if the VSE/VSAM condition corresponds to an ISAM condition, turns on the respective bit in the filenameC byte in the DTFIS. For irrecoverable errors that cannot be posted in the filenameC byte, the IIP prints a message, closes the VSE/VSAM file (by the VSE/VSAM CLOSE routine), and ends the job. If a physical I/O error occurs and ERREXT=YES was specified in the DTFIS, the IIP transfers additional error information to the processing program. Table 11 shows the format of the ERREXT parameter list.

Table 12 and Table 13 show the formats of the filenameC byte for ISAM processing through the IIP.

Table 11. ERREXT Parameter List for ISAM Programs with IIP

Bytes	Bits	Contents
0-3	-	DTF address
4-15	-	Not supported by the IIP
16	0	Data
	1	VSE/VSAM sequence set
	2	VSE/VSAM index set
	3-5	Not used
	6	Read operation
	7	Write operation
17	-	Not supported by the IIP

Table 12. FilenameC with IIP when IOROUT=ADD, RETRVE, or ADDRTR

Bit	Cause in ISAM	Cause in IIP/VSAM
0	Disk error	Disk error
1	Wrong length record	Not set
2	End of file	End of file
3	No record found	No record found
4	Illegal ID specified	Not supported by IIP
5	Duplicate record	Duplicate record
6	Overflow area full	No more VSE/VSAM data space available
7	Overflow	Not set

Table 13. FilenameC with IIP when IOROUT=LOAD

Bit	Cause in ISAM	Cause in IIP/VSAM
0	Disk error	Disk error
1	Wrong length record	Not set
2	Prime data area full	No more VSE/VSAM data space
3	Cylinder index area full	No more VSE/VSAM data space
4	Master index full	No more VSE/VSAM data space
5	Duplicate record	Duplicate record
6	Sequence check	Sequence check
7	Prime data area overflow	Not set **Note:** If there is no more VSE/VSAM data space, bits 2 through 4 are set.

End of Programming Interface Information

Appendix D. Compatibility With Other Products

This Appendix...

Describes what to avoid so as not to endanger the portability of VSE/VSAM files to other systems.

Describes similarities between VSE/VSAM and ACF/VTAM.

Portability of VSE/VSAM Files to DFSMSdfp VSAM

You can port files and volumes to MVS if you avoid the use of device types, file types, and functions that are not supported by MVS.

The following functions are not supported by DFSMSdfp VSAM:
- *VSE/VSAM Space Management for SAM Function*
- EXTOPT parameter of the SHOWCAT macro
- IDCAMS CANCEL command
- SYNCHK parameter of the PARM command
- IGNOREERROR parameter of the DELETE command
- %%-function in the NAME parameter of the DEFINE CLUSTER command that gives a certain partition- or processor independence to the cluster.

Other critical functions are listed below and explained afterwards.
- FBA support
- Dedicated VSE/VSAM volume
- Data space classification
- Default models
- Default volumes
- Multiple volume ownership
- Catalog check services
- VSE/VSAM Backup/Restore Function
- Device Dependency
- VSE/VSAM data compression

FBA Support

Files on an FBA device cannot be processed by MVS. This does not affect the processing of catalog entries or files for a CKD device.

Files on an FBA device can be transferred (through EXPORT and IMPORT) from VSE/VSAM to a CKD device on an MVS system and vice versa.

Dedicated VSE/VSAM Volume

The DEDICATE parameter in the DEFINE commands is not supported by MVS. However, a volume allocated to VSE/VSAM with the DEDICATE parameter can be processed by MVS.

Data Space Classification

Space class specifications are not supported by MVS, but a file, data space, or volume with space classes under VSE/VSAM can be processed by MVS/VSAM.

© Copyright IBM Corp. 1979, 2005

MVS/VSAM files can be transported to VSE/VSAM volumes defined with classes.

Default Models

They allow users of IDCAMS to choose their own parameter defaults. Default models are not supported by MVS/VSAM; however, the resultant file and catalog data can be processed by MVS.

Default Volumes

They allow users to omit explicit volume lists in the DEFINE CLUSTER and DEFINE ALTERNATEINDEX commands. Also, the parameter DEFAULTVOLUMES is provided in the IMPORT command to allow users to override the exported volumes list. The required volumes are selected from the volumes list associated with the default model.

The command functions are not supported by MVS; however, the resulting file and catalog data can be processed by MVS.

Multiple Volume Ownership

Multiple catalogs can own space on the same disk volume, providing that only one catalog resides on that volume.

After you use VSE/VSAM to define, on one volume, several spaces belonging to different catalogs, you can perform the following activities while running on MVS:

- Define or delete a file in the space belonging to any one of the catalogs.
- Access any file.
- Define additional space belonging to any one of the catalogs.
- Define a UNIQUE file belonging to any one of the catalogs.
- Delete a UNIQUE file.

Do not issue a DELETE SPACE, DELETE MASTERCATALOG, or DELETE USERCATALOG whenever spaces belonging to different catalogs reside on the volume. If you were to do this, the spaces belonging to other catalogs would be deleted from the volume, but their catalog entries would remain.

Catalog Check Services

Automatic Catalog Check

This service examines VSE/VSAM catalogs containing DFSMSdfp files (alias and generation data group), but it can validate only their horizontal and vertical extension chains. It does not check associations or volume information for DFSMSdfp files.

Backup/Restore

Backup/Restore first verifies that an MVS/VSAM file to be backed up can be successfully restored. In cases where restoration is not possible, VSE/VSAM ignores the backup request and, instead, issues a message.

The IDCAMS commands BACKUP and RESTORE are not supported by IDCAMS under MVS/VSAM.

Device Dependency

VSE/VSAM treats the IBM 3995 Model 151 Optical Library Dataserver as an IBM 3390 Model 2 direct access storage device. However, a VSE/VSAM catalog that resides on an IBM 3995 Model 151 Optical Library Dataserver cannot be shared with a DFSMSdfp system.

VSE/VSAM Data Compression

VSE/VSAM compressed files can be ported to an MVS system using the EXPORT and IMPORT or REPRO functions. The portable data set will be in uncompressed format.

Compatibility of VSE/VSAM with DFSMSdfp VSAM

ICF catalogs created under DFSMSdfp VSAM are incompatible with the VSE/VSAM catalog, and VSE/VSAM cannot process them. Compatibility of files, IDCAMS job streams, and VSE/VSAM user programs is unchanged.

Similarities between VSE/VSAM and ACF/VTAM

IBM VTAM is an access method for teleprocessing. There is considerable similarity between the two access methods (VSE/VSAM and ACF/VTAM) regarding control block names and fields, control block manipulation, and general approach to request handling.

Both access methods use an ACB. The VSE/VSAM ACB represents the file. In VTAM, however, the ACB essentially represents an application program. Both types of ACBs are objects of the OPEN macro instruction, and VSE/VSAM and VTAM ACBs can be opened with one macro instruction.

Both types of ACBs can contain pointers to an exit list. Both VSE/VSAM and VTAM exit lists contain addresses of routines to be entered when error conditions occur (LERAD and SYNAD exit routines) and addresses of routines to be entered when special situations occur.

Both access methods follow the same general I/O-request procedure. An I/O macro instruction is issued that indicates an RPL. The RPL in turn contains information about the request, such as the location of the I/O work area or whether the request is to be handled synchronously or asynchronously.

Finally, both access methods use the same macro instructions (GENCB, MODCB, TESTCB, and SHOWCB) to generate and manipulate their respective ACB, EXLST, and RPL control blocks.

To make control blocks unique, a special parameter is used when the control block is generated. By specifying AM=VTAM on the ACB, EXLST, or RPL macro instruction, the control block is generated in VTAM form. Omitting this parameter causes a VSE/VSAM control block to be built. A VSE/VSAM control block will also be built if AM=VSAM is specified. If an installation uses both of these access methods, it may be desirable to have AM=VSAM specified in VSE/VSAM programs for documentation purposes.

Appendix E. VSE/VSAM Labels

This Appendix ...

Provides **conceptual** information on the **labels** that are used with VSE/VSAM for identifying volumes, data space, and files. It explains how the labels are processed, and includes **definition examples** relating to job control and IDCAMS commands.

Types of VSE/VSAM Labels

VSE/VSAM maintains identifying information for its files in a central location called the VSE/VSAM catalog. Volumes that contain VSE/VSAM files have the same internal labels as other volumes. Most of the identifying information for VSE/VSAM files, however, is in the VSE/VSAM catalog.

VSE/VSAM uses a:
- Volume label (VOL1)
- Data space label and its continuation (format-1 and format-3)
- VTOC label (format-4)

User-standard labels are not supported. Neither is the F-5 label, but space is reserved for it on the VTOC for the purpose of MVS/SP compatibility.

Volume Label

The *volume label* (VOL1) is generally written during initialization. At that time, a permanent volume number is written on the volume as part of the label to give the volume a permanent ID.

Data Space Label

The VSE/VSAM *format-1 VTOC label* describes direct access space; the characteristics of the logical files that occupy that space are described in the VSE/VSAM catalog.

There is a format-1 label for every VSE/VSAM data space that is on the volume. Every data space consists of one or more separate extents:
- Up to three extents are described in the format-1 label.
- Extents additional to the first three extents are described in a format-3 label; the format-3 label is pointed to by the format-1 label. (Refer to "Space Continuation Label" on page 334.)

Usually, you do not name a VSE/VSAM data space, because VSE/VSAM automatically assigns a name to the data space. This name is placed into byte 1 through 44 of the key area (called the *44-byte key area*). However, if you allocate a data space to contain the data or the index of only one specific VSE/VSAM file (called a *unique* file), the 44-byte key area will contain the name given to the data or the index when you define it.

© Copyright IBM Corp. 1979, 2005

When a new VSE/VSAM data space is created (IDCAMS command DEFINE), existing format-1 and format-3 labels are read and checked, and new labels are created by the catalog and space management routines.

Space Continuation Label

A *format-3 VTOC label* is written whenever a VSE/VSAM data space occupies more than three separate areas (extents) of a volume. It is used to supply the limits (starting and ending addresses) of the additional extents. Thirteen separate extents can be defined by one format-3 label. This label is pointed to by the format-1 label.

VTOC Label

A *format-4 VTOC label* defines the *volume table of contents* (VTOC). Also, if a volume contains VSE/VSAM spaces, the label defines the volume as a *VSE/VSAM volume*.

The format-4 label is always the first record in the VTOC. The record is written when you initialize your disk pack by using the *IBM Device Support Facilities (ICKDSF)*. For details, refer to the manual *Device Support Facilities User's Guide and Reference*.

The OPEN/CLOSE routines refer to the format-4 label to determine the extent of the VTOC.

The format-1 and format-3 labels are stored in the VTOC and are processed as described under "VTOC Label Processing" on page 335.

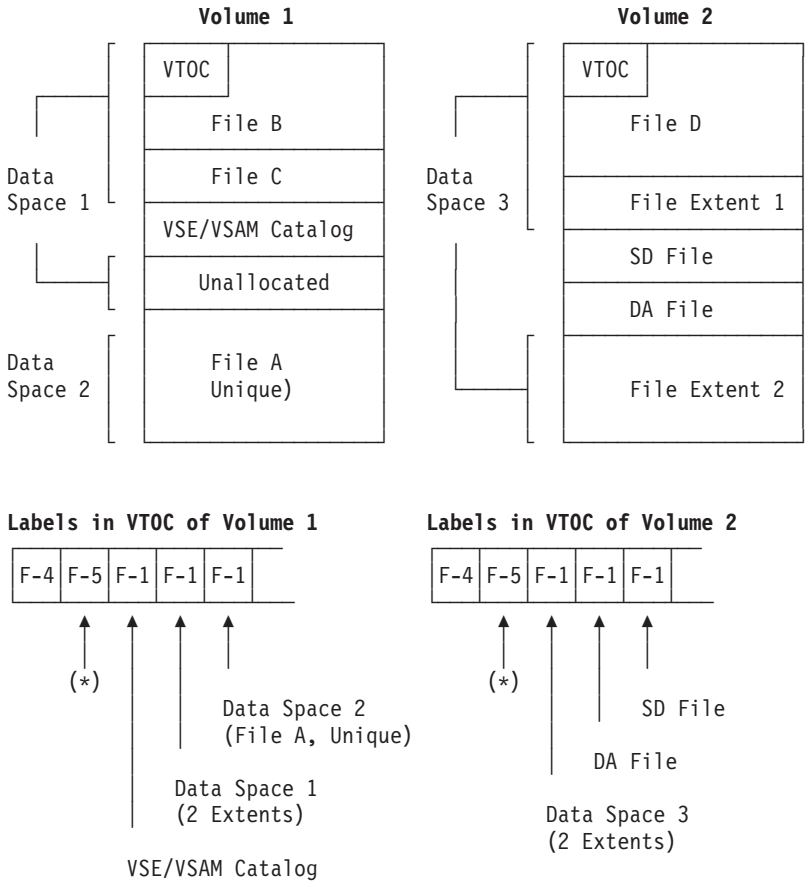
Location of Labels

Volume Layouts

Each volume has a VTOC that contains labels for the data spaces.

Figure 68 on page 335 shows two volumes that contain VSE/VSAM data spaces (1, 2, and 3). The figure illustrates the relationship between volumes, VSE/VSAM data spaces, and labels in the VTOC. Specifically:

- The two volumes contain the VSE/VSAM data spaces 1, 2, and 3. Each volume has a VTOC that describes the data spaces owned by VSE/VSAM. The files are described in the VSE/VSAM catalog.
- *Data Space 2* is occupied by *File A*; this file is assumed to be a *unique* file. If a unique file occupies a data space, no other file can be suballocated in the data space, and File A cannot be extended to any other data space.
- The 44-byte name field of the label for Data Space 2 contains the name (the file-ID) of file A. The 44-byte name fields of the other data spaces contain the data space name that is automatically generated by VSE/VSAM



(*) VSE/VSAM does not use the format-5 VTOC label,
but space is reserved for the label for reasons
of DFSMSdfp compatibility.

Figure 68. Volume Layouts of VSE/VSAM Files

Label Information Area

VSE/VSAM file label information, and standard labels for a user catalog, can be submitted following // OPTION STDLABEL=ADD or // OPTION PARSTD=ADD. VSE/VSAM searches the partition temporary user label area (USRLABEL), the partition standard label area (PARSTD), and the system standard label area (STDLABEL), in that order. Thus, it is possible to override permanent label sets for a single job by submitting the new label set under

// OPTION USRLABEL. The default is // OPTION USRLABEL and can be omitted.

VTOC Label Processing

VSE/VSAM Data Spaces

The format-1 and format-3 VTOC labels describe VSE/VSAM data spaces. A data space consists of one or more extents on a single volume allocated to VSE/VSAM and controlled by a VSE/VSAM catalog. VSE/VSAM files are written in data spaces.

Even if it does not contain any files, a data space is owned by VSE/VSAM and is not available for files of other access methods.

Label processing is done when data spaces (including catalogs and unique files) are created or deleted, and during ALTER NEWNAME for unique files.

The format-1 and (if needed) format-3 VTOC labels are created and checked (for overlap or duplicate name) only when data spaces are created (including data spaces for unique files). If data spaces are deleted, their format-1 and format-3 labels are removed from the VTOC. Labels are also altered during RESETCAT processing if the data in the label and the catalog do not agree. When VSE/VSAM files are processed, the VSE/VSAM catalog is used for checking the location and characteristics of the files.

VSE/VSAM Files

VTOC label processing takes place only for *unique* VSE/VSAM files that are defined, deleted, or renamed.

VSE/VSAM files are normally defined after data spaces have been defined. The direct access space for the files is suballocated by VSE/VSAM from one or more data spaces. You may select the volume or volumes the file will reside on. You tell VSE/VSAM how much space to suballocate to the file initially and, optionally, how much additional space to suballocate when the file must be extended. VSE/VSAM decides which data spaces or portions of data spaces to suballocate to a file.

You can, however, specify the size and exact location of the file when you define it. In this case, the file is called *unique* and occupies its own data space which is defined when the file is defined. No other files can occupy that data space. If the file extends across more than one volume, it occupies one data space on every volume. The format-1 and format-3 labels still describe the data space(s) occupied by the unique file. A key-sequenced unique file requires separate data spaces for the data and the index components.

The *file-ID* parameter of the // DLBL statement (if specified) indicates the file you want to process. It is the same as the name of the file, stored in the catalog, which was specified in the NAME(entryname) parameter of the DEFINE statement. For VSE/VSAM data spaces, the format-1 label contains a data space name that is generated by VSE/VSAM.

VTOC Labels for FBA Devices

The physical block is the basic unit of storage on an FBA device. A disk address is a physical block number relative to the beginning of the volume.

A VTOC for an FBA is divided into control intervals (CIs) of the VSE/VSAM relative record format; the VTOC labels reside in these CIs. There is a slot for the VTOC label and its corresponding RDF in the CI. The CI size is a multiple of FBA block size; a CI always starts on a block boundary. Specify VTOC size through the DSF program.

The VOL1 label contains the VTOC CI size, the number of blocks per CI, and the number of labels per CI. VTOC labels are referenced according to relative record number (beginning with 1).

VSE/VSAM Data Space

VOL1 Label Processing

The VSE/VSAM VOL1 label fields are the same as for the other access methods.

The standard volume label (VOL1) must be located as follows:

- For CKD: on cylinder 0, track 0, record 3 (CKD).
- For FBA: in physical block 1.

  This block is called the *volume label block*.

If the VOL1 label is not located correctly, the job is cancelled.

The VOL1 label, written by the IBM Device Support Facilities (ICKDSF) program, contains a permanent volume number.

If any additional volume labels follow the VOL1 label, VSE/VSAM ignores them.

From the VOL1 label, VSE/VSAM determines the location of the VTOC.

Format-1 VTOC Label Processing for Unique Files

You must supply one // DLBL statement when creating a unique file and one // EXTENT statement for every separate extent on the volume that the data space will occupy. A multivolume unique file requires only one // DLBL statement, even though it occupies a data space on every volume.

// DLBL Statement: The // DLBL statement for defining a data space under VSE/VSAM requires only the filename parameter and the VSE/VSAM code. The // DLBL *filename* is identical to the *dname* specified in the FILE parameter of the DEFINE command.

The *file-ID* parameter is not required and is ignored if you specify it. The *date* parameter can be specified, but it has no real function. VSE/VSAM data spaces and files can be deleted only by using the DELETE command of IDCAMS.

// EXTENT Statement: An // EXTENT statement defines a continuous extent of the volume that is to be allocated to VSE/VSAM. There can be up to 16 extents in a data space, and a data space is contained entirely on one volume.

The // EXTENT statement provides the starting address (relative address) and the number of tracks (CKD) or blocks (FBA), which indirectly give the ending address. The // EXTENT statement also provides the order in which this extent should be processed in a multiple-extent unique file.

VSE/VSAM validates the // EXTENT specifications by checking the extent limits against the limits of the format-4 label, and every format-1 and format-3 label already written in the VTOC. If the new extent overlaps an existing extent, VSE/VSAM issues a message to the operator. If the overlapped extent is part of a file of another access method (expired or unexpired), the operator can delete the file or terminate the job. If the overlapped extent is part of a VSE/VSAM data space (or unique file), the operator can only cancel the job. VSE/VSAM data spaces or files (expired or unexpired) can only be deleted through the IDCAMS command DELETE.

If all extents of the new unique file are valid, VSE/VSAM writes one (or two, for a KSDS) format-1 label, and (if necessary) the format-3 label into an available location in the VTOC.

For the data or the index of a unique file, you may specify a data space name in the DEFINE command. If specified, this name is entered in the catalog and in the label. Remember that even though the name of a unique file is entered in the labels of the data space it occupies, the information describing the file is in the catalog.

Bytes 45-60, 63-75, 83-84, and 94 are written in the format-1 label for VSE/VSAM. This information is for compatibility with the format-1 labels of other access methods; during processing, VSE/VSAM uses the catalog, rather than using information from the VTOC.

Bytes 106-115 define the first (or only) extent allocated to the unique file component. If there is more than one extent, bytes 116-125 define the second extent, and bytes 126-135 define the third extent. These fields are written from the // EXTENT statements you supply.

If you have included more than three // EXTENT statements, VSE/VSAM writes a format-3 label and writes the address of that label in the pointer field (bytes 136-140) of the format-1 label.

If the unique file is deleted, the format-1 label (and if present, the format-3 label) is removed from the VTOC.

Format-3 VTOC Label Processing

The VSE/VSAM format-3 label fields are the same as for the other access methods, but a VSE/VSAM data space can have only one format-3 label.

If more than three extents are required for the data space (or unique file), VSE/VSAM sets up a format-3 label for the additional extents. A data space can consist of up to 16 extents, so only one format-3 label is allowed. VSE/VSAM processes the extent fields of the format-3 labels in the same manner as those of the format-1 label.

If the data space is deleted, the format-3 label is removed from the VTOC, along with the format-1 label.

Format-4 VTOC Label Processing

The format-4 label describes the VTOC (it does *not* describe the files or data spaces of individual access methods). However, a VSE/VSAM *indicator field* (bytes 77-87) is written in the format-4 label of any volume that contains VSE/VSAM data spaces or unique files. This field (volume time stamp) indicates the date and time the most recent VSE/VSAM data space was added to or deleted from the volume. For MVS compatibility reasons, this time stamp is repeated in bytes 88-95.

The same date and time are entered in the catalog. VSE/VSAM OPEN routines check if the volume time stamp matches the time stamp for it in the catalog. If they do not match, processing continues, but an error code is issued to indicate that the VTOC might not agree with the data space information in the volume's catalog entry.

Bit 0 of byte 85 indicates that this volume is owned by a VSE/VSAM catalog. Either VSE/VSAM space was defined on the volume, or the volume was listed as a CANDIDATE volume in the DEFINE SPACE command.

If all VSE/VSAM data space is deleted from a volume, the VSE/VSAM indicator field (bytes 77-87) is erased. The deleted space can be used by other z/VSE access methods.

VSE/VSAM Files

Defining a File: Suballocating Data Space (Non-Unique Files)

When a *non-unique* file is defined, the space for it can be *suballocated* from one or more existing data spaces on one or more volumes. This is illustrated in Figure 70 on page 343. VTOC label processing is not performed for the following reasons:

- Information needed to set up the file is in the DEFINE command.
- Information about data spaces to be suballocated to the file is in the VSE/VSAM catalog.

The resulting description of the file is entered in the catalog. The // DLBL and // EXTENT statements are not required; they are ignored if specified for a catalog.

You indicate the volume(s) on which the file will reside, the amount of space to be initially suballocated to the file and, optionally, the amount of space to be suballocated if the file must be extended. VSE/VSAM selects the extent(s) on the volume on which to write the file. If you specify more volumes than necessary for the primary space, the additional volumes can be used when the file is extended, if they contain free data space.

If none of the volumes contains free data space, new data spaces must be defined, or volumes with free data space can be made available to the file through the IDCAMS command ALTER. You can indicate in which order the volumes should be used. You can also decide to place certain portions of the file (key ranges) on certain volumes. If the file must be extended, VSE/VSAM can use only the volumes you indicated. For further information, refer to "Multiple Volume Support" on page 102.

Volume Mounting: The volume containing the catalog must be mounted, but the volumes on which the file is defined need not be mounted. Additional information about volume mounting requirements appears in "Using an Object as a Model" on page 57.

File Loading: Loading a file is a separate step from defining it. Records can be loaded into a file by a VSE/VSAM processing program by using the PUT macro, or the IDCAMS command REPRO.

Defining a File: Unique

A unique file occupies space described in the VTOC through // DLBL and // EXTENT statements (all this is similar to unique files with other access methods). Defining a unique file is illustrated in Figure 71 on page 343.

The data space for a unique file is defined (implicitly) in the same DEFINE command as the file itself. Characteristics of the file (such as logical record length) are specified in the command, just as with a suballocated file. Space information is taken from // DLBL and // EXTENT statements instead of from the DEFINE command.

The data and index of a unique key-sequenced file or alternate index require separate data spaces, and hence, separate // DLBL and // EXTENT statements.

A unique file cannot be extended. The extents of the file are the same as the extents of the data spaces and, because they are described in the VTOC, cannot be changed without deleting the file.

Label processing is performed for the data spaces of a unique file as described under "VSE/VSAM Data Space" on page 337. The only difference is that the 44-byte names of the data and index are placed in the labels and in the file's catalog entry. The data spaces of unique files are described in the VSE/VSAM catalog as well as in the VTOC.

Processing a File

When a previously defined file is processed by a VSE/VSAM application program or by a PRINT or REPRO command, a // DLBL statement is required for the file.

The statement is retrieved by VSE/VSAM OPEN from the label area. OPEN obtains the // DLBL statement from the file name specified in the IDCAMS macro ACB in the processing program. All the information required to process the file is in the VSE/VSAM catalog or the label area; no VTOC processing is performed (see Figure 72 on page 344).

// DLBL Statement: The // DLBL statement is used to find the 44-byte name of the file in the catalog. The 44-byte name matches the file-ID parameter. For PRINT and REPRO and VSE/VSAM application programs, the CAT parameter is required only if you want to override the system's assumption that the job catalog, or, if there is none, the master catalog, owns the file. The function of the job catalog is explained under "Specifying a Job Catalog" on page 42.

Volume Mounting: If the volumes allocated to the file are not mounted, messages are issued to the operator to mount the required volumes or cancel the job. A file can span a maximum of 16 volumes. If a multivolume file is opened for direct or keyed-sequential processing, all volumes must be mounted. If it is opened for addressed-sequential processing, only one volume at a time need be mounted.

The first allocation made on every volume is always the primary allocation. VSE/VSAM extends a suballocated file if:

- Secondary space allocation was specified when the file was defined.
- No secondary space allocation was specified, but overflow volumes are specified in the VOLUMES parameter of the DEFINE CLUSTER command. In that case, the primary allocation is taken.
- A volume that contains or can contain part of the file has unused data space of the required class.

Use the IDCAMS command ALTER to make more volumes available to the file after it has been defined.

The VOL1 label is checked to verify that the correct volume is mounted (volume serial number), and the format-4 label is checked to verify that the catalog is at the proper level (volume time stamp). Processing for these labels is described under "VSE/VSAM Data Space" on page 337.

Job Stream Examples

In the following, **Figure 69** through **Figure 72** show examples of the job streams you must supply to:

- Define a data space
- Define a file in a catalog
- Define a unique file
- Process a file

Notes:

1. In Figure 70 through Figure 71, further parameters are required in the DEFINE command to specify the characteristics (such as logical record length) of the VSE/VSAM file. These parameters are not shown, because they do not affect space allocation and label processing.
2. For the description of the IDCAMS command "DEFINE", refer to the manual *VSE/VSAM Commands*. More information about the job control statements required for VSE/VSAM is in "Use of z/VSE Job Control Statements for VSE/VSAM" on page 23.

Example - Define Data Spaces

Figure 69 on page 342 shows "samples" of the job streams you must supply to define data spaces. The figure shows allocation of an entire volume to VSE/VSAM (as a single data space), and allocation of a data space that is smaller than a single volume.

"Sample 3" and "Sample 6" show allocation of data spaces on different volumes of the same device type.

DEFINE command parameters supply the data space information.

For the master catalog, a // DLBL statement is required. In the samples, assume that the statement is in the label information area.

```
==== Sample 1: ====================================================
        // JOB      ALLOCATE A VOLUME TO VSE/VSAM
        * VOLUME IS OWNED BY MASTER CATALOG
        * ALL UNALLOCATED SPACE IS GIVEN TO VSE/VSAM
        // EXEC     IDCAMS,SIZE=AUTO
           DEFINE   SPACE(DEDICATE-
                    VOLUMES(PAY001))
        /*
        /&

==== Sample 2: ====================================================
        // JOB      DEFINE A VSE/VSAM DATA SPACE
        * SPACE IS OWNED BY MASTER CATALOG
        // EXEC     IDCAMS,SIZE=AUTO
           DEFINE   SPACE(ORIGIN(760) TRACKS(570)-
                    VOLUMES(PAY002))
        /*
        /&

==== Sample 3: ====================================================
        // JOB      DEFINE VSE/VSAM DATA SPACES ON SEVERAL VOLUMES
        * SPACES ARE OWNED BY USER CATALOG MYUCAT
        // DLBL IJSYSUC,'MYUCAT',,VSAM
        * DEFAULT ORIGIN USED FOR DATA SPACE ALLOCATION
        // EXEC     IDCAMS,SIZE=AUTO
           DEFINE   SPACE(TRACKS(190)-
                    VOLUMES(PAY003,PAY004))
        /*
        /&
==== Sample 4: ====================================================
        // JOB      ALLOCATE A VOLUME TO VSE/VSAM
        * VOLUME IS OWNED BY MASTER CATALOG
        * ALL UNALLOCATED SPACE IS GIVEN TO VSE/VSAM
        // EXEC     IDCAMS,SIZE=AUTO
           DEFINE   SPACE(DEDICATE-
                    VOLUMES(INV001))
        /*
        /&

==== Sample 5: ====================================================
        // JOB      DEFINE A VSE/VSAM DATA SPACE ON A VOLUME
        * SPACE IS OWNED BY MASTER CATALOG
        // EXEC     IDCAMS,SIZE=AUTO
           DEFINE   SPACE(ORIGIN(960) BLOCKS(2240)-
                    VOLUMES(INV002))
        /*
        /&

==== Sample 6: ====================================================
        // JOB      DEFINE VSE/VSAM DATA SPACES ON SEVERAL VOLUMES
        * SPACES ARE OWNED BY USER CATALOG MYUCAT
        // DLBL IJSYSUC,'MYUCAT',,VSAM
        * DEFAULT ORIGIN USED FOR DATA SPACE ALLOCATION
        // EXEC     IDCAMS,SIZE=AUTO
           DEFINE   SPACE(BLOCKS(3100)-
                    VOLUMES(DEV001,DEV002))
        /*
        /&

==== Sample 7: ====================================================
        // JOB      DEFINE SPACES ON A VOLUME BELONGING TO TWO CATALOGS
        // EXEC IDCAMS,SIZE=AUTO
           DEFINE SPACE(VOLUMES(SCRTC1) /* SPACE BELONGING TO MASTER CATALOG */-
                  CYLINDERS(40) ORIGIN(171)) /* CYLINDERS 9-48               */
           DEFINE SPACE(VOLUMES(SCRTC1) /* SPACE BELONGING TO USER CATALOG   */-
                  DEDICATE )   /* THE REST OF THE SPACE AVAILABLE            */-
                  CATALOG (MYUCAT)
        /*
        /&
```

Figure 69. Examples: Defining VSE/VSAM Data Spaces

Example - Define a File in a Catalog

Figure 70 shows the job stream you must submit to define a file that is suballocated from an existing data space. This file is recorded in the master catalog.

```
// JOB     SUB-ALLOCATE VSE/VSAM FILE
// EXEC    IDCAMS,SIZE=AUTO
   DEFINE  CLUSTER-
           NAME(MSTRFIL1)-
           VOLUME(PAY002)TRACKS(285 19))
/*
/&
```

Figure 70. Example: Defining a VSE/VSAM File Suballocated from a Data Space

Example - Define a Unique File

Figure 71 shows the job stream for defining a unique file. The data space information is supplied in // EXTENT statements. IDCAMS requires the VOLUMES and CYLINDERS (BLOCKS, TRACKS, or RECORDS) parameters in the DEFINE command if no MODEL is used.

```
// JOB     ALLOCATE A UNIQUE VSE/VSAM FILE
// DLBL    VDATANM,,,VSAM
// EXTENT  ,338002,1,,1330,380
// DLBL    VINDXNM,,,VSAM
// EXTENT  ,338002,1,,1710,190
// EXEC    IDCAMS,SIZE=AUTO
   DEFINE  CLUSTER(NAME(MSTRFIL3)UNIQUE)-
           DATA(FILE(VDATANM)VOLUMES(PAY002)CYLINDERS(20))-
           INDEX(FILE(VINDXNM)VOLUMES(PAY002)CYLINDERS(10))
/*
/&
```

Figure 71. Example: Defining a Unique VSE/VSAM File (File-ID MSTRFILE)

Example - Process a File

Figure 72 shows the job stream for processing a VSE/VSAM file. The CAT parameter of the // DLBL statement indicates the file name of the user catalog in which the file is recorded. The CAT parameter is written into the label information area. For details on the use of this parameter, see “Use of z/VSE Job Control Statements for VSE/VSAM” on page 23.

```
// JOB      PROCESS A VSE/VSAM FILE
// DLBL     VFILENM,'MSTRFILE',,VSAM,CAT=PRIVCAT
            (for the file)
// DLBL     PRIVCAT,'MYUCAT',,VSAM
// EXEC     USERPGM,SIZE=20K
   CSECT
     .
     .
     .
   ACB  DDNAME=VFILENM,...
     .
     .
     .
   END
/*
/&
```

Figure 72. Example: Processing a VSE/VSAM File with an Assembler Program

Appendix F. Diagnosis Tools

This Chapter ...
Contains Diagnosis, Modification, or Tuning Information.

Describes the **VSE/VSAM tools** summarized in Figure 73, below. The information is primarily for system administrators. The descriptions emphasize **running** the tools, rather than interpreting the output.

Under certain conditions, the *IBM support representative* might ask you to supply the output of diagnosis tools.

Tool	Purpose
Catalog Check Service Aid (IKQVCHK)	To identify erroneous catalog records. Under certain conditions, VSE/VSAM automatically invokes the tool, or you can invoke it yourself.
SNAP Dump Facility (IKQVEDA)	To print an error symptom string at the console. You can invoke a SNAP dump to provide an error code trace during program processing.
Maintain VTOC and VOL1 Utility (IKQVDU)	To assists you in maintaining the VTOC and VOL1 labels on disk devices.

Figure 73. VSE/VSAM Diagnosis Tools

For information on other diagnosis tools available in the z/VSE environment (for example, for producing various types of "dumps"), refer to the manual *z/VSE Guide for Solving Problems*.

Catalog Check Service Aid (IKQVCHK)

The *Catalog Check Service Aid* (IKQVCHK) helps you to determine whether a catalog has been damaged and, if damaged, the type and extent of the damage.

IKQVCHK is called under the following circumstances:

- If a file was *not* closed on a previous OPEN for update. In this case, VSE/VSAM OPEN tries to VERIFY the file before opening it. If the VERIFY is successful, VSE/VSAM calls IKQVCHK to examine the catalog records that describe the file. Note that only the records pertaining to that file are checked. The rest of the catalog is not examined.

  OPEN error codes might tell you to run IKQVCHK yourself for additional information.
- If the DELETE command with IGNOREERROR specified is issued. In this case, IDCAMS calls IKQVCHK, and the entire catalog is checked to ensure catalog integrity.

Furthermore, you should run IKQVCHK to assess catalog integrity in the following circumstances:

© Copyright IBM Corp. 1979, 2005

- After a system failure.
- When a file or catalog does not behave as expected.
- As part of regular system maintenance.

In Case of Errors

IKQVCHK issues error messages that identify missing or inconsistent information.

Perform the corrective action documented under the "IKQ-Prefixes" in the *z/VSE Messages and Codes*. Take the action before contacting IBM for support. If the problem persists, report it. Make IKQVCHK output available so that the system administrator or your IBM support representative can assess the extent of catalog damage and how much rebuilding is required.

How to Run a Check

Issue the following job control statement:

```
// EXEC IKQVCHK,SIZE=AUTO,PARM='aaaa...a/bbbbbbbb'
```

where:

```
aaaa      is the name (up to 44 characters) of
          the catalog that is to be checked.
          The entire catalog is checked.

bbbbbbbb  is the master, control, or update password of
          the catalog that is to be checked.
```

If you omit the PARM parameter, the default catalog is checked. (The default catalog is the job catalog if the IJSYSUC DLBL statement is specified. Otherwise, it is the master catalog.)

Examples of Error Messages

The following examples show a few of the problems that IKQVCHK can diagnose and the kinds of error messages that it produces. These examples are for users who want a deeper understanding of IKQVCHK.

Catalog errors are difficult to understand, because they involve internal catalog records, data, and control blocks which most users do not see. The *programmer action* associated with every message, however, does not require a full knowledge of the error condition. Similarly, you do not have to understand the listing of catalog records and the 512-byte catalog record dump that accompany the messages.

For the full documentation of all error messages issued by IKQVCHK, see the "IKQ-Prefixes" in the *z/VSE Messages and Codes*.

Example: Key-Range Names Not Matching

Figure 74 on page 347 shows the output associated with message IKQ0016I. In this example, the problem is that the low-key-range name for a particular object does not match the high-key-range name for that object, where the:

- Low-key-range name is XXXS1.INDEX
- High-key-range name is KSDS1.INDEX (identified by the submessage ASSOCIATED HKR REC FOLLOWS)

To correct this problem, run a DELETE command, specifying IGNOREERROR for KSDS1.INDEX. VSE/VSAM will delete the name KSDS1.INDEX from the catalog.

```
IKQ0016I DATA SET NAME NOT SAME IN HIGH AND LOW KEY RANGE RECORDS
LKR REC WITH INVALID DATA :
0000  0000002E 01000000 00000000 00000000   00000000 00000000 00000000 00000000 *................................*
0020  00000000 00000000 00000000 C9015700   8FE7E7E7 E2F14BC9 D5C4C5E7 40404040 *...........I....XXXS1.INDEX    *
0040  40404040 40404040 40404040 40404040   40404040 40404040 40404040 40FFFFFF *                               ...*
0060  FFFFFFFF FF92057F 00000F20 F000FFFF   FFFF0000 01000001 80000010 000000A0 *...........0..................*
0080  00FFFFFF FF0000FF FFFFFFFF FFFFFF00   00000000 05000000 C0000000 00010100 *................................*
00A0  00620201 00006803 01000000 44010062   60000060 00040000 000A0000 000A0000 *................................*
00C0  00000000 00001000 00000FF9 00000000   00000000 00000000 00000000 00000000 *...........9....................*
00E0  A54E6475 B838FE02 00010001 00000001   00000000 00000000 00000000 00000000 *................................*
0100  00009000 00000000 00000000 0000000F   0006C300 002C0327 3010200E F1F1F1F1 *..................C.........1111*
0120  F1F10000 80010000 00000000 10000000   A0000000 1000000A 00010000 02000000 *11..............................*
0140  00001400 11000000 01000000 01000100   00000000 009FFF00 00000000 00000000 *................................*
0160  00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000 *................................*
0180  00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000 *................................*
01A0  00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000 *................................*
01C0  00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000 *................................*
01E0  00000000 00000000 00000000 00000000   00000000 00000000 000001F9 01F90000 *.........................9.9..*
ASSOCIATED HKR REC FOLLOWS:
NAME = KSDS1.INDEX                              CI = 00002E
```

Figure 74. Example: Key-Range Names not Matching

Example: Erroneous Association Group Occurrence

Figure 75 on page 348 shows a problem in which two error messages are produced, IKQ0027I and IKQ0028I. The first hex dump shows the association group occurrence at X'08B'. The third byte of the association group occurrence indicates the record type (D).

The contents of CI 2 are printed just below the submessage ASSOC WITH UNEQUAL TYPE. Field X'2C' of a catalog record always tells what type of catalog record it is. In this case, the record type is X'C3' or C, meaning a cluster record, which does not match the record type (D) in the group occurrence (the first record).

To correct this error, issue a DELETE IGNOREERROR command for KSDS2 (specified in NAME).

```
IKQ0027I RECORD TYPE IN ASSOCIATION GROUP OCCURRENCE NOT EQUAL TO RECORD TYPE IN RECORD IT REFERENCES
IKQ0028I AFFECTED GROUP OCCURRENCE AT DISPLACEMENT 135 (X'087')
REC WITH ERRONEOUS GO FOLLOWS:
0000  0000002F 01000000 00000000 00000000   00000000 00000000 00000000 00000000 *................................*
0020  00000000 00000000 00000000 C3008D00   6CD2E2C4 E2F24040 40404040 40404040 *............C....KSDS2          *
0040  40404040 40404040 40404040 40404040   40404040 40404040 40404040 40FFFFFF *                            ...*
0060  FFFFFFFF FF92057F 00000F00 00000000   00030000 00020100 00060202 00000044 *................................*
0080  010006C4 00003000 06C90000 31000000   00000000 00000000 00000000 00000000 *...D.....I.....................*
00A0  00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000 *................................*
00C0  00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000 *................................*
00E0  00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000 *................................*
0100  00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000 *................................*
0120  00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000 *................................*
0140  00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000 *................................*
0160  00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000 *................................*
0180  00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000 *................................*
01A0  00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000 *................................*
01C0  00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000 *................................*
01E0  00000000 00000000 00000000 00000000   00000000 00000000 000001F9 01F90000 *.........................9.9..*
ASSOC REC WITH UNEQUAL TYPE:
0000  00000031 01000000 00000000 00000000   00000000 00000000 00000000 00000000 *................................*
0020  00000000 00000000 00000000 C3015700   8FD2E2C4 E2F24BC9 D5C4C5E7 40404040 *............C....KSDS2.INDEX    *
0040  40404040 40404040 40404040 40404040   40404040 40404040 40404040 40FFFFFF *                            ...*
0060  FFFFFFFF FF92057F 00000F00 6000FFFF   FFFF0000 03000003 80000008 000001B0 *................................*
0080  00FFFFFF FF0000FF FFFFFFFF FFFFFF00   00000000 05000000 C0000000 00010100 *................................*
00A0  00620201 00006803 01000000 44010062   60080060 000C0000 000A0000 00120000 *................................*
00C0  00000000 00000800 000007F9 00000000   00000000 00000000 00000000 00000000 *...........9....................*
00E0  A54E6DEF 570E8C04 00010001 00000001   00000000 00000000 00000000 00000000 *................................*
0100  0001A800 00000000 00000000 00000005   0006C300 002F0327 3010200E F1F1F1F1 *..................C.........1111*
0120  F1F10000 80010000 00000000 08000001   B0000000 08000012 00010000 02000000 *11..............................*
0140  00001400 01000100 01000100 03000300   00000000 01AFFF00 00000000 00000000 *................................*
0160  00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000 *................................*
0180  00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000 *................................*
01A0  00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000 *................................*
01C0  00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000 *................................*
01E0  00000000 00000000 00000000 00000000   00000000 00000000 000001F9 01F90000 *.........................9.9..*
ASSOCIATED HKR REC FOLLOWS:
NAME = KSDS2                                    CI = 00002F
```

Figure 75. Example: Incorrect Association Group Occurrence

Output of a Check

Figure 76 shows catalog data that was produced by a run of IKQVCHK against a catalog named VSAM.MCAT.

```
 DATA ABOUT VSAM CATALOG - VSAM.MCAT
=============================================================
 THIS CATALOG CONTAINS    USED HKR-RECORDS           60    See 1.
                          FORMATTED RECORDS          87    See 2.
                          - CCR RECORDS          L    1
                          - FREE RECORDS         F    2
                          - GDG BASE ENTRIES     B    0
                          - CLUSTER RECORDS      C   14
                          - INDEX RECORDS        I   11
                          - DATA RECORDS         D   21
                          - VOLUME RECORDS       V    1
                          - VOLUME EXT. RECORDS  W    4
                          - NONVSAM RECORDS      A    0
                          - EXTENSION RECORDS    E   14
                          - AIX RECORDS          G    7
                          - PATH RECORDS         R    7
                          - ALIAS RECORDS        X    0
                          - UPGRADE RECORDS      Y    4
                          - USER-CAT RECORDS     U    1
                          - UNAVAILABLE RECORDS  *    0
  ==========================================================
   NO ERRORS WERE FOUND IN THIS CATALOG                    See 3.
```

Figure 76. Example: Output from the Catalog Check Service Aid (IKQVCHK)

Notes:

1. **High-Key-Range Catalog Records**

   Each *USED HKR-RECORD* is a 47-byte "true name record". It relates the NAME (as specified in DEFINE CLUSTER or AIX), or the volume number (as specified in DEFINE SPACE) to the CI number of the catalog record that describes the object.

2. **Low-Key-Range Catalog Records**

   The output line *FORMATTED RECORDS* shows the sum of formatted records in the low-key range of the catalog (it equals the total of all fields below the output line). Every low-key-range record occupies a full CI.For explanations to the *formatted records*, refer to Figure 77 on page 350.

3. Depending on the circumstances, one of the following summary statements appears at the bottom of the catalog data:

   NO ERRORS WERE FOUND IN THIS CATALOG
   MINOR ERRORS WERE FOUND AND CORRECTED IN THIS CATALOG
   SERIOUS ERRORS WERE FOUND IN THIS CATALOG

   The statement is preceded by messages that identify the catalog errors that were found.

Record Types and Catalog Identifiers

In the output of IKQVCHK (shown in Figure 76 on page 348), every record type is followed by its one-letter catalog ID.

Record Type	Code	Description
CCR RECORDS	L	Catalog control records keep track of which CIs are allocated. When VSE/VSAM needs space for a catalog record, the CCR indicates that space will be taken from the unformatted section or from a list of CIs that are no longer in use.
FREE RECORDS	F	Free record; the CI in which it resides can be reused by another kind of catalog record on subsequent DEFINEs. Free records are records that have been used and made available again. Records that have not yet been used are not free records.
GDG BASE ENTRIES	B	Generation data group; applies to MVS objects only.
CLUSTER RECORDS	C	There is one cluster record for every cluster defined in the catalog.
INDEX RECORDS	I	An index record describes the index component of a cluster or AIX. There is one index record for every KSDS cluster or AIX defined in the catalog.
DATA RECORDS	D	A data record describes the data component of a cluster or AIX. There is one data record for every cluster or AIX defined in the catalog. The number of data records should equal the number of cluster records (C) plus the number of AIX (G) records.
VOLUME RECORDS	V	A volume record describes the following: • VSE/VSAM data space on the volume; • Every component residing in VSE/VSAM data space on the volume; • Any available space in that data space. There is one volume record for every volume owned by the catalog.
VOLUME EXTENSION RECORDS	W	A volume extension record provides the same data as a volume record. When a volume record becomes full, a volume extension record is created for it. There are as many volume extension records as necessary to contain overflow information.
NONVSAM RECORDS	A	A nonVSAM record describes a nonVSAM file. There is one nonVSAM record for every nonVSAM file defined in the catalog.
EXTENSION RECORDS	E	An extension record contains overflow information from another catalog record (except type V or F). There are as many extension records as necessary to contain overflow information.
AIX RECORDS	G	An AIX record describes an alternate index. There is one AIX record for every alternate index defined in the catalog.
PATH RECORDS	R	A path record describes a path. There is one path record for every path defined in the catalog.
ALIAS RECORDS	X	Alias; applies to DFSMSdfp objects only.
UPGRADE RECORDS	Y	An upgrade record describes a set of alternate indexes that are to be upgraded (kept up to date) when their base cluster is modified.
USER-CAT RECORDS	U	A user cat record describes a user catalog. The master catalog contains one user cat record for every user catalog defined in it.
UNAVAILABLE RECORDS		An unavailable record is one that exists but is inaccessible because one or more pointers to it were destroyed.

Figure 77. Low-Key-Range Catalog Records and Codes

SNAP Dump (IKQVEDA)

The IBM support representative may ask you to run a SNAP dump to provide information for problem diagnosis. You can enable any of the following *types* of SNAP dump:

Type: Enables:

0001

- Catalog management error code trace
- Compression control services trace
- Compression management services trace for OPEN and CLOSE

0002 Buffer manager trace

0003 CLOSE control block dump (at the beginning of CLOSE processing)

0004 VSE/VSAM I/O trace

0005 I/O error trace

0006 OPEN control block dump (when OPEN processing is complete)

0007 OPEN error trace (prints control blocks if an error occurs during OPEN processing)

0008 Catalog management I/O trace (prints all I/O operations done by VSE/VSAM catalog management)

0009 Record management error trace (prints control blocks for any error detected by VSE/VSAM record management)

0010 Automatic CLOSE.

VSE/VSAM is shipped with this SNAP enabled. To disable automatic CLOSE, disable this SNAP.

0011 Managed-SAM control block trace

0012 SHAREOPTIONS(4) z/VSE locking activity trace

0013 In-core wrap trace for the last sixty file access activities for a file

0015 Compression management services control block trace

0016 Compression management services trace

SNAP Dump Number	Output
0001	To SYSLOG
0002 - 0009	To SYSLST
0010	No Output
0011	To SYSLST
0012	To SYSLST
0013	In-Core
0015	To SYSLST
0016	To SYSLOG

With the MSHP PATCH command, the trace output can be limited to specific elements. Your IBM support representative can give you further details.

How to Run a SNAP Dump

Activating a SNAP Dump

To activate IKQVEDA from the system console (SYSLOG) or SYSRDR, enter:

```
// EXEC IKQVEDA,PARM='SYSnnn'
```

where SYSnnn specifies the LU from which the SNAP commands are entered:

```
SYSLOG - SNAP commands are entered from the system console
         (this is the default if PARM is omitted)
SYSIPT - SNAP commands are read from SYSIPT
```

PARM is optional. If the input is to be entered from the console, the system responds with:

```
ENTER FUNCTION ENABLE | DISABLE | END | HELP
```

Enabling a SNAP Dump

To enable the SNAP dump, enter:

```
ENABLE SNAP=00nn,PART=yy
```

where yy is the partition (BG or Fn) in which the SNAP is to be enabled. If PART is omitted, the SNAP is enabled for the issuing partition.

Note that the SNAP dump becomes effective immediately (without re-IPL), and only for the partition which you have specified (or defaulted).

Then the system prints:

```
SNAP 00nn ENABLED IN PARTITION yy
ENTER FUNCTION ENABLE | DISABLE | END | HELP
```

To keep the SNAP option that you just selected (ENABLE) in effect, and to go on to run your program, enter:

```
END
```

If you want to activate the HELP function (which produces explanatory messages on the console), enter:

```
HELP
```

Note that for a *dynamic* partition, the trace must be enabled while the VSE/POWER job that is to be traced is already active. At the end of the VSE/POWER job, the SNAP traces will be lost.

Notes:

1. Input from SYSIPT: If the input is to be read from SYSIPT, you must prepare your input records in advance and place them on SYSIPT before the EXEC command is entered (either on SYSLOG or SYSRDR). The format of these records must be the same as described above for the console input. The last command must be END. All messages are issued on SYSLOG.
2. After *enabling* or *disabling* a SNAP dump, the message IKQ0082I is issued. The message lists all SNAP dumps that are currently enabled for the partition.
3. If you attempt to disable SNAP dump 0010, a warning message is issued.

   SNAP dump 0010 is used to control VSE/VSAM automatic CLOSE processing. It should only be disabled after discussion with your IBM support representative.

Disabling a SNAP Dump

1. To disable the SNAP dump after the program is finished, enter:

   ```
   // EXEC IKQVEDA,PARM='SYSnnn'
   ```

   where SYSnnn is either SYSLOG (default) or SYSIPT. PARM is optional.

   The system responds with:

   ```
   ENTER FUNCTION ENABLE | DISABLE | DISABLE | END | HELP
   ```
2. Enter:

```
DISABLE SNAP=00nn,PART=yy
```

PART is optional. The system responds with:

```
SNAP 00nn DISABLED IN PARTITION yy
ENTER FUNCTION ENABLE | DISABLE | END | HELP
```

3. Enter:

```
END
```

Example: SNAP Dump 0001

The following example shows the use of IKQVEDA to enable SNAP 0001 as a batch job in the BG partition.

Input (from SYSIN):

```
// JOB EXAMPLE
// EXEC IKQVEDA,PARM='SYSIPT'
ENABLE SNAP=01
END
/*
/&
```

Output (on SYSLOG):

```
// JOB EXAMPLE ...
IKQ0082I SNAP 0001 ENABLED IN PARTITION BG
EOJ EXAMPLE ...
```

Activating

You must now activate SNAP 0001 in your program. Do this by including one of three UPSI statements in your job stream.

The UPSI job control statement specifies under which conditions the SNAP code should be executed. Because user programs can also use the UPSI byte, make sure that there is no conflict between the UPSI setting for SNAP and the UPSI requirements for any program running in the same partition. Using UPSI, you can specify that the error symptom message be printed under one of the following conditions. If you do not specify an UPSI value, 00000000 is the default.

// UPSI 0
When the catalog management return code is not 0, 40, 68, or 160, or if LISTCAT issued a return code not equal to 8. (These codes occur during normal processing.)

// UPSI 1
When the catalog management return code is not zero.

// UPSI 11
For all catalog management return codes, including zero.

// UPSI 01
Same as 1 but with operator reply required.

// UPSI 011
Same as 11 but with operator reply required.

// UPSI 0111
Same as 0 but with operator reply required.

// UPSI 00001
Compression control services trace requires operator reply when a request failed.

// UPSI 000011
Compression control services trace requires operator reply.

SNAP 0001 Output

SNAP 0001 prints the following *error symptom string* at the console (SYSLOG):

```
nnn,mn,rrr,ffff,ccc
```

The error symptom string is preceded by the partition identifier (for example, BG, F1, F2).

The meaning of the string elements:

nnn Is the Catalog Management return code (decimal).

mn Are the last two characters of the name of the module that encountered the error (IGG0CLmn).

rrr Is the Catalog Management reason code (decimal). The return and reason codes are documented under the "IDCAMS Codes" in the *z/VSE Messages and Codes*.

ffff Indicates which of the following functions was processed:
ALT - alter
DEF - define VSE/VSAM object
DEFA - define nonVSAM file
DEFC - define catalog
DEFS - define space
DEL - delete VSE/VSAM object or nonVSAM file
DELC - delete catalog
DELS - delete space
LOC - locate
LSTC - list catalog
UPD - update or update-extend

ccc Is one of the following:
- The number of the CI associated with the function (decimal)
- The object name, full length
- The volume number in EBCDIC

Figure 78 on page 355 is an example of the console listing for a job in which a cluster and two alternate indexes are defined:

```
Control Statement                    Meaning
================================================================================
// UPSI 11000000                     Print error symptom string for all VSE/VSAM
                                     return codes.
// EXEC IDCAMS,SIZE=AUTO


Error Symptom String                    Meaning
================================================================================
BG  000,  ,000,LOC ,000000              Locate for CI 000000.
BG  000,  ,000,LOC ,MASTER              Locate for master catalog volume.
BG  000,  ,000,LOC ,000001              Locate for CI 000001.
BG  000,  ,000,LOC ,000001
BG  008,CG,006,DEL ,V3V003.KSDS         Before defining V3V003.KSDS, user
                                        issues a DELETE to make sure that the
                                        object does not already exist (it doesn't).
BG  008,CG,006,LOC ,DEFAULT.MODEL.KSDS  Before attempting to define the new
                                        cluster, VSE/VSAM looks for an optional
                                        default model.  No default model exists.
BG  000,  ,000,DEF ,V3V003.KSDS         VSAM defines the new cluster.
BG  008,CG,006,LOC ,DEFAULT.MODEL.AIX   Before attempting to define the new AIX,
                                        VSE/VSAM looks for an optional
                                        default model.  No default model exists.
BG  000,  ,000,DEF ,V3V003.KSDS.AIX1    VSAM defines the first AIX.
BG  008,CG,006,LOC ,DEFAULT.MODEL.AIX
BG  000,  ,000,DEF ,V3V003.KSDS.AIX2    VSAM defines the second AIX.
```

Figure 78. Example: SNAP Dump Output

Maintaining VTOC and VOL1 Labels on Disk (IKQVDU)

The utility IKQVDU assists in maintaining the VTOC and VOL1 labels on disk devices.

Note: This utility changes information in the VTOC only; it does not change catalog entries. If you want to redefine a VSE/VSAM data space or UNIQUE file, you must first issue a DELETE command to erase the file's catalog information.

If you want conceptual information on labels, refer to Appendix E, "VSE/VSAM Labels," on page 333.

How to Run the IKQVDU

The following procedures should be followed to use IKQVDU at the *system console* for such maintenance. The key difference in the three procedures is the presence or absence of a // UPSI job control statement.

PROCEDURE 1 **Type in the following, then press ENTER**	**Explanation**
1. // ASSGN SYS000,X'cuu'	For *cuu*, type in the address of the disk drive whose volume is to be accessed.
2. // UPSI 1	This job control statement is optional. If it is included, the following events take place on the volume that was assigned to SYS000: • The VSE/VSAM volume ownership bit and catalog recovery area (CRA) pointer in the F4 VTOC label are reset. The ownership bit is reset regardless of how many catalogs own space on the volume. • The entire VTOC is scratched, that is, empty VTOC labels are written over existing F1, F2, and F3 labels, except for labels that have names starting with the characters "DOS.", "VSE.", or "PAGE". • An operator authorization prompt is issued if the VTOC label to be scratched is security protected.
3. // EXEC IKQVDU,SIZE=AUTO	Start execution of the IKQVDU phase. Then run an IDCAMS DELETE SPACE command to delete the catalog information for the volume you just scratched. Specify the FORCE parameter if necessary.

PROCEDURE 2 **Type in the following, then press ENTER**	**Explanation**
1. // ASSGN SYS000,X'cuu'	For *cuu*, type in the address of the disk drive whose volume is to be accessed.
2. // UPSI 11	This job control statement is optional. If it is included, the following events take place on the volume that was assigned to SYS000: • The VSE/VSAM volume ownership bit and CRA pointer in the F4 label are reset. The ownership bit is reset regardless of how many catalogs own space on the volume. • The entire VTOC is scratched, that is, F0 labels are written over existing F1, F2, and F3 labels, except for labels that have names starting with the characters "DOS.", "VSE.", or "PAGE".
3. // EXEC IKQVDU,SIZE=AUTO	Start execution of the IKQVDU phase. Then run an IDCAMS DELETE SPACE command to delete the catalog information for the volume you just scratched. Specify the FORCE parameter if necessary.

PROCEDURE 3 Type in the following, then press ENTER	Explanation
1. // ASSGN SYS000,X'cuu'	For *cuu*, type in the address of the disk drive whose volume is to be accessed.
2. // EXEC IKQVDU,SIZE=AUTO	Start execution of the IKQVDU phase.
SPECIFY FUNCTION OR REPLY '?' FOR OPTIONS READY	• You can specify the functions listed in Figure 80 on page 358. If you specify a function, the list shown in Figure 79 is *not* displayed. • You can enter the character **?** This displays (at the system console) a list of the functions that IKQVDU can perform; refer to Figure 79.

```
TO SET THE VOLUME OWNERSHIP FLAG REPLY 'SET OWNERSHIP'
TO SET THE CRA POINTER REPLY 'SET OWNERSHIP'
TO RESET THE VOLUME OWNERSHIP FLAG AND CRA POINTER REPLY
    'RESET OWNERSHIP' OR 'RESET CRA'
TO SET THE SECURITY FLAG IN A F1 LABEL REPLY 'SET
     SECURITY'
TO RESET THE SECURITY FLAG IN A F1 LABEL REPLY 'RESET
     SECURITY'
TO REMOVE A LABEL FROM THE VTOC REPLY 'SCRATCH'
TO RENAME A LABEL REPLY 'RENAME'
TO ALLOCATE A LABEL REPLY 'ALLOCATE'
TO REINITIATE PROCESSING REPLY 'RESTART'
TO ALTER OR DISPLAY A disk VOL1 LABEL
REPLY 'CLIP LABEL=SER=N..N' OR 'CLIP LABEL=DISPLAY'
TO TERMINATE PROCESSING REPLY 'END'
READY
```

Figure 79. Display of IKQVDU Functions

Function Enter one of the following, then press ENTER	Explanation
SET OWNERSHIP	Causes the VSE/VSAM ownership bit to be set in the F4 VTOC label and optionally allows the user to set the CRA pointer.
RESET CRA or RESET OWNERSHIP	Causes the VSE/VSAM ownership bit and CRA pointer to be reset in the F4 VTOC label. The ownership bit is reset regardless of how many catalogs own space on the volume.
SET SECURITY	Causes the security bit to be set in the F1 VTOC label. When the console responds with ENTER DSN, reply with the data set name of the VTOC label to be modified.
RESET SECURITY	Causes the security bit in the F1 label to be reset. When the console responds with ENTER DSN, reply with the data set name of the VTOC label to be modified.
SCRATCH DSN=dsname	Causes the VTOC label with the specified file name to be scratched. If the file is a VSE/VSAM data space or a UNIQUE VSE/VSAM file, run an IDCAMS DELETE SPACE or DELETE CLUSTER command to delete the catalog information for the object(s) you just scratched. Specify the FORCE parameter if necessary.
SCRATCH VTOC	Causes the entire VTOC to be scratched except for file names starting with the characters “DOS.”, “VSE.”, and “PAGE”. In addition, an operator-authorization prompt will be issued if the VTOC label is security-protected or describes a catalog. If the VTOC contained VSE/VSAM data spaces, run an IDCAMS DELETE SPACE command to delete the catalog information for the volume you just scratched. Specify the FORCE parameter if necessary.
RENAME	Causes the DSNAME portion of the F1 VTOC label to be changed. When the console responds with ENTER OLD DSN, reply with the file name of the VTOC label to be changed. Be sure to enter the correct OLD DSN. No error checking is performed if an invalid name is specified. When the console responds with ENTER NEW DSN, reply with the new file name.

Figure 80. Explanation to IKQVDU Functions (Part 1 of 2)

Function Enter one of the following, then press ENTER	Explanation
ALLOCATE	Causes a new label to be created and written in the VTOC. To use this function, a DLBL/EXTENT job control statement must be provided. When the console responds with ENTER FILENAME, reply with the same file name as that in the DLBL statement referred to above. When the console responds with ENTER NEW DSN, reply with the file name of the file to be created. When the console responds with DO YOU WISH TO SECURITY PROTECT THIS DATA SET? reply YES or NO. A reply of YES causes the data security bit to be set in the F1 VTOC label. A reply of NO causes the data security bit to be reset.
RESTART	Causes processing to be reinitiated with a READY prompt. This keyword can be used as a response to any operator prompt.
CLIP LABEL=DISPLAY	Causes the volume serial number to be displayed on the system console.
CLIP LABEL=SER=n..n	Causes the existing volume serial number to be changed to the one specified as n..n.
END	Causes processing to terminate.

Figure 80. Explanation to IKQVDU Functions (Part 2 of 2)

Error Message and Codes (from IKQVDU)

If an error occurs during execution of IKQVDU, a message of the following format is displayed at the system console:

```
ERROR** DADSM RETURN CODE IS nnn condition

where:
   nnn        is the code (for example: 020)
   condition  describes the problem (for example: VTOC FULL)
```

The following shows the code (nnn), the associated condition, and the action required to correct the condition.

004 I/O ERROR WHILE READING VOLUME LABEL
Action: If the problem was not caused by a hardware error, restore the volume.

008 VOLUME NOT MOUNTED
Action: Mount the correct volume.

012 I/O ERROR ON VTOC
Action: If the problem was not caused by a hardware error, restore the volume.

016 DUPLICATE NAME ON VOLUME
Action: Choose another file name or scratch the original file from the volume. If duplication is because of key ranges, ensure every UNIQUE key range is on a separate volume.

020 VTOC FULL
Action: Delete any nonVSAM files or VSE/VSAM data spaces no longer needed from the volume to make additional Format 1 labels available, or reinitialize the volume with a larger VTOC.

024 EXTENT OVERLAPS EXPIRED FILE
Action: Examine the VTOC listing to determine where the overlap

occurred. Correct the EXTENT statement causing the error. To delete the expired file, open a DTF using the same file-ID as that of the expired file, and instruct the operator to reply DELETE to message 4n33A when it is issued.

028 EXTENT OVERLAPS UNEXPIRED FILE
Action: Compare the high and low extent limits on the EXTENT statement or LSERV output with the file or data space limits on the VTOC display. If the extents overlap, correct the EXTENT statement in error.

032 EXTENT OVERLAPS PROTECTED UNEXPIRED FILE
Action: Examine the VTOC to determine where the overlap occurred. Correct the EXTENT statement causing the error. If necessary, use another volume.

036 EXTENT OVERLAPS VTOC
Action: Execute LVTOC. The Format 4 label (the first label in the VTOC display) contains the extent limits of the VTOC. If the program executed uses a temporary label set and overlaps the VTOC, correct the EXTENT statements that overlap. If the job uses standard or partition standard labels, use the LSERV output to correct the extents of the overlapping file, VSE/VSAM data space, or UNIQUE VSE/VSAM file. Then rebuild the appropriate label tracks.

040 REQUIRED EXTENTS MISSING
Action: If temporary labels were used, match the extents on the incoming EXTENT card with the extents in the LVTOC output. If standard (permanent) labels were used, match the extents in the LSERV output with those in the LVTOC output.

044 LABEL NOT FOUND
Action: Use the LVTOC output to check for all file labels used in OPEN macros. If the file has been destroyed, it was probably because of deletion of overlapping extents on an unexpired file, and the file must be rebuilt.

048 INVALID LABEL ADDRESS
Action: Examine the VTOC for a label having an invalid forward chain pointer, and delete it. If no invalid labels are found, just rerun the job.

056 EXTENT OVERLAPS PROTECTED EXPIRED FILE
Action: Examine the VTOC listing to determine where the overlap occurred. Correct the EXTENT statement causing the error. If it is not necessary to save the expired file, open a DTF using the same file-ID as that of the expired file, and instruct the operator to reply DELETE to message 4n33A when it is issued.

064 GETVIS FAILURE ENCOUNTERED
Action: Allocate GETVIS area. If VSE/VSAM is running in the SVA, re-IPL and specify a new value for SET SVA. If VSE/VSAM is running in a partition, rerun the job in a larger partition.

072 CDLOAD FAILURE ENCOUNTERED
Action: Either the CDLOAD directory or the GETVIS area is full. Allocate more space.

080 OVERLAP AMONG NEW EXTENTS
Action: If DLBL and EXTENT statements are included in the program, determine the conflicting extents and correct them. If a standard label set is used, use the LSERV output to locate and correct the conflicting file extents, and rebuild the standard label tracks.

088 FORMAT 4 LABEL NOT FOUND
Action: Reinitialize the VTOC to create a format-4 label.

092 VOL1 LABEL NOT FOUND
Action: Reinitialize the volume to create a VOL1 label.

096 JIB PROCESSING FAILURE
Action: Rerun the job when more JIBs are available.

Glossary

ir This glossary includes terms and definitions related primarily to VSE/VSAM. If you do not find the term you are looking for, refer to the index of this book, or to the *IBM Dictionary of Computing* , SC20-1699.

The glossary includes definitions with:

- Symbol * where there is a one-to-one copy from the *IBM Dictionary of Computing*
- Symbol (A) from the *American National Dictionary for Information Processing Systems* copyright 1982 by the Computer and Business Equipment Manufacturers Association (CBEMA). Copies may be purchased from the American National Standards Institute, 1430 Broadway, New York, New York 10018. Definitions are identified by symbol (A) after definition.
- Symbols (I) or (T) from the *ISO Vocabulary - Information Processing* and the *ISO Vocabulary - Office Machines* developed by the International Organization for Standardization, Technical Committee 97, Subcommittee 1. Definitions of published sections of the vocabularies are identified by symbol (I) after the definition; definitions from draft international standards, draft proposals, and working papers in development by the ISO/TC97/SC1 vocabulary subcommittee are identified by symbol (T) after the definition, indicating final agreement has not yet been reached among participating members.

The part of speech being defined is indicated by the opening words of the descriptive phrase: "To ..." indicates a verb and "Pertaining to ..." indicates a modifier. Any other wording indicates a noun or noun phrase.

Numerics

31-bit addressing. Provides addressability for address spaces of up to 2 gigabytes.

A

access method. A program (such as VSE/VSAM or VTAM) that allows the user to define files and addresses, and to move data to and from them. It is a technique for moving data between main storage and input/output devices.

address. 1. The location in the storage of a computer where data are stored.

2. In data communication, the unique code assigned to every device or work station connected to a network.

addressing mode (AMODE). A program attribute that refers to the address length that a program is prepared to handle on entry. Addresses may be either 24 bits or 31 bits in length. In 24-bit addressing mode, the processor treats all virtual addresses as 24-bit values; in 31-bit addressing mode, the processor treats all virtual addresses as 31-bit values.

address space. A range of up to two gigabytes of contiguous virtual storage addresses that the system creates for a user. Unlike a data space, an address space contains user data **and** programs, as well as system data and programs, some of which are common to all address spaces. Instructions execute in an address space (not a data space).

AIX. See *alternate index*.

alternate index (AIX). In systems with VSE/VSAM, the index entries of a given base cluster organized by an alternate key, that is, a key other than the prime key of the base cluster. For example, a personnel file primarily ordered by names can be indexed also by department number.

*** alternate tape.** A tape drive to which the operating system switches automatically for tape read or write operations if the end of the volume has been reached on the originally used tape drive.

alternate track. On a direct access storage device, a track designated to contain data in place of a defective track.

AMODE. See *addressing mode*.

application program. A program written for or by a user that applies to the user's work, such as a program that does inventory control or payroll.

ASI. See *automated system initialization*.

assemble. To translate a program from assembler language into object code.

assembler. A computer program that converts assembly language instructions into object code.

© Copyright IBM Corp. 1979, 2005

*** assembler language.** A source language that includes symbolic machine language statements in which there is a on-to-one correspondence with the instruction formats and data formats of the computer.

*** automated system initialization (ASI).** A function that allows control information for system startup to be cataloged for automatic retrieval during system startup.

*** autostart.** In z/VSE, a facility that starts-up VSE/POWER with little or no operator involvement.

*** auxiliary storage.** All addressable storage, other than main storage, that can be accessed by an input/output channel; for example, storage on magnetic tape or direct access storage devices. Synonymous with external storage and with secondary storage.

B

*** backup copy.** A copy of information or data that is kept in case the original is changed or destroyed.

batch processing. 1. Serial processing of computer programs. 2. Pertaining to the technique of processing a set of computer programs in such a way that each is completed before the next program of the set is started. (A)

batch program. A program that is processed in series with other programs and therefore normally processes data without user interaction.

block. Usually, a block consists of several records of a file that are transmitted as a unit. But if records are very large, a block can also be part of a record only. On an FBA disk, a block is a string of 512 bytes of data. In the (E)CKD environment, blocks are called records. See also *control block*.

buffer. An area of storage temporarily reserved for input or output operations; an area into which data is read or from which data is written. Synonymous with I/O area.

byte. Eight adjacent binary digits that are operated upon as a unit and that constitute the smallest addressable unit of information within a computer system. Normally, it represents a stored character.

C

catalog. A directory of files and libraries, with reference to their locations. A catalog may contain other information such as the types of devices in which the files are stored, passwords, and blocking factors. (I) (A)

See also *VSE/VSAM master catalog*.

catalog recovery area (CRA). In systems with VSE/VSAM, an entry-sequenced data set that exists on each volume owned by a recoverable catalog, including the catalog volume itself. The CRA contains copies of the catalog records and can be used to recover a damaged or invalid catalog. As of z/VSE 3.1, the CRA is no longer supported.

channel program. One or more channel command words that control a sequence of data channel operations. Execution of this sequence is initiated by a single start I/O (SIO) instruction.

CA. See *control area*.

CI. See *control interval*.

CKD. See *count-key-data (CKD) device*.

close. The function that ends the connection between a file and a program, and ends the processing. Contrast with open.

cluster. In systems with VSE/VSAM, a named structure consisting of a group of related components; for example, a data component with its index component.

component. 1. Hardware or software that is part of a computer system. 2. A functional part of an operating system, for example: job control program, VSE/POWER. 3. In VSE/VSAM, a named, cataloged group of stored records, such as the data component or index component of a key-sequenced file or alternate index.

conditional job control. The capability of the job control program to process or to skip one or more statements based on a condition that is tested by the program.

configuration. The devices and programs that make up a system, subsystem, or network.

control area (CA). In VSE/VSAM, a group of control intervals (CIs) used as a unit for formatting a data set before adding records to it. Also, in a key-sequenced data set, the set of control intervals pointed to by a sequence-set index record (which, for example, is used by VSE/VSAM for distributing free space).

control block. An area within a program or a routine defined for the purpose of storing and maintaining control information.

control interval (CI). A fixed length area of disk storage where VSE/VSAM stores records and creates distributes free space. It is the unit of information that VSE/VSAM transfers to or from disk storage.

count-key-data (CKD) device. A disk storage device that stores data in the record format: count field, key field, data field. The count field contains, among others, the address of the record in the format: cylinder, head (track), record number and the length of the data field. The key field, if present, contains the record's key

(search argument). CKD disk space is allocated by tracks and cylinders. Contrast with fixed-block-architecture (FBA) device.

CRA. See *catalog recovery area*.

D

DASD. See *direct access storage device*.

Data Facility Product (DFP). See *DFSMSdfp*.

data import. The process of reformatting data that was used under one operating system (for example, IBM System/3) such that it can subsequently be used under a different operating system (for example, the IBM z/VSE system).

data management. A major function of the operating system. It involves organizing, storing, locating, and retrieving data.

data set. The major unit of data storage and retrieval, consisting of a collection of data in one or several prescribed arrangements and described by control information to which the system has access.

See also *file*.

*** default value.** A value assumed when no value has been specified. Synonymous with assumed value.

device. A hardware component of a computer system with a specific purpose.

device address. 1. The identification of an input/output device by its channel and unit number. 2. In data communication, the identification of any device to which data can be sent or from which data can be received.

*** device class.** The generic name for a group of device types, for example, all display stations belong to the same device class. Contrast with device type.

*** device type.** The name of a particular kind of device; for example, 9345 (IBM DASD module), 3420 (IBM magnetic tape unit). Contrast with device class.

DFP. See *DFSMSdfp*.

DFSMSdfp. DFSMSdfp (formerly known as DFP) provides data management, device support, program library management, utility functions, and support for the z/OS operating system.

direct access. Accessing data on a storage device using their address and not their sequence. This is the typical access on disk devices as opposed to magnetic tapes. Contrast with sequential access.

*** direct access storage.** A storage device that provides direct access to data. (1)

*** direct access storage device (DASD).** A device in which access time is effectively independent of the location of the data. See also *fixed-block-architecture (FBA) device*.

directory. 1. A table of identifiers and references to the corresponding items of data. (I) (A) 2. In z/VSE, an index that is used by the system to locate one or more sequential blocks of program information that are stored on direct access storage.

disk. Synonymous for magnetic disk.

diskette. A thin, flexible magnetic disk and a semi-rigid protective jacket, in which the disk is permanently enclosed.

disk sharing. An option that lets independently operating computer systems to jointly use common data residing on shared disk devices.

dump. 1. To record, at a particular instant, the contents of all or part of one storage device in another storage device. Dumping is usually for the purpose of debugging. (T) 2. Data that has been dumped. (T) 3. To copy data in a readable format from main or auxiliary storage onto an external medium such as tape, diskette, or printer. 4. To copy the contents of all or part of virtual storage for the purpose of collecting error information.

E

Enterprise Storage Server (ESS). An IBM disk storage system designed for performance, automation, integration, and continuous data availability. It supports advanced copy functions, which can be critical for increasing data availability by providing important disaster recovery and backup protection.

*** entry-sequenced file.** A VSE/VSAM file whose records are loaded without respect to their contents and whose relative byte addresses cannot change. Records are retrieved and stored by addressed access, and new records are added to the end of the file.

entry-sequenced data set (ESDS). An entry-sequenced file under VSE/VSAM. Its records are loaded without respect to their contents, and whose relative byte addresses cannot change. Records are retrieved and stored by addressed access, and new records are added to the end of the file. See also *SAM ESDS file*.

exit. 1. To execute an instruction within a portion of a computer program in order to terminate the execution of that portion. 2. See *user exit routine*.

exit routine. 1. Either of two types of routines: installation exit routines or user exit routines. Synonymous with exit program. 2. See *user exit routine*.

extended-addressed KSDS. A KSDS exceeding 4 GB in size.

extent. Continuous space on a disk or diskette that is occupied by or reserved for a particular file or VSE/VSAM data space.

*** external storage.** Storage that is accessible to a processor only through input-output channels. An external storage may sometimes be considered as peripheral equipment. Synonymous with auxiliary storage. (T)

F

FBA. See *fixed-block-architecture (FBA) device*.

FBA block. A unit of data that is transported as a unit on FBA disk devices.

Fibre Channel Protocol (FCP). A combination of hardware and software conforming to the Fibre Channel standards and allowing system and peripheral connections via FICON and FICON Express feature cards on IBM zSeries processors. In z/VSE, zSeries FCP is employed to access industry-standard SCSI disk devices.

file. A named set of records stored or processed as a unit. (T) Synonymous with data set.

fixed-block-architecture (FBA) device. A disk storage device that stores data in blocks of fixed size. These blocks are addressed by block number relative to the beginning of the particular file. Contrast with count-key-data (CKD) device.

G

generation. See *macrogeneration*.

generic name. The initial characters common to names, that can be used to identify a group of items. A generic name usually ends with an *; for example, ORD* identifies all items whose names begin with the characters ORD.

I

index. 1. A table used to locate records in an indexed sequential data set or an indexed file. 2. In VSE/VSAM, an ordered collection of pairs, each consisting of a key and a pointer, used by VSE/VSAM to sequence and locate the records of a key-sequenced data set or file; it is organized in levels of index records. See also *alternate index*.

initial program load (IPL). The process of loading system programs and preparing the system to run jobs.

input. 1. Pertaining to a functional unit or channel involved in an input process, or to the data involved in such process. 2. Loosely, input data, input process. 3. Information or data to be processed.

input/output (I/O). See *input* and *output*.

IPL. See *initial program load*.

J

JCL. See *job control language*.

job. One program, or a group of related programs called job steps, complete with the JCL statements necessary for a particular run. A job is identified in the job stream by a JOB statement followed by one EXEC statement for each of the programs or job steps.

job catalog. A catalog made available for a job by using the file name IJSYSUC in the respective DLBL statement.

job control language (JCL). A language that serves to prepare a job or each job step of a job to be run. Some of its functions are: to identify the job, to determine the I/O devices to be used, set switches for program use, log (or print) its own statements, and fetch the first phase of each job step.

job control statement. A particular statement of JCL.

job step. One of a group of related programs complete with the JCL statements necessary for a particular run. Every job step is identified in the job stream by an EXEC statement under one JOB statement for the entire job.

job stream. The sequence of jobs as submitted to an operating system.

K

*** KB (kilobyte).** 1024 bytes of storage.

key. In VSE/VSAM, one or several characters taken from a certain field (key field) in data records for identification and sequence of index entries or of the records themselves.

key sequence. The collating sequence of either records themselves or of their keys in the index, or the collating sequence of records and keys. The key-sequence is alphanumeric.

key-sequenced data set (KSDS). Under VSE/VSAM, a key-sequenced file. See *key-sequenced file*.

key-sequenced file. A VSE/VSAM file whose records are loaded in key sequence and controlled by an index.

Records are retrieved and stored by keyed access or by addressed access, and new records are inserted in the file in key sequence.

KSDS. See *key-sequenced data set*.

L

* **label.** A record that identifies a volume on tape, disk, or diskette, or that identifies a file on the volume.

large DASD. A DASD device that (1) has a capacity exceeding 64K tracks and (2) does not have VSAM space created prior to VSE/ESA 2.6 that is owned by a catalog.

local shared resources (LSR). 1. An option for sharing I/O buffers, I/O-related control blocks, and channel programs among VSE/VSAM data sets in a resource pool that serves one partition or address space. 2. A VSE/VSAM option activated by the macros BLDVRP, DLVRP, and WRTBFR to share control blocks among files.

logical record. A user record, normally pertaining to a single subject and processed by data management as a unit. Contrast with physical record which may be larger or smaller.

LSR. See *local shared resources*.

M

macro. See *macroinstruction*.

macrodefinition. A set of statements and instructions that defines the name of, format of, and conditions for generating a sequence of assembler statements and machine instructions from a single source statement.

macroexpansion. See *macrogeneration*.

macrogeneration. An operation in which an assembler produces a sequence of assembler language statements by processing a macrodefinition called by a macroinstruction. Macrogeneration takes place before assembly. Synonymous with macroexpansion.

macroinstruction. In assembler programming, an assembler language statement that causes the assembler to process a predefined set of statements called a macrodefinition. The statements normally produced from the macrodefinition replace the macroinstruction in the program.

magnetic tape. A tape with a magnetizable layer on which data can be stored. (T)

maximum (max) CA. A unit of allocation equivalent to the maximum control area size on a count-key-data or fixed-block device. On a CKD device, the max CA is equal to one cylinder.

* **MB (megabyte).** One megabyte equals 1,048,576 bytes.

message. In z/VSE, a communication sent from a program to the operator or user. It can appear on a console, a display terminal, or in a printout.

* **migrate.** To move to a changed operating environment, usually to a new release or version of a system.

minimum (min) CA. A unit of allocation equivalent to the minimum control area size on a count-key-data or fixed-block device. On a CKD device, the min CA is equal to one track.

module. A program unit that is discrete and identifiable with respect to compiling, combining with other units, and loading; for example, the input to, or output from an assembler, compiler, linkage editor, or executive routine. (A)

O

online processing. Processing by which the input data enters the computer directly from a display station and the output data is transmitted directly to the display station.

open. To connect a file or a library to a program for processing. Contrast with close.

* **operating system.** Software that controls the execution of programs and that may provide services such as resource allocation, scheduling, input/output control, and data management. Although operating systems are predominantly software, partial hardware implementations are possible. (T)

* **operator command.** A statement to a control program, issued via a console or terminal. It causes the control program to provide requested information, alter normal operations, initiate new operations, or end existing operations.

output. 1. Pertaining to a functional unit or channel involved in an output process, or to the data involved in such process. 2. Loosely, output data, output process. 3. Information or data that has been processed.

P

* **partition.** In z/VSE, a division of the virtual address area that is available for program execution.

* **password.** In computer security, a string of characters known to the computer system and a user. The user must specify it to gain full or limited access to the system and to the data stored in it.

path. 1. In ACF/VTAM, the intervening nodes and data links connecting a terminal and an application

program in the host processor. 2. In VSE/VSAM, a named logical entity providing access to the records of a base cluster either directly or through an alternate index.

* **physical record.** The amount of data transferred to or from auxiliary storage. Synonymous with block.

processor. The hardware component that interprets and executes instructions.

* **processor storage.** 1. The storage provided by one or more processing units. 2. In virtual storage systems, synonymous with real storage.

R

* **random processing.** 1. The treatment of data without respect to its location in external storage, and in an arbitrary sequence governed by the input against which it is to be processed. 2. The processing of records in an order other than the order in which they exist in a file.

* **read.** To acquire or interpret data from a storage device, from a data medium, or from another source. (I) (A)

* **real storage.** The main storage in a virtual system. Physically, real storage and main storage are identical. Conceptually however, real storage represents only part of the range of addresses available to the user of a virtual storage system. Traditionally, the total range of addresses available to the user was provided by the main storage. (I) (A)

record. A collection of related data or words, treated as a unit. See *logical record* and *physical record*.

* **recover.** After an execution failure, to establish a previous or new status from which execution can be resumed. (T)

relative-record data set (RRDS). A VSE/VSAM file whose records are loaded into fixed-length slots and accessed by the relative-record numbers of these slots.

residency mode (RMODE). A program attribute that refers to the location where a program is expected to reside in virtual storage.

restore. To write back on disk data that was previously written from disk to an intermediate storage medium such as tape.

RMODE. See *residency mode*.

routine. A program, or part of a program, that may have some general or frequent use. (T)

* **RPG II.** A commercially oriented programming language suitable specifically designed for writing application programs intended for business data processing.

RRDS. See *relative-record data set*.

run. 1. A performance of one or more jobs or programs. (I) (A) 2. To cause a program, utility, or other machine function to be performed.

S

SAM. See *sequential access method*.

SAM ESDS file. A SAM file managed in VSE/VSAM space, so it can be accessed by both SAM and VSE/VSAM macros.

SCSI. A peripheral interface originally introduced for small computers. IBM Enterprise Storage Server (ESS) disks can be accessed via a SCSI interface implemented via zSeries Fibre Channel Protocol (FCP) channels.

SDL. See *system directory list*.

sequential access. The serial retrieval of records in their entry sequence or serial storage of records with or without a premeditated order. Contrast with direct access.

sequential access method (SAM). A data access method that writes to and reads from an I/O device record after record (or block after block). On request, the support performs device control operations such as line spacing or page ejects on a printer, or skip a certain number of tape marks on a tape drive.

sequential file. A file in which records are processed in the order in which they are entered and stored.

service program. A program in general support of computer processes, for example, a diagnostic program, a trace program, or a sort program. (T)

* **shared virtual area (SVA).** In z/VSE, a high address area of virtual storage that contains a system directory list (SDL) of frequently used phases, resident programs that can be shared between partitions, and an area for system support.

* **spanned record.** A logical record contained in more than one block.

split. To double a specified unit of storage space (CI or CA) dynamically when the specified minimum of free space gets used up by new records.

* **standard label.** A fixed-format record that identifies a volume of data such as a tape reel or a file that is part of a volume of data.

startup. The process of performing IPL of the operating system and of getting all subsystems and application programs ready for operation.

storage. A device, or part of a device, that can retain data. See also *auxiliary storage, processor storage, virtual storage*.

* **suballocated file.** A VSE/VSAM file that occupies a portion of an already defined data space. The data space may contain other files. Contrast with unique file.

SVA. See *shared virtual area*.

system directory list (SDL). A list containing directory entries of frequently-used phases and of all phases resident in the SVA. The list resides in the SVA.

T

track. A circular path on the surface of a disk or diskette. Smallest unit of physical disk space.

U

* **unique file.** A VSE/VSAM file that occupies a data space of its own. The data space is defined at the same time as the file and cannot contain any other file. Contrast with suballocated file.

unit of transfer. The amount of data that can be transferred between virtual storage and an I/O device in response to a read or write request.

user exit routine. A user-written routine that receives control at predefined user exit points.

* **utility program.** 1. A computer program in general support of computer processes; for example, a diagnostic program, a trace program, or a sort program. (T) Synonymous with service program. 2. A program that performs an everyday task such as copying data from one storage device to another.

V

variable-length relative-record data set (VRDS). A VSE/VSAM relative-record data set with variable-length records. See also *relative-record data set*.

* **virtual address.** The address of a location in virtual storage. A virtual address must be translated into a real address in order to process the data in processor storage.

* **virtual address area.** The area available as a program address range.

virtual address space. In z/VSE, a subdivision of the virtual address area available to the user for the allocation of private, non-shared partitions.

virtual disk. A range of up to two gigabytes of contiguous virtual storage addresses that a program can use as workspace. Although the virtual disk exists in storage, it appears as a real FBA disk device to the user program. All I/O operations directed to a virtual disk are intercepted and the data to be written to, or read from, the disk is moved to or from a data space.

Like a data space, a virtual disk can hold only user data; it does not contain shared areas, system data or programs. Unlike an address space or a data space, data is not directly addressable on a virtual disk. To manipulate data on a virtual disk, the program has to perform I/O operations.

virtual storage. Addressable space image for the user from which instructions and data are mapped into processor storage locations.

virtual storage access method (VSAM). An access method for indexed or sequential processing of fixed and variable length records on direct access devices. The records in a VSE/VSAM data set or file can be organized: in logical sequence using a key field (key sequence); in physical sequence in which they are written on the data set or file (entry sequence); by using relative record numbers.

These and other functions are provided by the IBM product VSE/VSAM.

volume. A data carrier mounted and demounted as a unit; for example, a reel of magnetic tape, a disk pack. (I) Some disk units have no demountable packs. In that case, a volume is the portion available to one read/write mechanism.

volume identifier. The volume serial number, which is a number in a volume label assigned when a volume is prepared for use by the system. See *volume serial number*.

* **volume serial number.** A number in a volume label assigned when a volume is prepared for use in a system.

volume table of contents (VTOC). A table on a disk volume that describes every file on it.

VRDS. See *variable-length relative-record data set*.

VSE/VSAM. Virtual Storage Extended/Virtual Storage Access Method. See *virtual storage access method*.

VSAM. See *virtual storage access method*.

* **VSE/VSAM managed space.** A user-defined space on disk that is under the control of VSE/VSAM.

* **VSE/VSAM master catalog.** A key-sequenced data set or file with an index containing extensive data set and volume information that VSE/VSAM requires to locate data sets or files, allocate and deallocate storage space, verify the authorization of a program or operator to gain access to a data set or file, and to accumulate use statistics for data sets or files.

*** VSE/VSAM recoverable catalog.** A VSE/VSAM catalog defined with the recoverable attribute, causing duplicate catalog entries to be placed into catalog recovery for recovery in the event of a catalog failure.

*** VSE/VSAM user catalog.** An optional VSE/VSAM catalog used in the same way as the master catalog and pointed to by the master catalog. Use of user catalogs lessens the contention for the master catalog and facilitates volume portability.

VSE. Virtual Storage Extended. 1. Synonym for VSE/Advanced Functions. 2. An operating system that is an extension of Disk Operating System/Virtual Storage (DOS/VSE).

VSE/Advanced Functions. The basic operating support needed for a VSE-controlled installation. Synonymous with VSE.

VTOC. See *volume table of contents*.

W

work file. A file used to for temporary storage of data being processed.

Related IBM Manuals

The following lists the manuals referred to in this book.

IBM z/VSE

VSE/VSAM Commands, SC33-8245
z/VSE Planning, SC33-8221
z/VSE System Control Statements, SC33-8225
z/VSE Administration, SC33-8224
z/VSE Operation, SC33-8239
z/VSE Messages and Codes, SC33-8226, SC33-8227, SC33-8228
z/VSE Guide for Solving Problems, SC33-8232
z/VSE Guide to System Functions, SC33-8233
z/VSE System Control Statements, SC33-8225
z/VSE System Macros Reference, SC33-8230
z/VSE System Utilities, SC33-8234

Various

Device Support Facilities User's Guide and Reference, GC35-0033
IBM Dictionary of Computing, SC20-1699
American National Dictionary for Information Processing Systems
ISO Vocabulary - Information Processing
ISO Vocabulary - Office Machines

© Copyright IBM Corp. 1979, 2005

Index

Special characters

// DLBL job control statement (VSE/VSAM) 26
// DLBL statement, when required 23
// EXEC IDCAMS statement 18
// EXEC job control statement (VSE/VSAM) 18, 36
// EXEC PARM 39
// EXTENT job control statement (VSE/VSAM) 39
// UPSI statement (with maintain VTOC/VOL1 labels) 355
// UPSI statement (with SNAP dump) 353

Numerics

16MB line of storage, buffers above 16
3390-9 disk device 77
3995-151 Optical Library Server support 90
3995-151, catalog on 331

A

abbreviations, list of xvii
abnormal
  end with reusable file, default for 30
  end-of-job disposition (managed-SAM access) 35
  end-of-job disposition (VSE/VSAM access) 34
  termination, find end-of-file at 111
abnormal end of job 202
ACB (VSE/VSAM macro)
  access to SAM ESDS files 151
  buffer space allocation 96
  buffer space for index records 121
  buffer space size 95
  data set name sharing 192
  DDNAME (file name) parameter 27
  password, supplying 126
  relationship to FILE(dname) 27
  VTAM compatibility 331
ACB JRNAD exit (monitoring CA splits) 112
ACB macro 196
  See access method control block macro (ACB)
ACB macro format 197
access method control block (ACB) 184
access method control block macro (ACB)
  activation of requests 204
  active requests 232
  operand notations for macros 295
  parameter lists for macros 303
  positioning information 204
access method services
  See IDCAMS utility program
access modes valid for SAM ESDS files 156
access modes valid for VSE/VSAM ESDS files 156
access passwords in a catalog 126
ACF/VTAM similarities with VSE/VSAM 331
activating VSE/VSAM data compression 67
adding records to a file 111
addition, addressed sequential 281
addressed access 233, 236, 238, 244, 287
addressed-direct retrieval 272
addressed-sequential retrieval 270
allocate buffer space
  above the 16MB line 16
  ACB macro specification 96
  for file 27
  for path 98
  minimum 96
  performance considerations 99
  preventing deadlock in exclusive control 101
  supervisor buffers, specifying number of 22
allocate file implicitly
  primary allocation (SAM ESDS) 29
  RECORD parameter (SAM ESDS) 29
  RECSIZE parameter 29
  secondary allocation 29
  secondary allocation (SAM ESDS) 29
allocation limits of control area for CKD devices 86
ALTER (IDCAMS command)
  catalog entries (SAM ESDS) 170
  free space, changing values 111
  using with SAM ESDS files 170
alter file definitions in password-protected catalog 126
altering VSE/VSAM control blocks 226
alternate index
  access to 8
  advantage of 7
  example 8
  path to 8
  suballocating data space 84
  UPGRADE attribute and share options 130
  upgrade record, what it is 350
AMS
  See IDCAMS utility program
assign
  device to master catalog 21
  device to volume 22
audience of this book xv
authorization verification routine 128
authorize access to resources 125
automated system initialization (ASI)
  IPL commands, specifying 21

B

back up
  considerations (catalogs) 137
  considerations (files) 134, 135
  considerations (volumes) 134, 136
  dialog for 12
  generic names, using 56
  tools 139
BACKUP command
  migration to large DASD 78
  migration to SCSI disk 55
  overview 3
  use of 135
Backup/Restore Function
  loading into SVA 19
  protection of resources, use in 137
  storage requirements 19
  use of 3
  use with user-generated supervisor 19
backward reading of a VSE/VSAM file 241
base cluster, paths to 8
basic information on VSE/VSAM
  advantages of VSE/VSAM 1
  alternate index, advantage of 7
  authorize access to resources 125
  buffer allocation above 16MB line 16
  buffer space for CIs (with LSR) 99
  buffer space for CIs (with NSR) 94
  catalog check program (IKQVCHK) 345
  catalog, defining files in 41
  catalogs supported 6
  catalogs, advantages of 7
  catalogs, defining through job control 40
  catalogs, moving from device to device 53
  central control of files 1
  cluster 6
  commands of IDCAMS, overview of 8
  communicate with VSE/VSAM 8
  compatibility VSE/VSAM Version 2 and 7 13
  compatibility with other products 329
  control area (CA) 5
  control area size 85
  control interval (CI) 5
  control interval size 88
  data integrity 125
  data management 47
  data organization concepts 4
  data protection 1, 125
  data space 5
  data space classification 83
  defining data space 48
  defining files 48
  device independence 2

© Copyright IBM Corp. 1979, 2005

basic information on VSE/VSAM (continued)
diagnosis tools, overview 345
dialog, overview of 11
distributed free space 111
dynamic space allocation 2
environments, applicable 13
file organization 4
file types supported 4
files control 1
files, defining 48
files, move from device to device 53
files, transporting between systems 52
functions of VSE/VSAM 2
IDCAMS commands (overview) 8
IDCAMS utility program, use of 8
index options and performance 120
indexes supported 7
integrity of resources 132
interactive interface for users 11
introduction 1
ISAM files, support for 321
ISAM Interface Program 321
ISAM Interface Program, use of 2
ISAM to VSE/VSAM, convert 2
job catalog 7
job control requirements 23
job control to access VSE/VSAM files 11
labels used with VSE/VSAM 333
macros of VSE/VSAM 10
maintain VTOC/VOL1 labels (IKQVDU) 355
master catalog 7
modeling 57
operation, IPL commands 21
organization elements 5
overview 1
ownership of data space 47
password checking 126
password-protected objects, operating on 126
passwords 1
performance considerations 67, 83
planning for VSE/VSAM 13
portability of data 2
protection of resources 125
protection of resources, tools for 139
real mode operation 14
SAM ESDS file, purpose of 2
SAM ESDS files explained 151
SAM files, support for 151
SAM to VSE/VSAM, convert 2
scope of VSE/VSAM 1
SNAP dump program (IKQVEDA) 350
space allocation 108
space management 47
statistics on files 121
storage required for VSE/VSAM 13
use of VSE/VSAM 1
user catalogs 7
virtual disk support 51
virtual mode operation 14
VSE/VSAM Backup/Restore Function, use of 3
basic information on VSE/VSAM (continued)
VSE/VSAM Space Management for SAM Function, use of 2
BLDVRP macro
See resource pool build macro (BLDVRP)
BLDVRP macro format 208
block size related to CI size (for data component) 90
blocks per max CA (FBA devices) 87
blocks per min CA (FBA devices) 87
book, about this xv
boundaries of CAs and performance 86
buffer allocation, miscellaneous notes 99
buffer hashing 100
buffer pools (LSR), statistics on use 256
buffer pools, statistics on LSR 122
buffer space, I/O
additional (for CIs) 94
allocation (per ACB macro specification) 96
allocation parameters in ACB macro 96
control intervals (SAM ESDS) 167
default 88
direct processing considerations 95
for a file, specifying 27
for index records 120
minimum data buffers 96
minimum index buffers 97
performance considerations 95
preventing deadlock in exclusive control 101
sequential processing considerations 95
specifying 27
specifying for index records 121
specifying through ACB macro 95
specifying through BUFFERSPACE parameter 95
specifying through DLBL statement 96
specifying, methods of 95
supervisor buffers, specifying number of 22
buffer write macro (WRTBFR)
operand notation 303
buffer writing 263
buffering, extended 237, 287
buffering, normal 237
buffers for VSE/VSAM use (ACB macro) 198
buffers, specifying number of 27
BUFFERSPACE parameter 95, 121
effect on CA size with large DASD 79
BUFND parameter
in // DLBL 27, 96
in ACB macro 95
BUFNI parameter
in // DLBL 28, 96, 121
in ACB macro 95, 121
BUFSIZE operand (IPL) 22
BUFSIZE parameter
in IPL command 22
supervisor buffers, specifying 22
BUFSP parameter
in // DLBL 27, 96, 121
in ACB macro 95, 121
bytes per track (CKD devices) 86

C

CA
See control area (CA)
capacities of disk storage devices 86
CAT=filename specification 28
catalog
// DLBL required 24
See also catalog (job)
See also catalog (master)
See also catalog (notrecoverable)
See also catalog (recoverable)
See also catalog (user)
accessing passwords, hierarchy of 127
advantages of 7
assigning data space to performance class 84
authorization verification routine 128
back up considerations/methods 137
buffers for control blocks 38
check program (IKQVCHK), purpose of 345
content relating to files/volumes 135
cross-system sharing 131
damaged, rebuilding if 138
data output from catalog check 348
data space ownership 47
data space ownership, releasing 49
default, explained 44
defining files in 41
defining through job control 40
DFSMSdfp VSAM, compatibility of ICF catalogs 331
entries for cluster, purpose of 47
entries for files, purpose of 47
entries for files/volumes 135
entries for volumes, purpose of 47
entries for volumes, scope of 49
explicit specification 43
file name, specifying 28
file ownership, overriding 28
files in, performance considerations 83
high-key-range records 349
IDCAMS commands and job control 24
indicator of volume ownership 49
job catalog 7
job control requirements 23
listing entries 127
listing file definitions 126
low-key-range records 349
management, dialog for 11
master catalogs 7
migrating from device to device 53
migration 55
mounting requirements for volume 41
name, specifying through IDCAMS 24

catalog *(continued)*
name, specifying through job control 23
ownership of space on volume (restrictions) 41
password protection (user and master catalog) 126
performance on information requests 41
protection considerations 137
recover from catalog cannot be opened 146
recover from catalog volume unusable 147
recover from files cannot be opened 146
recover from inaccessible volume 148
recover from unusable catalog 146
recovering a 137
recovery considerations 137
relationship to files 40
reload function and REPRO 137
required 40
restore considerations 137
search order 44
search order and // DLBL specification 44
sharing across systems 131
space ownership 48
space ownership on volume (restrictions) 41
supported 6
transporting between VSE/VSAM systems 53
types supported 6
unload function and REPRO 137
unload to VSAM or nonVSAM file 137
user catalogs 7
volume ownership, removing 49
catalog (job)
// DLBL required 24
define through job control, how to 42
purpose 7
relationship to user catalog 7
use of 42
catalog (master)
// DLBL required 24
assigning device to 21
back up, creating a 137
define through job control, how to 41
file-ID 41
job control requirements 23
multiple 40
name of 41
not shared across systems 131
password protection 127
relationship to user catalogs 7
catalog (notrecoverable)
recover from loss of data 137
recover from unusable catalog 138
steps in protecting and recovering 137
what it is 137
catalog (user)
// DLBL required 24
advantages of 7
back up, creating a 137
define through job control, how to 42
deleting empty 127
multiple 41
password protection 127
relationship to master catalog 7
shared across systems 131
catalog check program (IKQVCHK)
actions on error discovery 346
examples 346
output of 348
purpose 345
running 346
when to use 345
catalog display macro (SHOWCAT)
operand notation 303
parameter list 313
catalog mismatch
actions and causes 148
data space group 147
entries do not match description of volumes 147
extents 147
file directory 147
files 147
guide to solving problems 147
high RBA 147
key range 148
minimization 148
recovery procedures 143
space map 147
statistics 147
volume entry 147
volume information 148
catalog protection
back up considerations/methods 137
backup copy, creating a 137
REPRO command, use of 137
VSE/Fast Copy utility, use of 137
catalog recovery area (CRA)
content of 135
listing content of 143
maintaining 355
not sharable 192
procedures for recovery 143
size (catalog migration) 54
catalog restore considerations 137
catalog space
GETVIS requirements 38
multiple volume ownership (MVS) 330
ownership 48
ownership bit 49
cataloging files 48
catalogs supported 6
catalogs, back up of
See catalog protection
CCDS 67
CCDS, how to define it 70
CDLOAD macro 315
chain of RPLs, positioning information for 234
chaining I/O requests 234
chaining request parameter lists 276
change catalog entries (SAM ESDS) 170
changes in this book xix
changing VSE/VSAM control blocks 226
checking of passwords by VSE/VSAM 126
CI
See control interval (CI)
CI access 236, 245, 246
CIDF (control information definition field) 113
CKD
See device
CLASS parameter 83, 84
class values for data spaces 84
close disposition 202
abnormal end-of-job (managed-SAM access) 35
abnormal end-of-job (VSE/VSAM access) 34
avoiding loss of data 36
deletion of file 34, 35
explained 34
file contents (managed-SAM access), loss of 35
file contents (VSE/VSAM access), loss of 34
file contents, protecting 36
managed-SAM access 35
of files 34
VSE/VSAM access 34
CLOSE macro
close routine 211
closing a file 211
compared to TCLOSE 211
description 211
format 211
return codes 212
use 187
close routine 211
closing a file 211
cluster
command (SAM ESDS) 158
data component 6
define (SAM ESDS) 158
defining 48
index component 6
paths to alternate indexes 8
suballocating data space 84
what it is 6
CLUSTER command (with SAM ESDS) 171
CMP-ACTIVE 69
CMP-REJECT 69
CMP-UNDET 69, 145
CMPPENDING 69
CMPSC 67
CNV access 245
codes
ACB return code 30
actions on errors 346
catalog management reason code 354
catalog management return code 354
CODE option (file access) 126
CODES attribute in modeling 59
condition code, changing a 10
condition code, testing a 10

codes *(continued)*
data security 126
error handling (with IIP) 327
job return code is not 0 (managed-SAM access) 35
job return code is not 0 (VSE/VSAM access) 34
maintain VTOC/VOL1 labels (IKQVDU) 359
on open failure 30
on time stamp mismatch 338
sequence error (with IIP) 324
tracing during processing (SNAP dump) 345
with OPEN error message (cross-system sharing) 131
commands
See also commands (IDCAMS)
See commands (other than IDCAMS)
commands (IDCAMS)
// DLBL required for catalogs 24
accessing passwords in a catalog 127
and // DLBL specifications 26
and // EXEC specifications 36
and // EXTENT specifications 39
BACKUP 3
BACKUP, use of 135
BUFFERSPACE and BUFSP relationship 27
BUFND relationship 27
BUFNI relationship 28
catalog, defining files in 41
considerations for IDCAMS operations 127
data integrity 132
DEFINE and file-ID relationship 27
DEFINE CLUSTER 48
DEFINE CLUSTER (use in data integrity) 133
DEFINE SPACE 48
DEFINE SPACE (use in data integrity) 132
DEFINE USERCATALOG (use in data integrity) 133
defining SAM ESDS file explicitly 158
deleting empty data space 127
deleting non-empty data space 127
deleting protected file entry from catalog 127
DTFIS (ISAM) related to DEFINE 325
DTFIS (ISAM), support for 325
expiration date relationship to // DLBL 27
explicit catalog specification 43
EXPORT command, use of 136
EXPORTRA command, use of 136
file access, considerations for 128
file name, specifying 27
functional commands 9
IMPORT command, use of 136
IMPORTRA command, use of 136
job control statements required for files 25
listing catalog entries 127
commands (IDCAMS) *(continued)*
migrating catalogs from device to device 55
migrating files from one device type to another 55
migrating files from one volume to another 56
modal commands 10
MODEL subparameter 58
name of catalog, specifying 24
operations and passwords 127
overview 8
password authorizations 127
password incorrect, prompt on 127
PRINT and CAT relationship 28
REPRO and CAT relationship 28
REPRO command, use of 136, 137
RESTORE 3
RESTORE, use of 135
SAM ESDS files, support for 153
SAM ESDS files, using with 170
time stamp entry, when updated 50
transporting catalogs between VSE/VSAM systems 53
transporting files between VSE/VSAM and DFSMSdfp VSAM 52
transporting files between VSE/VSAM and MVS/VSAM 52
transporting files between VSE/VSAM systems 53
types of commands 9
with SAM ESDS 153
commands (other than IDCAMS)
ASI, specifications for 21
IPL, specifications for 21
lock file, defining 21
master catalog, assigning device to 21
supervisor buffers, specifying number of 22
communicate with VSE/VSAM, how to 8
compatibility
ACF/VTAM with VSE/VSAM 331
DFSMSdfp VSAM ICF catalogs with VSE/VSAM 331
VSE/VSAM files to DFSMSdfp VSAM 329
VSE/VSAM Version 2 and 6 13
with other IBM products 329
compressed data, introduction 67
compressed files, working with 67
Compression Control Data Set 67
Compression Control Data Set, Defining the 70
compression control services trace 351
compression facility, ESA/390 67
compression management services control block trace 351
compression prediction
See VSE/VSAM Compression Prediction Tool (IKQPRED)
compression states 69
conceptual information
See basic information on VSE/VSAM
concurrent I/O requests 204
connect processing program to file
See OPEN macro
connecting a file for processing 228
control area (CA)
allocation limits for CKD devices 86
and index record size 86
content of 5
crossing boundaries 86
disk storage size (CKD devices) 86
disk storage size (FBA devices) 87
fixed-size blocks (FBA devices) 85
free space, specifying 111, 113
maximum size 85
maximum size, relationship to cylinder size 85
minimum size 85
minimum size, relationship to track size 85
preformatting before inserting records 110
size of 5
size, influencing 86
size, performance implications 86
specifying space 86
statistical information 121
statistical information not updated 122
unused free space 113
what it is 5
control area, splitting of
direct processing 114
examples of record insertion 114
monitoring, means for 112
overheads 112
placement of records 114
rules 114
sequential processing 114
control block display macro (SHOWCB)
operand notation 299
parameter list 308
control block generate macro (GENCB)
operand notation 297
parameter list 304
control block manipulation macros 184
control block modify macro (MODCB)
operand notation 298
parameter list 306
control block test macro (TESTCB)
operand notation 300
parameter list 309
control information definition field (CIDF) 113
control interval (CI)
and index record size 86
block size computation for data component 88
buffer space (SAM ESDS) 167
for data component, performance considerations 92
free space, specifying 111, 113
handling CIs 125
password parameter 125
physical block size computation for data component 88
relationship to block size 88
size calculations 93
size defaults 88

control interval (CI) *(continued)*
size in a data component 89
size in an index component 92
size of 5
size relation to block size (for data component) 90
size relation to track space (for data component) 90
size relationship to other specifications 88
size, displaying actual settings 92
size, effect on buffer size 94
size, performance considerations 92
specifying size 88
statistical information 121
statistical information not updated 122
unused free space 113
what it is 5
control interval, splitting of
direct processing 114
examples of record insertion 114
placement of records 114
rules 114
sequential processing 114
CONTROLPW (control interval password parameter) 125
converting ISAM files to VSE/VSAM files 2
converting SAM files to VSE/VSAM files 2
count key data (CKD) device
See device
CRA
See catalog recovery area (CRA)
cross-system sharing 131
current type data, recovery of 139
CYLINDER parameter 109
cylinder size (CKD devices) relationship to max CA size 85
cylinders per volume (CKD devices) 86

D

DASD
support of large DASD 77
DASD sharing facility (VSE/VSAM) 21, 129
data
component name 6
integrity, tools for 132, 139
organization concepts 4
portability 2
portability to MVS/VSAM 329
recovery procedures 143
recovery, levels of 139
data compression
activation 67
backup 150
backup and restore, EXPORT command 136
CMP-UNDET 145
CMP-UNDET, how to recover 145
compatibility considerations 331
compression control services trace 351
data compression *(continued)*
compression management services control block trace 351
compression states 69
data format of records 69
defining the CCDS 70
eligible data set types 71
introduction 67
portability to DFSMSdfp VSAM 329
restrictions 71
transporting files between systems 53
data loss, avoiding 132
data protection 125
data recovery 125
data secure file bit, setting of 48
data set name sharing (ACB macro) 198
data set name, sharing of 192
data space
class value, assigning 83
continuation, where described 334
controlling performance 83
define (example) 341
defining 48
defining for files with UNIQUE attribute 48, 339
delete, how to 49, 358
deletion, effect on VTOC 336
descriptor 333
empty, deleting 127
extents, where recorded 49, 333
format-1 label processing 337
format-3 label processing 338
format-4 label processing 338
group mismatch 147
label (format-1 label) 333
label deletion from VTOC 336
label processing overview 335
name in the VTOC 50, 333
name, assignment of 333
names, format of 50
non-empty, deleting 127
ownership 5
ownership, releasing from 49
performance, controlling 83
protection status, indication of 48
purpose of 5
records of ownership, where located 48
redefine, how to 355
relationship of volumes, files, labels 334
suballocating 83
suballocation 336, 339
unique (user-specified names) 50
VOL1 label processing 337
VTOC label processing 335
work files in 51
data space class
CLASS parameter 83
classification, values for 84
default value 84
example 84
for MVS/VSAM 84
high performance value 84
IMPORT command 85
restrictions 85
data space class *(continued)*
USECLASS parameter 84
user-defined values 84
where specified 83
data, backup of
See protection of resources
data, restore of
See restore
DATE disposition (ACB macro) 202
date, expiration 27
DBB 68
DDN (in ACB macro) 192
DDNAME (file name) parameter of ACB 27
deadlock in exclusive control 101
DEF (IPL command) 21
DEF SYSCAT (assign device to master catalog) 21
default catalog 44
// DLBL with application programs 23
explained 44
ownership, overriding 28
with catalog check program (IKQVCHK) 346
default volumes 65
deferred requests, relating 263
deferring write operations 262
DEFINE (IDCAMS command)
allocation of space 109
allocation of space for dynamic files 109
allocation of space for modeling 109
basics on dynamic files 109
block size specification 88
buffer space default 88
buffer space for index records 121
buffer space size 95
CI size specification 88
cluster 48
CLUSTER parameters (with SAM ESDS) 159
CLUSTER parameters with DTFIS (ISAM) 325
control interval size 88
data space class 83
data space on volume 48
date specification 27
dynamic file, specifying 109
files, defining 48
free space 111
identify volume to contain files 48
key compression 93
key range (with multiple volume) 102
modeling of objects 57
modeling of parameters 57, 63
NOWRITECHECK parameter 110
ORDERED parameter 103
ordered space allocation 103
preformatting space for CAs 110
record size 88
record size computation 88
records, write check of 110
RECOVERY parameter 110
RECOVERY/SPEED considerations 110

DEFINE (IDCAMS command) *(continued)*
space allocation (multiple volume), examples 103
space allocation options 108
space allocation parameters 109
SPEED parameter 110
UNORDERED parameter 103
unordered space allocation 103
volume to contain file, identify 48
WRITECHECK parameter 110
DEFINE CLUSTER command (use in data integrity) 133
define requests
See parameter list request macro (RPL)
DEFINE SPACE command (use in data integrity) 132
DEFINE USERCATALOG command (use in data integrity) 133
defining
access authority 125
clusters 48
data space 48
files 48
files with UNIQUE attribute 48
indexes 7
nonVSAM files in VSE/VSAM catalog, identifying 48
objects (command overview) 9
objects by way of modeling 57
defining file
See files
defining job catalog
See catalog (job)
defining master catalog
See catalog (master)
Defining the Compression Control Data Set 70
defining user catalog
See catalog (user)
DELETE (IDCAMS command)
data space 49
IGNOREERROR and checks on catalog 345
remove volume ownership from catalog 49
using with SAM ESDS files 171
DELETE disposition (ACB macro) 202
delete records (ERASE macro) 214
delete VSE/VSAM resource pool (DLVRP) macro 212
deleting
access authority 125
cluster 358
data space 49
data space, effect on VTOC 336
data space, how to 49, 358
empty data space 127
empty user catalog 127
files (SAM ESDS) 171
information from catalog 142
information from VTOC 142
labels from VTOC 336
non-empty data space 127
objects (command overview) 9
protected file entry from catalog 127
records 113
deleting *(continued)*
space using DELETE SPACE FORCE 142
unprotected files 128
VTOC label 358
deletion, addressed sequential 286
deletion, keyed direct 286
device
assignment when mounting volume 22
automatic assignment to volume 22
block size related to CI size 90
disk storage size (FBA devices) 87
fixed-size blocks (FBA devices) 85
limits of CA allocation 86
SCSI 80
storage capacities for CKD 86
support for 80
track space related to block size 90
track space related to CI size 90
track/cylinder size for CKD 85
device dependencies 14, 77
device migration
See migrating
Device Support Facilities (DSF)
initialize disk pack (format-4 label creation) 334
specify VTOC size 336
when used 334
DFSMSdfp VSAM
compatibility of ICF catalogs with VSE/VSAM 331
transporting files to VSE/VSAM 52
diagnosis tools
catalog check program (IKQVCHK) 345
LISTCRA (IDCAMS command) to list CRA 143
maintain VTOC/VOL1 labels (IKQVDU) 355
overview 345
SNAP dump program (IKQVEDA) 350
diagnostics (processing option PARM) 38
dialogs (z/VSE)
overview 11
dictionary 67
dictionary building block 68
direct processing 239
direct retrieval 242
directory mismatch 147
disconnect processing program from file
See CLOSE macro
disk extent information 39
disk storage devices, capacities of 86
diskette extent information 39
DISP specification 28
disposition of a file 202
disposition of files 29
disposition, close 202
distributed free space 111
DLBL statement (job control)
alternative specification for catalog name 24
and catalog search order 44
and defining files 48
DLBL statement (job control) *(continued)*
avoiding loss of data 36
buffer number, specifying 27
buffer space for index records 121
buffer space size 96
buffer space, specifying 27
CAT=filename specification 28
DISP (disposition) specification 28
disposition control (SAM ESDS) 155
explicit catalog specification 43
file disposition, specifying 28
file name for catalog owning a file 28
file name, specifying 27
file-ID specification 27
files with UNIQUE attribute, how to define 48
for file processing 340
for job catalog 42
for master catalog 23
for unique files 337
for user catalog 42
format of 27
information for implicit define (SAM ESDS) 163
job catalog, how to define 42
master catalog, how to define 41
options of the ACB macro 202
record allocation number (SAM ESDS), specifying 29
record allocation size (SAM ESDS), specifying 29
RECORDS specification 29
RECSIZE specification (SAM ESDS) 29
required when using IDCAMS commands 24
user catalog, how to define 42
virtual storage for buffer space 27
DLF (IPL command) 21
DLVRP (delete VSE/VSAM resource pool) macro 212
DLVRP macro
See resource pool delete macro (DLVRP)
DLVRP macro format 212
dname (file name) specification 27
downlevel type data, recovery of 139
DSN (in ACB macro) 192
DSN structure 201
DSN structure, GETVIS space requirement 15
DSN=dsname (scratch VTOC label for a specified file) 358
DTFPH support for SAM ESDS 151, 165
DTFSD information (implicit define SAM ESDS file) 162
DTFSD information (implicit define SAM ESDS work file) 163
DTFSD MOUNTED=SINGLE information (implicit define SAM ESDS work file) 163
DTFSD support for SAM ESDS 151
duplicate data condition, recover from 144
dynamic files 109

dynamic space
See also dynamic space allocation
See files (DYNAMIC)
dynamic space allocation
advantages 109, 152
basics 5
device independence 2
dynamic file, advantages 109
dynamic file, specifying 109
dynamic file, what it is 109
example (SAM ESDS) 177
EXCP support (SAM ESDS) 165
for files 109
how it works 5, 109, 152
job control (SAM ESDS) 152
making a file a dynamic file 109, 178
NOALLOCATION, use of 109
performance 109, 165
primary allocation 154
restrictions 110
REUSE with NOALLOCATION, purpose of 178
SAM ESDS files, specifying for 164
secondary allocation 154
specifying for work files 160
unallocated dynamic file 30
with SAM ESDS files 152
work files (SAM ESDS), specifying for 154

E

empty
CAs/CIs and FREESPACE parameter 113
data space (non-empty), deleting 127
data space, deleting 127
deallocate a file, meaning of 30
dynamic file and EXPORT/EXPORTRA support 110
file, open failure 34
object, backup 3, 135
object, restore 3
object, what it is 3, 56
objects, backup 55
objects, restore 55
reset a file, meaning of 30
user catalog, deleting 127
empty file (SAM ESDS), treatment of 166
end-of-file processing 185, 216
ending a processing request (ENDREQ macro) 213
ENDREQ macro
See processing end request macro (ENDREQ)
ENDREQ macro format 214
Enterprise Storage Server (ESS) xix, 9, 87
environments for VSE/VSAM 13
EODAD (end of data address) exit 185, 216
ERASE macro
description 214
format 214
operands 214
errors
See codes
ESA/390 compression facility 67
ESDS file (SAM)
access by ACB 151
access by VSE/VSAM programs 151
access considerations for managed-SAM files) 166
access mode and record format, specifying 159
access modes, valid 156
accessing data through DTFPH 161
allocating disk space as single extend 161
allocating file extension space 155
allocation size, requesting 153
alter catalog entries 170
ALTER command, using 170
assignments ignored 167
BLOCKS parameter explained 160
buffer space for CI 167
change programs to VSE/VSAM programs 156
CI and track/cylinder relationship 153
CLOSE, checking for 173
CLUSTER command, using 171
CLUSTER parameters explained 159
convert to VSE/VSAM files 157
convert unmanaged-SAM files 172
creating, steps in 157
CYLINDERS parameter explained 160
data not in CI format, accessing 159
define a default model (example) 177
DEFINE CLUSTER command, purpose of 151
DEFINE CLUSTER command, using 171
define dynamic (example) 177
define implicitly at OPEN 155
defined 151
defining explicitly 158
defining implicitly 161
defining, ways of 158
DELETE command, using 171
deleting explicitly 171
deleting implicitly 173
device independence 153
device-dependent SAM functions, restrictions for 167
disk-independence, recommendations for 167
DISP (disposition) parameter 155
disposition control for REUSE file 155
disposition default (example) 155
DTF not opened (file ignored) 167
DTF specifications, restrictions for 168
DTFPH access support 165
DTFPH method of access, when to use 159
DTFPH specifications, restrictions for 168
DTFPH support 151
ESDS file (SAM) *(continued)*
DTFSD information (implicit define work file) 163
DTFSD information (implicit define) 162
DTFSD MOUNTED=SINGLE information (implicit define) 163
DTFSD support 151
dynamic space allocation 152
dynamic space allocation (work files) 160
dynamic space allocation, performance 165
dynamic space allocation, specifying 164
empty file, treatment of 166
explicitly define 158
EXPORT command, using 171
extending existing files 155
format and access mode, specifying 159
format differences to VSE/VSAM ESDS file 179
formats, abbreviations used for 159
functions available 152
GETVIS space, specifying 167
ICCF, partition independence with 161
IDCAMS commands, using 153, 170
ignored (DTF not opened) 167
implicit define 153
implicit define (example) 176
implicit define cluster, occurrences of 161
implicit define cluster, VSE/VSAM assumptions 162
implicit define, information from DLBL statement 163
implicit define, information from DTFSD 162, 163
implicit define, information from DTFSD MOUNTED=SINGLE 163
implicit define, information from EXTEND statement 164
implicit delete 153
implicit deletion, cases of 173
IMPORT command, using 171
job control 152
level 1 advantages over SAM file 152, 156
level 2 advantages over SAM file 156
level 3 advantages over SAM file 157
LISTCAT command, using 171
loading (example) 174
managed-SAM access, define considerations 165
managed-SAM access, purpose of 165
managed-SAM, access considerations 166
management for work files (SAM ESDS) 154
migrating SAM files 155
modeling for DEFINE CLUSTER command 153
multiple extends, support for 154
multiple volumes, support for 154

ESDS file (SAM) *(continued)*
NOCIFORMAT specification, meaning of 159
NONINDEXED file, establishing as SAM ESDS file 159
overview 151
partition independence (work files) 154
partition independence, specifying 159, 161
physical record 181
planning for 154
portability 153
primary space allocation (dynamic) 154
PRINT command, using 172
processor independence (work files) 154
processor independence, specifying 159, 161
programs changes 156
protection 153
purpose of 2
RECORDFORMAT parameter explained 159
RECORDFORMAT parameter, purpose of 151
RECORDFORMAT to RECORDSIZE relationship 160
RECORDSIZE parameter explained 159
recovery 153
REPRO command, using 172
requirements for creating/using 151
resetting at OPEN 155
restrictions in using 167
retention period default (work files) 161
REUSE parameter 162
SAM file, defining for use in VSE/VSAM data space 152
SAM logical block 181
secondary space allocation (dynamic) 154
sharing characteristics 166
space assignment 157
space management (automatic) 154
space, where to specify 158
specifying 151, 157
specifying a file as ESDS file 159
system work file support 163, 168
TRACKS parameter explained 160
using IDCAMS commands 170
VERIFY command, using 173
VOLUMES parameter explained 160
volumes, specifying 160
VSE/VSAM access, considerations for 169
VSE/VSAM block 181
VSE/VSAM Space Management for SAM Function, use of 152
work files, considerations 154
work files, space management for 154
ESDS file (VSE/VSAM)
access modes, valid 156
ESDS file (VSE/VSAM) *(continued)*
format differences to SAM ESDS file 179
examples
allocation for multiple volumes (an exercise) 107
alternate index 8
CA splits 114
catalog check program (IKQVCHK) 346
CI splits 114
data space class 84
data spaces, define 341
defining catalogs through job control 42
explicit catalog specification 43
explicit models (allocation) 58
explicit models (noallocation) 60
file in not recoverable catalog, define 343
file, process 343
implicit models (allocation) 61
invoke IDCAMS, how to 38
job catalog, use of 42
job streams 340
macro operands, notation of 295
maintain VTOC/VOL1 labels 355
PARM parameter 38
SAM ESDS file, define default model 177
SAM ESDS file, define dynamic 177
SAM ESDS file, define implicitly 176
SAM ESDS file, loading 174
SNAP dump program (IKQVEDA) 353
space allocation on multiple volumes 103
unique file, define 343
volume layout (data spaces, files, VTOC) 335
exclusive control, preventing deadlock in 101
EXCPAD (EXCP address) exit 185, 217
EXEC
See EXEC statement (job control)
EXEC statement (job control)
catalog check program (IKQVCHK) 346
coding rules 38
format of 37
GETVIS for non-SVA-eligible phases 38
invoke IDCAMS, how to 38
job termination, how to avoid 38
non-SVA-eligible phases, GETVIS for 38
PARM parameter, advantages of using 39
PARM parameter, examples of 38
processing options 38
real mode, execute program in 37
size to load a program 38
SIZE=AUTO recommended 38
SNAP dump program (IKQVEDA) 351
VTOC/VOL1 label maintain program (IKQVDU) 355
execute form of GENCB, MODCB, SHOWCB, TESTCB 291
exit list (EXLST) macro 185
exit list macro (EXLST)
positioning if I/O error occurs 221
exit, security-verification 128
EXLST macro 214
See exit list macro (EXLST)
EXLST macro format 215
expiration date 27
explicit
allocation model 57
allocation model (example) 58
catalog specification 43
define cluster (SAM ESDS) 158
defining file (SAM ESDS) 158
modeling (SAM ESDS) 153
modeling, specifying the model 57
models 57
models (allocation) 58, 60
NOALLOCATION model 57
NOALLOCATION model (example) 60
opening of file 25
specifications when migrating objects to other device 57
EXPORT (IDCAMS command)
dynamic file support 110
use of 136
using with SAM ESDS files 171
export file (SAM ESDS) 171
EXPORTRA (IDCAMS command)
dynamic file support 110
use of 136
extended user buffering option 237
extended-addressed KSDS 233, 236, 238, 245, 246, 261, 287
EXTENT
See EXTENT statement (job control)
extent information, when to specify 39
extent overlap, action on 337
EXTENT statement (job control)
and defining files 48
define files with UNIQUE attribute 48
for unique files 337
for unique files, validation 337
format of 39
information for implicit define SAM ESDS file 164
logical unit of volume 39
number of blocks/tracks for file 39
relative block/track of extent 39
serial number of volume 39
extents of data space, where recorded 49
EXTRACT codes 317

F

Fast Copy utility (VSE)
backup copy, creating a 136, 137
backup copy, restoring a 136
use in protecting resources 132
fast recovery of files, preparing for 150
FBA
See device
Fibre Channel Protocol (FCP) 80

file closing, temporarily 257
file disposition
  basic information 29
  considerations (DLBL, ACB, DTF) 36
  DISP parameter, default of 30
  on closing files 34
  on opening files 30
  specification for OPEN/CLOSE processing 28
  states at OPEN 30
  status at CLOSE 34
  where to specify 29
file processing, close 211
file protection
  back up considerations 135
  BACKUP command, use of 135
  data secure file bit, setting of 48
  EXPORT command, use of 136
  EXPORTRA command, use of 136
  IMPORT command, use of 136
  IMPORTRA command, use of 136
  REPRO command, use of 136
  RESTORE command, use of 135
  user-written back up programs 136
file space
  dynamic allocation 109
  NOALLOCATION parameter 109
file types supported 4
file, connecting for processing 228
files
  // DLBL statement 23
  adding records, considerations for 111
  altering definitions in password-protected catalog 126
  and lock facility 130
  authorization verification routine 128
  back up and recovery considerations 134, 135
  back up methods 135
  block, relative 39
  blocks, allocating number of 39
  buffer space, specifying 27
  buffers, specifying number of 27
  cataloging 48
  CLOSE disposition 34
  cross-system sharing 131
  data space records, where located 48
  data space suballocation 336, 339
  define in not recoverable catalog (example) 343
  define unique (example) 343
  defining 48
  defining and DLBL/EXTENT statements 48
  defining in catalog 41
  defining with UNIQUE attribute 48
  defining, what it means 7
  delete, how to 358
  deleting protected file entry from catalog 127
  directory mismatch 147
  disposition, basic information on 29
  disposition, specifying 28
  duplicate file for back up, creating 136
  entries in catalog 135
files *(continued)*
  entries in catalog, purpose of 47
  extents mismatch 147
  file-ID, specifying 27
  formats 4
  high RBA mismatch 147
  IDCAMS and job control statements 25
  identifier, specifying 27
  identify volume to contain files 48
  identifying nonVSAM files in VSE/VSAM catalog 48
  in catalog, performance consideration 83
  job control statements required 25
  listing definitions in a catalog 126
  loading 339
  loading a file, considerations for 111
  loading records, considerations for 111
  locking activities, trace of 351
  management 7
  management, dialog for 11
  migrating from one device type to another 55
  migrating from one volume to another 56
  mismatches 147
  name associated with volume 27
  name, specifying 27
  non-unique, defining 54, 336, 339
  nonVSAM, unloading a catalog to 137
  number of blocks, allocating 39
  number of tracks, allocating 39
  OPEN disposition 30
  opening for processing (SHAREOPTIONS) 130
  organization 4
  ownership, overriding 28
  portability to MVS/VSAM 329
  porting, create copy for 136
  processing (example) 343
  processing (SHAREOPTIONS) 130
  processing, DLBL statement for 340
  protection considerations 128, 134
  quick recovery, considerations for 150
  read integrity, ensuring 130
  record allocation size (SAM ESDS), specifying 29
  record, allocation size (SAM ESDS) 29
  records adding, considerations for 111
  records, number of 29
  recover from duplicate data condition 144
  recover from file cannot be opened 145
  recover from file completely unreadable 145
  recover from file not properly closed 143
  recover from file partially unreadable 145
files *(continued)*
  recover from files cannot be opened 146
  recover from inaccessibility 145
  recover from incomplete write to disk 144
  recover from incorrect high RBA 144
  recovery and backup considerations 134
  relationship of volumes, data space, labels 334
  relationship to catalogs 40
  reorganization considerations 113
  restore 135
  reusable, specifying 28
  SAM ESDS files, planning for 154
  SAM file for back up, creating 136
  search order in catalogs 44
  secondary allocation, minimize 133
  sharing 129
  sharing across systems 131
  sharing and protection 129
  sharing control blocks 192
  sharing, options for 130
  states at OPEN disposition 30
  statistic mismatch 147
  statistical information 121
  statistical information not updated 122
  trace of activities 351
  track, relative 39
  tracks, allocating number of 39
  transporting between systems 52
  types supported 4
  unique, explanation for 336
  unprotected, deletion of 128
  user-written back up programs 136
  volume mounting, requirements for 339
  write integrity, ensuring 130
files (DYNAMIC)
  advantages 109
  allocation of space 109
  define example (SAM ESDS) 177
  deleting 110
  restrictions 110
  space allocation (SAM ESDS) 152
  space allocation (SAM ESDS), performance 165
  space allocation (SAM ESDS), specifying 164
  space allocation, suppressing 109
  specifying 109
  what it is 109
files (UNIQUE)
  define (example) 343
  define data space 48, 339
  delete, how to 358
  DLBL statement needed 337
  explained 336
  EXTENT statement needed 337
  format-1 label processing 337
  format-3 label processing 338
  format-4 label processing 338
  key area contents (format-1 label) 333
  label processing 340

files (UNIQUE) *(continued)*
record of physical extents 135
redefine, how to 355
UNIQUE attribute and job control statements 48
user-specified names 50
volume time stamp 338
files (WORK)
considerations 154
considerations for access 167
DTFSD information (implicit define) 163
GETVIS space 37
IJSYSnn support 168
on virtual disk 51
on virtual disk, preparation for 51
partition/processor independence 154
planning for (SAM ESDS) 154
recommendations (SAM ESDS) 160
retention period default 161
single extent allocation, requesting 163
system work file support (SAM ESDS) 168
files, back up of
See file protection
files, restore of
if catalog damaged 138
RESTORE command, use of 135
FORCE option 142
format differences (VSE/VSAM ESDS and SAM ESDS) 179
format of compressed data 69
format-1 VTOC label 48
format-1 VTOC label, purpose of 333
format-3 VTOC label, purpose of 334
format-4 VTOC label 49
format-4 VTOC label settings (when space released) 49
format-4 VTOC label, purpose of 334
free space
causes of 111
changing values 111
CI size relative to record size (performance) 92
default 111
file loading considerations 111
for mass insertion 111
in CA, specifying 113
in CI, specifying 113
parameter 111
reclaiming 113
specifying 111
threshold 112
too much/too little 112
unused if nonspanned data CI 92
unused, cause of 113
FREESPACE parameter 111
function codes for alternate index processing 290
functions of VSE/VSAM 2

G

GENCB (generate control block) macro 187, 221
GENCB macro
See control block generate macro (GENCB)
GENCB macro format 222
generate form of GENCB, MODCB, SHOWCB, TESTCB 292
generated names for data spaces 50
generic back up 56
generic restore 56
GET macro 224
See records retrieve macro (GET)
GET macro format 224
GETVIS space
default adjustment 18
failure correction 360
for buffers 14, 38
for control blocks 14, 38
for IDCAMS 18
for IDCAMS, where to specify 18
for non-SVA-eligible phases 38
for non-SVA-eligible routines 13
for users of the VSE/VSAM Space Management for SAM Function 37, 167
minimum for files 14
non-GETVIS space for job control routines 18
specifying for users of the VSE/VSAM Space Management for SAM Function 18
usage, keeping small 16

H

hardware compression facility, ESA/390 67
hierarchy of catalog search 44
high RBA mismatch 147

I

I/O areas for a VSE/VSAM file 198, 246
I/O buffer
See buffer space, I/O
I/O buffers, managing 262
I/O operations, overlapping of 217
I/O routines, user 317
ICF (Integrated Catalog Facility 331
ICF catalogs, compatibility with VSE/VSAM catalogs 331
IDCAMS commands
See commands (IDCAMS)
IDCAMS utility program
GETVIS, where to specify 18
how to invoke (job control) 38
job termination, how to avoid 38
password authorizations 127
password prompt 127
storage requirements 18
use in protecting resources 132
use of 8
IGNOREERROR parameter and catalog check 345
IJSYSCT (name of master catalog) 41
IKQPRED
See also VSE/VSAM Compression Prediction Tool (IKQPRED)
overview 72
IKQVCHK (diagnosis tool) 345
IKQVDU (diagnosis tool) 345
IKQVEDA (diagnosis tool) 345
implicit
define file (SAM ESDS) 153
delete file (SAM ESDS) 153
file definition (SAM ESDS), allocation size 29
information from DLBL statement (SAM ESDS) 163
model, specifying name 61
modeling, choosing the model 57
models 57
NOALLOCATION model 57
NOALLOCATION model (example) 61
opening of file 25
IMPORT (IDCAMS command)
data space class 85
key compression 93
using with SAM ESDS files 171
import file (SAM ESDS) 171
IMPORTRA (IDCAMS command)
using with SAM ESDS files 153
in-core wrap trace 351
inaccessible file, recover from 145
incomplete write to disk, recover from 144
index
See also alternate index
component 6
component, CI size of 92
key compression 93
keys, reducing size of 93
options 7
separate from data residence 121
with VSE/VSAM 7
index options
AIX portions on different volumes 121
buffer space required 120
file portions on different volumes 121
index and data on separate volumes 121
key ranges 121
with KSDS and ESDS files 7
index records
in storage, methods of specifying 121
size to accommodate CAs and CIs 86
initial program load (IPL)
automatic device assignment to volume 22
commands for VSE/VSAM 21
lock file, defining 21
master catalog, assigning device to 21
supervisor buffers, specifying number of 22
volumes, ways of mounting 22
inserting records in a file 111
insertion, keyed direct 281
insertion, keyed-sequential 278

insertion, skip-sequential 279
Integrated Catalog Facility (ICF) 331
integrity of data
See protection of resources
integrity of data, tools for 139
interactive user interface
See dialogs (z/VSE)
Interrogation in data compression 68
introduction to VSE/VSAM 1
invoke IDCAMS (job control), how to 38
invoking IDCAMS from a program 295
invoking macro instructions 315
IPL
See initial program load (IPL)
ISAM (indexed-sequential access method)
compared to VSE/VSAM 321
converting to VSE/VSAM 2
files, convert to VSE/VSAM 2
performance improvements, possible 321
ISAM Interface Program (IIP)
defining a VSE/VSAM file 324
DTFIS related to DEFINE 325
ERREXT format 327
error handling 327
filenameC format 327
ISAM compared to VSE/VSAM 321
job control statements, changing 326
loading ISAM file into VSE/VSAM file 325
processing explained 326
storage requirements 18
using, prepare for 323
using, prerequisites for 324
VSE/VSAM functions available 322

J

job
cancellation (VOL1 label incorrect) 337
end-of-job disposition (managed-SAM access) 35
end-of-job disposition (VSE/VSAM access) 34
running a job 36
running a job step 36
step, start of 36
termination, how to avoid 38
job catalog
See catalog (job)
job catalog // DLBL statement 23
job control
access VSE/VSAM files 11
and IDCAMS commands 24
catalogs, defining 40
for files 48
for job catalog 42
for master catalog 41
for user catalog 42
GETVIS space for managed-SAM access 18
invoke IDCAMS 38
job control and IDCAMS commands 24
linkage editor, processing of control statements 37
job control *(continued)*
mounting volumes 22
non-GETVIS space for managed-SAM access 18
parameters 11
requirements 23
SAM in VSE/VSAM data space 152
simplified job control in using SAM ESDS 152
job control statements
// DLBL (VSE/VSAM) 26
// DLBL, where to specify 23
// EXEC (VSE/VSAM) 18, 36
// EXTENT (VSE/VSAM) 39
access VSE/VSAM files 11
alternative specification for // DLBL 24
and IDCAMS commands 24
avoiding job termination with IDCAMS 38
avoiding loss of data 36
defining catalogs (// DLBL) 41
defining files 48
explicit catalog specification 43
extent information, specifying 39
for application programs 23
for catalogs 23, 40
for ISAM programs 23
for SAM programs 23
implicit define SAM ESDS file 163
ISAM processing under VSE/VSAM 326
job catalog (// DLBL), specifying 42
master catalog (// DLBL), specifying 41
parameters, purpose of 11
program execution, starting 36
required for files 25, 48
required when using IDCAMS commands 24
requirements 23
running catalog check program (IKQVCHK) 346
running SNAP dump (IKQVEDA) 351
running VTOC/VOL1 labels maintain program (IKQVDU) 355
setting up execution of maintain VTOC/VOL1 labels 355
setting up execution of SNAP dump 353
user catalog (// DLBL), specifying 42
job step, start of 36
job stream
data spaces, define 341
examples 340
file in not recoverable catalog, define 343
file, process 343
unique file, define 343
JRNAD (journal) exit routine 185, 218

K

KEEP disposition (ACB macro) 202
key compression 93
key range mismatch 148
key ranges 121
keyed access 238
keyed deletion 244
keyed insertion 243
keyed positioning with POINT 273
keyed-direct retrieval 271
keyed-sequential retrieval 268
KEYRANGES parameter (with multiple volumes) 102
KSDS
performance considerations with large DASD 78

L

label
44-byte key area, what it is 333
check, when it occurs 336
creation, when it occurs 336
data space continuation, where described 334
data space extents, where described 333
data space name, assignment of 333
data space, description of 333
definition label for VTOC 334
deletion from VTOC, when it occurs 336
entries in the VTOC 48
explained 333
format-1 and format-3 labels, relationship of 333
format-1 label processing (unique files) 337
format-1 label, purpose of 333
format-1 VTOC 48
format-3 label , purpose of 334
format-3 label processing (unique files) 338
format-4 label processing (unique files) 338
format-4 label, purpose of 334
format-4 VTOC 49
format-4 VTOC settings (when space released) 49
information, submitting 335
key area contents (format-1 label) 333
location of 334
maintain program (IKQVDU), purpose of 355
not supported by VSE/VSAM 333
processing overview 335
processing, when it occurs 336
relationship of volumes, files, data space 334
sets, overriding 335
submitting 335
types 333
VOL1 label processing 337
VOL1 label, purpose of 333
VOL1 label, required location of 337
volume identifier 333
VTOC labels for FBA devices, residence of 336
VTOC, definition label for 334

large DASD
BUFFERSPACE parameter 79
KSDS 78
LISTCAT output with 79
migration using BACKUP 78
migration using RESTORE 78
performance considerations 78
restrictions 79
support for 77
LERAD exit routine 185, 220
levels of data recovery 139
linkage editor statements, processing of 37
list form of GENCB, MODCB, SHOWCB, TESTCB 291
LISTCAT (IDCAMS command)
displaying CI size 92
file statistics 121
monitoring CA splits 112
purpose 112, 121
relating names in VTOC (generated and user-specified) 50
using with SAM ESDS files 171
with large DASD 79
listing a file's definitions in a password-protected catalog 126
listing information (SAM ESDS) 171
load a SAM ESDS file (example) 174
loading a file 111, 339
loading the VSE/VSAM Backup/Restore Function into SVA 19
local shared resources (LSR)
allocate virtual storage for pools 194
buffer allocation 17
buffer pools statistics, requesting 256
buffer pools, statistics on use 256
connect pool to ACB 203
considerations 95, 99
control blocks of a file, sharing 192
data set name sharing 192
error codes, potential 210
file opening, specification before 210
I/O buffer sharing 192
LSR in ACB specified, effect on TCLOSE 257
LSR operand in ACB 202
MACRF operand, specifying 210
macros, overview 11
managing 99
multiple LSR pools 193
partition virtual storage 14
pool, connect to ACB 203
pool, residence of I/O buffers 209
pools above 16MB 193
pools, allocate virtual storage for 194
processing errors, avoiding 241, 242, 243
processing option, advantages of 192
program check 194
requirements with 31-bit addressing 194
restrictions with LSR 210
RMODE31 parameter in macros 194
separate data and index pools 210, 213
SHRPOOL parameter in ACB 203
space calculation 16
local shared resources (LSR) *(continued)*
statistics on buffer pools 122
TYPE operand of BLDVRP macro 210
TYPE operand of DLVRP macro 213
what to specify 207
write requests, deferring 262
locate mode 237, 246
lock facility (z/VSE), use of 130
lock file
defining 21
requirements, determining 21
locking activities, trace of 351
logical unit of volume containing extent 39
loss of data, avoiding 132
LSR
See local shared resources (LSR)
LSR operand in ACB 202

M

MACRF operand, specifying for LSR 210
macro groups 183
macros (VSE/VSAM)
coding, ways of 196
descriptions 195
display file statistics 121
operand notations explained 295
overview 10
parameter lists explained 303
purpose 10
syntax 195
test file parameters 121
unique control block, generating 331
VTAM control block, generating 331
maintain VTOC/VOL1 labels program (IKQVDU)
actions on error discovery 359
error message explained 359
execution, setting up for 355
output of 359
purpose and use 355
return codes explained 359
running 355
UPSI job control statement, setting of 355
managed-SAM access
See SAM access (managed)
managed-SAM file
See SAM file (managed)
managing I/O buffers 262
manipulation macros
overview 184
parameter list, internal 303
specifying 193
mass insertion of records 111
master catalog
See catalog (master)
master catalog // DLBL statement 41
MASTERPW (master password) 125
max CA per volume (CKD devices) 86
max CA per volume (FBA devices) 87
max CA, what it is 85
message area (OPEN/CLOSE/TCLOSE) 204
message format 206
messages
catalog check program (IKQVCHK) 346
maintain VTOC/VOL1 labels (IKQVDU) 359
migrating
catalogs from one device type to another 53
files from one device type to another 53
files from one volume to another 56
ISAM files to VSE/VSAM control 321
SAM files to VSE/VSAM control 155
to SCSI device 55
transporting files between systems 52
min CA per cylinder (CKD devices) 86
min CA per max CA (FBA devices) 87
min CA, what it is 85
mismatch problem
catalog entries do not match description of volumes 147
causes 148
data space group 147
extents mismatch 147
file directory mismatch 147
file statistics mismatch 147
files mismatch 147
guide to solving problems 147
high RBA mismatch 147
key range 148
minimizing catalog mismatches 148
recovery procedures 143
space map 147
volume entry 147
volume information mismatch 148
MODCB (modify control block) macro 188, 226
MODCB macro
See control block modify macro (MODCB)
MODCB macro format 226
modeling
access authority 125
advantages of 57
allocation of space 109
default models, using 61
default volumes 65
explicitly (allocation) 58
explicitly (noallocation) 60
for SAM ESDS 153
implicitly (allocation) 61
job termination, avoiding 58
MODEL subparameter 58
overriding system defaults 57
processing by VSE/VSAM 62
restrictions for 63
SAM ESDS file (example) 177
types 57
modifying VSE/VSAM control blocks 226
mounting need for volumes 51
move mode 237, 246
multiple extents (SAM ESDS) 154
multiple LSR pools 193

multiple volumes
DEFINE parameters for 102
ORDERED specification 103
performance notes 102
space allocation 102
space allocation (an exercise) 107
space allocation (with key range) 103
space allocation (without key range) 102
space allocation, examples of 103
support 102
UNORDERED specification 103
with SAM ESDS files 154
multivolume files 135
MVS, transfer files from VSE to 7
MVS/VSAM
data space class 84
moving VSE/VSAM files to 2
transporting files to VSE/VSAM 52

N

names for data spaces 50
NOALLOCATION parameter 109
non-empty data pace, deleting 127
non-shared resources (NSR)
buffer allocation 17
buffer space for CIs 94
considerations 99
I/O buffer sharing 192
I/O operations 192
master catalog 131
partition virtual storage 14
read/write integrity 192
space calculation 15
non-unique files, defining 54, 336, 339
nonVSAM file, unloading a catalog to 137
notrecoverable catalog
See catalog (notrecoverable)
NSR
See non-shared resources (NSR)
number of blocks for file, allocate 39
number of tracks for file, allocate 39

O

object (tasks, commands)
alter (command overview) 9
backup (command overview) 9
BACKUP command, use of 135
backup empty object 3, 135
backup multiple objects 56
backup operations on 3
backup to tape or disk 56
build alternate index (command overview) 9
cancel job (command overview) 9
cancel job step (command overview) 9
copying to another volume 56
data space, suballocating 83
define (command overview) 9
delete (command overview) 9
empty object, what it is 3
empty, what it is 56
object (tasks, commands) *(continued)*
first one on a volume 47
generated names 50
generic names, use of 56
interactive interface 11
levels of protection 125
list command 9
migrating catalogs 55
migrating files, methods for 56
migrating, methods for 53
MODEL subparameter 58
modeling of 57
modeling, advantages of 57
modeling, default volume lists 65
modeling, restrictions with 63
modeling, types of 57
move data (commands overview) 9
password-protected objects, operating on 126
passwords, protection by 1
performance, controlling 83
print (commands overview) 9
print command 9
restore (command overview) 9
RESTORE command, use of 135
restore empty object 3
restore multiple objects 56
restore operations on 3
restrictions for master catalog 52
suballocate data space for 47
verify (command overview) 9
open disposition
explained 30
of files 30
states with 30
OPEN macro
connecting a file for processing 228
description 228
format 228
open routine 228
operands 228
return codes 229
use 187
open routine 228
opening a file for processing 228
operand notation for macros
BLDVRP 302
DLVRP 302
explained 295
GENCB 297
MODCB 298
SHOWCAT 303
SHOWCB 299
TESTCB 300
WRTBFR 303
operation
authorize password submission 126
back up VSE/VSAM objects (dialog) 12
extent overlap, action on 337
file access code 126
job cancellation (VOL1 label incorrect) 337
password, supplying 126
real mode and virtual mode 37
restore VSE/VSAM objects (dialog) 12
operation *(continued)*
virtual mode and real mode 37
operation and job control 21
optimizing the performance of VSE/VSAM 83
options for processing, summary of 238
order of catalog search 44
order of space allocation 103
ORDERED parameter 103
organization elements with VSE/VSAM 5
overlap of volume extents, action on 337
overlapping I/O operations 217
overriding file catalog ownership 28
overview on VSE/VSAM 1
ownership
data space 5, 47
data space, releasing from catalog 49
indicator 49
of space 48
of volume 48
of volume and nonVSAM files 48
of volume, removing from catalog 49

P

parameter list (manipulation macros) 303
parameter list request macro (RPL)
chain of RPLs, positioning information 234
OPTCD values in RPL 225
OPTCD= specification and POINT macro 229
positioning for processing 239
processing of records, positioning for 239
records, processing of 239
sequential positioning 233
parameter lists for macros
BLDVRP 312
explained 303
GENCB 304
MODCB 306
SHOWCAT 313
SHOWCB 308
TESTCB 309
PARM parameter (EXEC statement), advantages of using 39
PARM parameter (EXEC statement), examples 38
partition independence (SAM ESDS work files) 154
partition space
adjusting minimum value (users of the VSE/VSAM Space Management for SAM Function) 37
for buffers 14
for control blocks 14
for IDCAMS 18
for non-SVA-eligible phases 38
for non-SVA-eligible routines 13, 14
for program execution 38
for real mode operation 14, 37
for SAM access routines 17
requirements 37

password
access to files, considerations for 128
access to passwords in a catalog 126
altering file definitions in a catalog 126
ATTEMPTS option (password control) 126
authorization routine (user-written) 128
authorize file access 126
authorize for submission by operator 126
catalog and files, relationship between 126
checking 126
CODE option (file access) 126
control interval accessing 125
deleting data space (empty/non-empty) 127
deleting protected file entry 127
deleting unprotected file 128
IDCAMS operations, considerations for 127
levels of access to resources 125
levels of access, relationship 125
listing catalog entries 127
listing definitions of a file 126
master catalog and user catalog, relationship of 127
MASTERPW (master password) 125
operator password, controlling submission 126
prompt from IDCAMS 127
protect resources 125
protected objects, operating on 126
read access 125
resources protection 125
submission through operator 126
submission through processing program 126
update access 125
UPDATEPW (update password) 125
user catalog and master catalog, relationship of 127
user catalog, deleting empty 127
user security-verification routine and MASTERPW 125
performance considerations
assigning data space to performance class 84
buffer space considerations 99
buffer space for CIs, optimizing 94
control area (CA) size 85, 86
control interval (CI) size 88, 92
data integrity 110
data protection 110
data space class values, use of 84
data space classification 83
degradation with real mode operation 14
dynamic files, advantages of 109
dynamic files, restrictions 110
file information, obtaining 121
files in catalog, number of 83
free space considerations 111
free space too much/too little 112
performance considerations *(continued)*
index and data on separate volumes 121
index options 120
index records, buffer space for 120
information requests to catalogs 41
ISAM, improvements possible for 321
key compression 93
measurements, means for 121
multiple volume support 102
optimizing VSE/VSAM 83
parts of a file on different volumes 121
parts of an AIX on different volumes 121
preformatting space for CAs 110
real mode and virtual mode 37
RECOVERY/SPEED considerations 110
space allocation 108
space allocation for SAM ESDS files 165
space utilization and CI size 90
statistics on files 121
suballocating data space 83
write check 110
physical block size for data component 88
physical record size for data component 88
planning
applicable environments 13
avoid performance degradation 14
buffer allocation above 16MB line 16
compatibility VSE/VSAM Version 2 and 7 13
considerations 13
for IDCAMS 18
for ISAM Interface Program (IIP) 18
for SAM ESDS files 154
for VSE/VSAM 13
for VSE/VSAM Backup/Restore Function 19
for VSE/VSAM Space Management for SAM Function 17
GETVIS for IDCAMS 18
local shared resource, space calculations 16
non-shared resource, space calculations 15
partition space for non-SVA-eligible routines 14
performance considerations 67, 83
quick recovery, considerations for 150
resources protection, considerations for 125
space for real mode operation 14
SVA required for VSE/VSAM 13
VSE/VSAM Backup/Restore Function with user-generated supervisor 19
POINT macro
See position for processing macro (POINT)
POINT macro format 230
portability of data 2
portability of VSE/VSAM files to DFSMSdfp VSAM 329
porting files
See migrating
position for processing macro (POINT)
positioning VSE/VSAM at a wanted record 229
positioning VSE/VSAM for processing 229
positioning VSE/VSAM for processing requests
activation of requests 204
active requests 232
at a wanted record 229
backward processing 229
cancellation of the position 213
chain of RPLs, positioning information 234
ending a request 213
for sequential or skip sequential processing 229
forward processing 229
if I/O error with data CI occurs 221
keeping for sequential or skip sequential processing 225
loss of positioning 221
OPTCD values in RPL for macros 225
positioning if OPTCD=(KEY,DIR,NSP) in RPL 230
positioning information maintained 204
positions for request macros in process 225
records into RBA if positioning not established 233
sequential positioning 233
primary allocation of file (SAM ESDS), records for 29
primary index
how generated 7
PRINT (IDCAMS command)
overview 9
using with SAM ESDS files 172
printed output (processing option) 38
procedures
catalog cannot be opened, recover from 146
catalog unusable, recover from 138, 146
catalog volume unusable, recover from 147
data management 47
defining data space 48
defining files 48
deleting data space 49
duplicate data condition, recover from 144
file cannot be opened, recover from 145
file completely unreadable, recover from 145
file inaccessible, recover from 145
file not properly closed, recover from 143
file partially unreadable, recover from 145

procedures *(continued)*
files cannot be opened, recover from 146
incorrect high RBA, recover from 144
ISAM to VSE/VSAM, converting from 323
migrating catalogs 55
migrating catalogs from device to device 53
migrating files from device to device 55
migrating SAM files 155
modeling objects 57
quick recovery of files 150
recognizing names in the VTOC 50
recovery of resources 143
relating names in VTOC (generated and user-specified) 50
running catalog check program (IKQVCHK) 346
running SNAP dump program (IKQVEDA) 351
running VTOC/VOL1 labels maintain program (IKQVDU) 355
space management 47
space ownership, releasing from catalog 49
transporting catalogs between VSE/VSAM systems 53
transporting files between VSE/VSAM and DFSMSdfp VSAM 52
transporting files between VSE/VSAM and MVS/VSAM 52
transporting files between VSE/VSAM systems 53
volume inaccessible, recover from 148
volume ownership, removing from catalog 49
work files on virtual disk 51
write to disk incomplete, recover from 144
processing end request macro (ENDREQ)
cancellation of the position for processing request 213
ending a processing a request 213
giving up the position for an RPL 213
processing of file, close 211
processing options (PARM parameter/command) 38
processing options, summary of 185, 238
processing shared data 129
processor independence (SAM ESDS work files) 154
program connection (OPEN macro)
See OPEN macro
program disconnection (CLOSE macro)
See CLOSE macro
program execution, start of 36
program load, storage for 38
program separation (TCLOSE)
See TCLOSE macro
prompting code 126
protection of resources
avoiding loss of data at CLOSE disposition 36
back up considerations (catalogs) 137
back up considerations (files) 134
back up considerations (volumes) 134
catalog check program, when to run 345
catalog content relating to files/volumes 135
catalogs 137
control interval (CI) access 125
cross-system data, specifying share options 131
data integrity tools 132
data integrity, commands for 132
data space classification 83
DEFINE CLUSTER command 133
DEFINE SPACE command 132
DEFINE USERCATALOG command 133
explained 125
Fast Copy utility 132
file access, considerations for 128
file sharing, degrees of 130
file unprotected, deletion of 128
files 134
IDCAMS operations, considerations for 127
levels of access to resources 125
levels of access, relationship 125
lock facility, use of 130
master access (MASTERPW parameter) 125
master catalog and user catalog, relationship of 127
operating on password-protected objects 126
password check 126
password relationship between catalog and files 126
password verification routine (user-written) 128
passwords, use of 125
quick recovery, preparing for 150
read access 125
recovery considerations (catalogs) 137
recovery considerations (files) 134
recovery considerations (volumes) 134
recovery specification, purpose of 110
RECOVERY/SPEED considerations 110
secondary allocation, minimize 133
shared files 129
SHAREOPTIONS parameter 129
tools 139
update access 125
user catalog and master catalog, relationship of 127
user security-verification routine and MASTERPW 125
volume separation 132
volumes 134
protection of resources *(continued)*
VTOC utility (IKQVDU) 132
with SAM ESDS files 153
write check 110
write check default 110
PUT macro
description 230
format 231
operands 231
positioning VSE/VSAM for processing 230

Q

quick recovery of files, preparing for 150

R

RBA (relative byte address)
direct processing of records (sequential) 239
processing of records (addressed) 244
processing of records (direct) 239
search arguments, specifying 273
testing for the last processed record 259
RDF (record definition field) 113
READPW (read password parameter) 125
real mode operation 14, 37
record allocation (SAM ESDS), specifying 29
record definition field (RDF) 113
record insert macro (PUT)
See PUT macro
RECORDFORMAT parameter (for SAM ESDS files) 151
records
adding 111
allocation size (SAM ESDS), specifying 29
deleting 113
examples CI/CI splits 114
loading into a file 111
loading into a file, considerations for 111
mass insertion 111
number of (SAM ESDS) 29
reclaiming space 113
retrieval of
See GET macro
size computation for data component 88
size relationship to control interval 88
size, specifying 88
write check 110
write check default 110
writing of
See PUT macro
records delete macro (ERASE)
See ERASE macro

records retrieve macro (GET)
positioning for processing, keeping 225
positions for request macros in process 225
RECORDS specification (SAM ESDS) 29
RECORDSIZE parameter 88
RECORDSIZE parameter (for SAM ESDS files) 158
recoverable catalog
See catalog (recoverable)
recovery of resources
a guide to 139
catalog cannot be opened 146
catalog damaged 138
catalog unusable 146
catalog volume unusable 147
catalogs 137
considerations (catalogs) 137
considerations (files) 134
considerations (volumes) 134
current type of data 139
downlevel type of data 139
duplicate data condition 144
file cannot be opened 145
file completely unreadable 145
file not properly closed 143
file partially unreadable 145
files cannot be opened 146
inaccessible file 145
incomplete write to disk 144
incorrect high RBA 144
levels of data recovery 139
procedures 143
quick recovery, preparing for 150
tools 139
volume is inaccessible 148
what it is 139
with SAM ESDS files 153
RECOVERY parameter 110
RECSIZE specification (SAM ESDS) 29
relating deferred requests 263
relationship of catalog entries to VSE/VSAM files and volumes 135
relative block of extent 39
relative track of extent 39
REPRO (IDCAMS command)
use of 136, 137
using with SAM ESDS files 172
request macros
examples 267
overview 183
return codes 289
request parameter list (RPL) 186, 231
requirements
and restrictions with modeling objects 63
for creating SAM ESDS file from SAM file 151
for job control 23
for lock file 21
ISAM to VSE/VSAM, converting from 323
spanned record, begin of 6
storage
See storage requirements
VOL1 label, location of 337
requirements *(continued)*
volume mounting 51
reset high RBA mismatch 147
resource pool build macro (BLDVRP)
operand notation 302
parameter list 312
resource pool delete macro (DLVRP)
operand notation 302
resource pool, building 207
resources protection
See protection of resources
resources, shared 191
restore
access to data 139
addressability of data 139
dialog for 12
generic names 56
generic names, using 56
RESTORE command
migration to large DASD 78
migration to SCSI disk 55
overview 3
use of 135
restrictions
large DASD 79
restrictions, data compression 71
retrieval of records 240, 245
return codes from BLDVRP 210
return codes from close routine 212
return codes from DLVRP 213
return codes from manipulation macros 291
return codes from OPEN 229
return codes from request macros 289
return codes from SHOWCAT 249
reusable (REUSE) file, disposition control for 155
reusable files 202
REUSE parameter 162
REUSE parameter with SAM ESDS work files 160
RMODE31 parameter in macros 194
RPL chain, positioning information for 234
RPL macro
See parameter list request macro (RPL)
RPL macro format 232

S

SAM (sequential access method)
file format differences to SAM ESDS 179
files, processing with VSE/VSAM 151
functions, restrictions with SAM ESDS 167
migrating to VSE/VSAM control 155
SAM access (managed)
// DLBL date parameter for managed-SAM file 27
// DLBL specifications, dependencies on 30
change programs 156
considerations 165
considerations for access to files 166
SAM access (managed) *(continued)*
considerations for access to work files 167
control block trace 351
differences to unmanaged-SAM access 165
DTF specifications, dependencies on 30
GETVIS space, specifying 167
modeling managed-SAM file 61
purpose of 165
requirements 17
steps in obtaining managed-SAM access 155
SAM access (unmanaged)
considerations 165
differences to managed-SAM access 165
disk-independence 167
steps in changing to managed-SAM access 155
SAM ESDS file
See ESDS file (SAM)
SAM file
converting to VSE/VSAM 2
creating back up 136
creating SAM ESDS files, requirement for 151
defining for use in VSE/VSAM data space 152
migrating to VSE/VSAM control 155
VSE/VSAM functions available 152
SAM file (managed)
// DLBL date parameter 27
disposition at CLOSE 30
GETVIS space, adjusting 37
modeling 61
options at OPEN 30
partition size, adjusting 37
possible restrictions 168
steps in obtaining managed-SAM access 155
SAM file (unmanaged)
advantages in changing to SAM ESDS files 152
steps in changing to managed-SAM access 155
SAM files and data compression 71
SAM files to VSE/VSAM, convert 2
Sampling in data compression 68
scratch VTOC label for a specified file 358
SCSI disk devices
FBA disk devices 80
migration using BACKUP 55
migration using RESTORE 55
restrictions 81
search order of catalogs 44
second close disposition 202
secondary allocation of file (SAM ESDS), records for 29
secondary allocation, minimize 133
security-verification exit 128
separate processing program from file
See TCLOSE macro
sequential positioning 233
sequential processing 239

sequential retrieval 240
serial-number of volume containing extent 39
shared resource (LSR) option in ACB 202
shared resources 191
shared resources, local
See local shared resources (LSR)
SHAREOPTIONS parameter
control file sharing 129
degrees of file sharing 130
purpose of 129
where to specify 129
with SAM ESDS files 166
SHAREOPTIONS(4) locking activity trace 351
sharing
catalogs across systems 131
cross-system data, specifying share options 131
cross-system user catalogs, specifying 131
DASDs, establish environment 129
data across system 131
data set names, advantages 192
data, protection of 129
file control blocks 192
files 129
files across systems 131
files and use of z/VSE LOCK facility 130
files, protection for 129
I/O buffers 192
integrity when opening a file through different ACBs 192
master catalog with shared user catalogs 131
options 130
sharing of data set name 192
SHOWCAT (display catalog) macro 189, 246
SHOWCAT macro
See catalog display macro (SHOWCAT)
SHOWCAT macro format 246
SHOWCB (display control block) macro 188, 251
SHOWCB macro 121
See control block display macro (SHOWCB)
SHOWCB macro format 252
SHRPOOL parameter in ACB 203
size of partition to load a program
See partition space
skip sequential insertion 243
skip sequential retrieval 243
skip-sequential retrieval 268
SNAP (IDCAMS command) 9, 141
SNAP dump program (IKQVEDA)
activating 351
disabling 352
dump numbers 350
enabling 352
examples 353
execution, setting up for 353
output of 354
purpose 350
SNAP dump program (IKQVEDA) *(continued)*
running 351
types of dumps 350
UPSI job control statement, setting of 353
space
allocation (dynamic file), suppressing 109
allocation (dynamic) with SAM ESDS 152, 164, 165
allocation for multiple volumes 102
allocation for multiple volumes (an exercise) 107
allocation options 108
allocation parameters 109
defaults 93
determination 93
dynamic allocation 5, 109
GETVIS space (SAM ESDS), specifying 167
management 6, 47
map, mismatch of 147
ownership 47
partition for time-dependent programs 37
suballocate data space 47
unused, cause of 113
utilization and CI size 90
SPACE FORCE command 142
spanned record, what it is 6
spanned records, handling 218, 237, 238
SPEED parameter 110
standard volume label (VOL1 label) 337
statistical information not updated 122
statistics mismatch 147
statistics on files 121
statistics provided by SHOWCB macro 256
storage
above/below the 16MB line 16
capacities of CKD devices 86
capacities of FBA devices 87
for loading programs 38
GETVIS for non-SVA-eligible routines 13
space for buffers (due to CIs) 94
storage requirements
buffers 14
considerations 13
control blocks 14
for index records 120
IDCAMS 18
if LSR is specified 16
if NSR is specified 15
ISAM Interface Program (IIP) 18
non-SVA-eligible routines 14
real mode operation 14
SVA for VSE/VSAM 13
VSE/VSAM 14
VSE/VSAM Backup/Restore Function 19
VSE/VSAM Space Management for SAM Function 17
suballocation of data space 109, 336
supervisor buffers, specifying 22
support of 3390-9 disk device 77
switching from direct to keyed-sequential retrieval 274
SYNAD exit routine 185, 220
syntax checking (processing option) 38
SYS (IPL command) 21, 22
system defaults, overriding through modeling 57
system work file (IJSYSnn) support 168

T

TCLOSE macro
description 257
format 257
operands 257
use 188
temporary closing of a file 257
terminating a request 214
terms explained 363
TESTCB (test control block) macro 188, 257
TESTCB macro 121
See control block test macro (TESTCB)
TESTCB macro format 258
threshold, free space 112
time stamp
entry in volume 50
entry in VTOC 50
error codes on mismatch 338
field in volume 338
time-dependent programs, partition space for 37
tools for resources protection and recovery 139
track size (CKD devices) relationship to min CA size 85
track space used for data component 90
tracks per cylinder (CKD devices) 86
transaction ID (in RPL macro) 187, 238, 263
transporting files
See migrating

U

unique control block for access methods, generating 331
unique file
See files (UNIQUE)
UNIQUE parameter 48, 109
unmanaged-SAM access
See SAM access (unmanaged)
unmanaged-SAM file
See SAM file (unmanaged)
UNORDERED parameter 103
update, addressed sequential 285
update, keyed direct 283
update, keyed sequential 282
UPDATEPW (update password parameter) 125
updating VSE/VSAM files 243, 245
upgrade record, what it is 350
upgrade set (alternate index)
buffer allocation 99
buffer allocation for path entry 98
data output from catalog check 348

upgrade set (alternate index) *(continued)*
partition requirements (if NSR) 15, 16
UPGRADE attribute and share options 130
upgrade record, what it is 350
UPSI statement (with maintain VTOC/VOL1 labels) 355
UPSI statement (with SNAP dump) 353
USECLASS parameter 84
user catalog
See catalog (user)
user catalog // DLBL statement 23
user I/O routines 317
user security-verification routine (USVR)
action with master password (MASTERPW) 125
explained 128
register content 128
specifying name of routine 128
user-written program for file back up 136
utilities for data integrity 132

V

verification of passwords (user-written routine) 128
VERIFY (IDCAMS command)
// DLBL required 25
compare catalog with EOF indicator 143
incorrect high RBA, handling of 144
mismatch, correct a 147
overview 10
using with SAM ESDS files 173
virtual disk support
preparation procedures 51
restrictions 52
use with VSE/VSAM 51
work files on 51
virtual mode operation 14, 37
virtual tape 71, 82
VM, transfer files from VSE to 7
VOL1 label, processing 337
VOL1 label, purpose of 333
volume
automatic device assignment 22
back up and recovery considerations 134
back up considerations/methods 136
data space suballocation 336, 339
defining data space on 48
defining for SAM ESDS files 160
entries in catalog 135
entries in catalog, purpose of 47
entries in catalog, scope of 49
extent information, specifying 39
extent overlap, action on 337
file name associated with 27
identifier, when written 333
indicator of ownership 49
information mismatch 148
label maintain program (IKQVDU) 355
layout of 334
lists with object modeling 65
volume *(continued)*
logical unit 39
migrating files from one volume to another 56
mismatch of catalog entry 147
mounting for file processing 339
mounting, ways of 22
mounting, when needed 51
ownership and nonVSAM files 48
ownership, records of 48
ownership, removing from catalog 49
portability 41
protection considerations 134
relationship of data space, files, labels 334
relative block 39
relative track 39
separate for indexes and data 121
separation for data integrity 132
serial number 39
space ownership by catalogs (restrictions) 41
space ownership, determining 138
time stamp 338
time stamp entry, when updated 50
VOL1 label, purpose of 333
volume layout example 335
VSE/Fast Copy utility, use of 136
volume back up
See volume protection
volume protection
back up considerations/methods 136
backup copy, creating a 136
VSE/Fast Copy utility, use of 136
volume space
define for VSAM and nonVSAM files 132
ownership, determine 138
volume table of contents (VTOC)
data component names (user-specified) 50
data space names, entries for 50
data space names, generation of 50
delete (scratch) 358
deletion of labels, when it occurs 336
Device Support Facilities (DSF), when used 334
format-4 label, purpose of 334
format-4 label, when written 334
index component names (user-specified) 50
label entries 48
label maintain program (IKQVDU) 355
label processing 338
label processing overview 335
label that defines the VTOC 334
labels for FBA devices, residence of 336
record of physical extents (unique files) 135
relating generated and user-specified names 50
relationship to labels 334
scratching label for a specified file 358
volume table of contents (VTOC) *(continued)*
size for FBA devices, specifying 336
time stamp 338
time stamp entry, when updated 50
volume time stamp 50, 338
VSAM
support of Large DASD 77
VSAM.COMPRESS.CONTROL 70
VSE/Fast Copy utility 132
VSE/VSAM access
to SAM ESDS files 169
to VSE/VSAM ESDS files 169
VSE/VSAM basic information
See basic information on VSE/VSAM
VSE/VSAM Compression Prediction Tool (IKQPRED)
examples 72
interpreting results 73
invocation 72
output description 74
process 73
VSE/VSAM DASD sharing facility 21
VSE/VSAM device dependencies 77
VSE/VSAM extended user buffering
new support 288
overview 287
using 288
VSE/VSAM for VM
DASD considerations 132
VSE/VSAM macros
See macros (VSE/VSAM)
VSE/VSAM Space Management for SAM Function
creating/using SAM ESDS files 151
GETVIS space default 37
IDCAMS commands, using 170
partition requirements 38
partition size 37
RECORDS specification for SAM ESDS files 29
RECSIZE specification for SAM ESDS files 29
storage requirements 17
use of 2, 152
VSE/VSAM support of Large DASD 77
VSE/VSAM virtual tape 82
VSE/VSAM Virtual Tape 82
VTAM similarities with VSE/VSAM 331
VTOC
See volume table of contents (VTOC)
VTOC utility (IKQVDU) 132, 355

W

words explained 363
work files
See files (WORK)
work files on virtual disk 51
wrap trace 351
write operations, deferring 262
WRITECHECK
data integrity 132
default when modeling 63
performance with NOWRITECHECK 110
purpose of 110, 142

WRITECHECK *(continued)*
with ALTER command 170
writing buffers 263
writing records
See PUT macro
WRTBFR (write buffer) macro 263
WRTBFR macro
See buffer write macro (WRTBFR)
WRTBFR macro format 264

Z

z/VSE Interactive Interface for users 11
z/VSE LOCK facility, use of 130

Readers' Comments — We'd Like to Hear from You

IBM z/VSE
VSE Central Functions
VSE/VSAM User's Guide and Application Programming
Version 7 Release 1

Publication No. SC33-8246-00

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

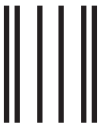
Company or Organization

Phone No.

Readers' Comments — We'd Like to Hear from You
SC33-8246-00

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

IBM Deutschland Entwicklung GmbH
Department 3248
Schoenaicher Strasse 220
D-71032 Boeblingen
Federal Republic of Germany

Fold and Tape | **Please do not staple** | Fold and Tape

Cut or Fold
Along Line

Program Number: 5686-066
5686-CF7

Printed in USA

SC33-8246-00

Spine information:

IBM z/VSE
VSE Central Functions

VSE/VSAM User's Guide and Application Programming

Version 7 Release 1 SC33-8246-00