

z/OS
3.2

DFSMSdfp Utilities



Note

Before using this information and the product it supports, read the information in [“Notices” on page 363.](#)

This edition applies to IBM® z/OS® 3.2 (5655-ZOS) and to all subsequent releases and modifications until otherwise indicated in new editions.

Last updated: 2025-09-30

© **Copyright International Business Machines Corporation 1979, 2025.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Figures.....	xi
Tables.....	xv
About this book.....	xix
Required product knowledge.....	xx
Referenced documents.....	xx
z/OS information.....	xxi
Notational Conventions.....	xxi
How to provide feedback to IBM.....	xxv
Summary of changes.....	xxvii
Summary of changes for z/OS 3.2.....	xxvii
Summary of changes for z/OS 3.1.....	xxvii
Chapter 1. Introduction.....	1
Guide to utility program functions.....	1
System utility programs.....	4
Data set utility programs.....	5
Control.....	6
Job Control Statements.....	6
Utility control statements.....	7
Special referencing aids.....	8
Chapter 2. GDKUTIL (Cloud Object Utility) Program.....	11
Control.....	11
Job Control.....	11
BUCKET Statement.....	12
CREDSNAM Statement.....	13
EXEC Statement.....	14
SYSOUT Statement.....	14
SYSPRINT Statement.....	14
OBJNAME Statement.....	14
OBJNAMEX Statement.....	14
LOCAL Statement.....	15
LOCNAME Statement.....	15
SYSIN Statement.....	16
GDKUTIL Examples.....	26
Example 1: Retrieve an object into a z/OS UNIX file.....	26
Example 2: Send a z/OS Sequential data set to a Cloud Object.....	26
Example 3: Create a Bucket.....	27
Example 4: List Objects in a cloud storage bucket with filtering.....	27
Example 5: Upload multiple files to cloud storage.....	27
Example 6: Download multiple objects from cloud storage.....	27
Example 7: Delete multiple objects from cloud storage.....	28
Example 8: Send a GDG version to a Cloud Object with Metadata tags.....	28
Example 9: New default credentials for the CLOUD1 provider.....	29
Example 10: Delete Credentials.....	29

Example 11: List Credential Entries.....	29
Example 12: Compress an object.....	29
Example 13: Decompress and download the object	30
Example 14: Upload a data set, keeping record information	30
Chapter 3. IEBCOMPR (Compare Data Sets) Program.....	31
Input and Output.....	32
Control.....	32
Job Control Statements.....	32
Utility Control Statements.....	33
IEBCOMPR Examples.....	35
Example 1: Compare Data Sets that Reside on Tape.....	35
Example 2: Compare Sequential Data Sets that Reside on Tape.....	36
Example 3: Compare Sequential Data Sets Written at Different Densities.....	36
Example 4: Compare Sequential Data Sets—Input Stream and Tape Input.....	37
Example 5: Copy and Compare Sequential Data Set in Two Job Steps.....	37
Example 6: Compare Two Partitioned Data Sets.....	38
Example 7: Copy and Compare Partitioned Data Set in Two Job Steps.....	38
Example 8: Compare Two PDSEs.....	39
Chapter 4. IEBCOPY (Library Copy) Program.....	41
Converting Load Modules to Program Objects or the Reverse.....	41
Converting Partitioned Data Sets to PDSEs.....	42
Copying Data Sets.....	42
Merging Data Sets.....	42
Increasing Directory Space for a Partitioned Data Set.....	42
Unloading (Backing up) Data Sets.....	43
Copying Directory Information between a Partitioned Data Set and a PDSE.....	43
Loading or Copying Unload Data Sets.....	43
IEBCOPY Unload Data Set DCB Parameters.....	43
Selecting Members to be Copied, Unloaded, or Loaded.....	44
Excluding Members from a Copy Operation.....	45
Copying Members That Have Alias Names (COPY Statement).....	45
Replacing Members in a Data Set.....	46
Renaming Selected Members.....	47
Copying Program Objects (COPYGRP and COPYGROUP Statements).....	47
COPYGROUP.....	48
Replacing Program Objects.....	48
Compressing a Partitioned Data Set.....	49
Processing Considerations for Compress.....	49
Altering Load Modules.....	50
Copying and Reblocking Load Modules.....	50
Changed COPYMOD operation.....	51
How IEBCOPY Uses Virtual Storage for Tables and Buffers.....	51
How IEBCOPY Allocates Tables and Buffers.....	51
Avoiding the Need to Supply Control Statements.....	52
Restrictions.....	52
Input and Output.....	54
Control.....	54
Job Control Statements.....	54
Utility Control Statements.....	59
IEBCOPY Examples.....	69
Example 1: Copy an entire data set.....	71
Example 2: Merge four data sets.....	72
Example 3: Copy and Replace Selected Members of a Data Set.....	73
Example 4: Unload and Compress a Data Set.....	75
Example 5: Merge Data Sets and Compress the Merged Data Set.....	76

Example 6: Multiple Copy Operations with One Output Data Set.....	78
Example 7: Multiple Copy Operations with Different Output Data Sets.....	81
Example 8: Loading a Data Set.....	86
Example 9: Unload Selected Members, Load, Copy and Merge.....	87
Example 10: Alter Load Modules in Place.....	88
Example 11: Replace a Load Module Using COPYMOD.....	89
Example 12: Reblock Load Library and Distribute It to Different Device Types.....	89
Example 13: Convert a Partitioned Data Set to a PDSE.....	90
Example 14: Copy Groups from a PDSE to a PDSE.....	91
Example 15: Copy Groups from a PDSE to a PDSE with Replace.....	91
Example 16: Copy a Selected Group from a PDSE to a PDSE.....	92
Example 17: Copy Selected Members and their Aliases from a PDS to a PDS.....	92
Example 18: Copy Selected Members from a PDS to a PDS.....	93
Chapter 5. IEBDG (Test Data Generator) Program.....	95
Selecting a Pattern.....	95
IBM-Supplied Patterns.....	95
User-Specified Patterns.....	96
Modifying Fields in a Record.....	97
Input and Output.....	98
Control.....	98
Job Control Statements.....	98
Utility Control Statements.....	100
IEBDG Examples.....	110
Example 1: Place Binary Zeros in Records Copied from Sequential Data Set.....	111
Example 2: Ripple 10-byte Alphabetic Pattern.....	111
Example 3: Create Output Records from Utility Control Statements.....	112
Example 4: Use Members and Input Records as Basis of Output Member.....	113
Example 5: Create Records in Three Output Data Sets and Write them to Three Partitioned Data Set Members.....	115
Example 6: Construct Records with Your Own Patterns.....	117
Chapter 6. IEBEDIT (Edit Job Stream) Program.....	119
Input and Output.....	119
Control.....	119
Job Control Statements.....	119
Utility Control Statement.....	120
IEBEDIT Examples.....	121
Example 1: Copy One Job.....	122
Example 2: Copy Steps from Three Jobs.....	122
Example 3: Include Step from One Job, Exclude Step from Another.....	123
Example 4: Copy Statement for JOBA and JOB STEPF.....	123
Example 5: Copy Entire Input Data Set.....	124
Example 6: Copy Entire Data Set to Include New Delimiter.....	124
Chapter 7. IEBGENER (Sequential Copy/Generate Data Set) Program.....	127
Creating a Backup Copy.....	127
Producing a Partitioned Data Set or PDSE from Sequential Input.....	127
Adding Members to a Partitioned Data Set or PDSE.....	128
Producing an Edited Data Set.....	129
Changing Logical Record Length.....	130
Using IEBGENER with Double-Byte Character Set Data.....	130
Input and Output.....	131
Control.....	131
Job Control Statements.....	131
Utility Control Statements.....	135
IEBGENER Examples.....	141

Example 1: Print a Sequential Data Set.....	142
Example 2: Create a Partitioned Data Set from Sequential Input.....	142
Example 3: Convert Sequential Input into Partitioned Members.....	143
Example 4: In-stream Input, Sequential Data Set to Tape Volume.....	144
Example 5: Produce Blocked Copy on Tape from Unblocked Disk File.....	144
Example 6: Edit and Copy a Sequential Input Data Set with Labels.....	145
Example 7: Edit and Copy a Sequential z/OS UNIX File to a Sequential Data Set.....	146
Example 8: Edit Double-Byte Character Set Data.....	147
Chapter 8. IEBIMAGE (Create Printer Image) Program.....	149
Storage Requirements for SYS1.IMAGELIB Data Set.....	149
Maintaining the SYS1.IMAGELIB Data Set.....	150
General Module Structure.....	151
Naming Conventions for Modules.....	152
Creating a Forms Control Buffer Module.....	152
3800 FCB Module Structure.....	152
4248 FCB Module Structure.....	153
FCB Module Listing.....	155
Creating a Copy Modification Module.....	156
COPYMOD Module Structure.....	156
COPYMOD Module Listing.....	156
Creating a Character Arrangement Table Module.....	157
TABLE Module Structure.....	158
TABLE Module Listing.....	159
Creating a Graphic Character Modification Module.....	160
GRAPHIC Module Structure.....	160
GRAPHIC Module Listing.....	161
Creating a Library Character Set Module.....	162
CHARSET Module Structure.....	163
CHARSET Module Listing.....	163
Input and Output.....	164
Control.....	165
Job Control Statements.....	165
Utility Control Statements.....	166
IEBIMAGE Examples.....	181
Example 1: Build a New 3800 Forms Control Buffer Module.....	182
Example 2: Replace a 3800 Forms Control Buffer Module.....	182
Example 3: Replace a 3800 Forms Control Buffer Module.....	183
Example 4: Build a New 3800 Forms Control Buffer Module.....	184
Example 5: Replace the 3800 Forms Control Buffer Module STD3.....	184
Example 6: Build a New 3800 Forms Control Buffer Module for Additional ISO Paper Sizes.....	185
Example 7: Build a 4248 Forms Control Buffer Module.....	185
Example 8: Build a New Copy Modification Module.....	186
Example 9: Build a New Copy Modification Module from an Existing Copy.....	187
Example 10: Add a New Character to a Character Arrangement Table Module.....	187
Example 11: Build a New Character Arrangement Table Module from an Existing Copy.....	188
Example 12: Build Graphic Characters in a Character Arrangement Table Module.....	189
Example 13: Delete Graphic References From a Character Arrangement Table Module.....	189
Example 14: List the World Trade National Use Graphics Graphic Character Modification Module.....	190
Example 15: Build a Graphic Character Modification Module from the Character Modification Module World Trade GRAFMOD.....	190
Example 16: Build a New Graphic Character Modification Module and Modify a Character Arrangement Table to Use It.....	191
Example 17: Build a Graphic Character Modification Module from Multiple Sources.....	192
Example 18: Define and Use a Character in a Graphic Character Modification Module.....	193
Example 19: List a Library Character Set Module.....	194

Example 20: Build a Library Character Set Module.....	195
Example 21: Build a Library Character Set Module and Modify a Character Arrangement Table to Use It.....	195
Example 22: Build a Library Character Set Module from Multiple Sources.....	196
Chapter 9. IEBISAM Program.....	199
Chapter 10. IEBPDSE (PDSE Validation) Program.....	201
Input and Output.....	201
Control.....	201
Job Control Statements.....	201
IEBPDSE Examples.....	202
Example 1: Validate one PDSE.....	202
Example 2: Validate two PDSEs.....	202
Example 3: Validate a PDSE with the DUMP option.....	202
Example 4: Perform pending delete processing without analyzing the PDSE.....	202
Chapter 11. IEBTPCH (Print-Punch) Program.....	203
Printing or Punching an Entire Data Set or Selected Member.....	203
Printing or Punching an Edited Data Set.....	203
Printing or Punching Double-Byte Character Set Data.....	204
Printing or Punching Selected Records.....	204
Printing or Punching a Partitioned Directory.....	204
Printing or Punching to Disk or Tape.....	204
Input and Output.....	205
Control.....	205
Job Control Statements.....	205
Utility Control Statements.....	206
IEBTPCH Examples.....	214
Example 1: Print Partitioned Data Set.....	215
Example 2: Punch Sequential Data Sets.....	215
Example 3: Duplicate a Card Deck.....	216
Example 4: Print Sequential Data Set According to Default Format.....	216
Example 5: Print Sequential Data Set According to User Specifications.....	216
Example 6: Print Three Record Groups.....	217
Example 7: Print a Pre-Formatted Data Set.....	218
Example 8: Print Directory of a Partitioned Data Set.....	218
Example 9: Print Selected Records of a Partitioned Data Set.....	219
Example 10: Convert to Hexadecimal and Print Partitioned Data.....	219
Example 11: Print Member Containing DBCS Data.....	220
Chapter 12. IEBUPDTE (Update Data Set) Program.....	221
Creating and Updating Data Set Libraries.....	221
Modifying an Existing Data Set.....	221
Changing Data Set Organization.....	221
Input and Output.....	221
Control.....	222
Job Control Statements.....	222
Utility Control Statements.....	224
IEBUPDTE Examples.....	232
Example 1: Place Two Procedures in SYS1.PROCLIB.....	233
Example 2: Create a Three-Member Library.....	234
Example 3: Create New Library Using SYS1.MACLIB as a Source.....	234
Example 4: Update a Library Member.....	235
Example 5: Create New Master Data Set and Delete Selected Records.....	236
Example 6: Create and Update a Library Member.....	236
Example 7: Insert Records into a Library Member.....	237

Example 8: Renumber and Insert Records into a Library Member.....	238
Example 9: Create a Sequential Data Set from Card Input.....	240
Example 10: Copy Sequential Data Set from One Volume to Another.....	241
Example 11: Create a New Partitioned Data Set.....	241
Chapter 13. IEHINITT (Initialize Tape) Program.....	243
Placing a Standard Label Set on Magnetic Tape.....	245
Using DFSMSrmm.....	245
Replacing the Key Encrypting Key Structure.....	246
Input and Output.....	246
Control.....	247
Job Control Statements.....	247
Utility Control Statements.....	248
IEHINITT Examples.....	251
Example 1: Write EBCDIC Labels on Three Tapes.....	252
Example 2: Write an ISO/ANSI Label on a Tape.....	252
Example 3: Place Two Groups of Serial Numbers on Six Tape Volumes.....	253
Example 4: Place Serial Number on Eight Tape Volumes.....	253
Example 5: Write EBCDIC Labels in Different Densities.....	253
Example 6: Write Serial Numbers on Tape Volumes at Two Densities.....	254
Example 7: Write an ISO/ANSI Label with an Access Code.....	254
Example 8: Write on a tape following labeling without demounting and remounting.....	254
Example 9: Rekey one tape volume.....	255
Example 10: Multiple tapes specified for SER keyword.....	255
Example 11: Three tapes with NUMBTAPE specified.....	255
Example 12: Multiple REKEY Control Statements.....	255
Example 13: Multiple Control Statements with NUMBTAPE.....	256
Example 14: Printout of INITT Statement Specifications and Initial Volume Label Information...	256
Chapter 14. IEHLIST (List System Data) Program.....	257
Listing a Partitioned Data Set or PDSE Directory.....	257
Edited Format.....	257
Unedited (Dump) Format.....	258
Listing a Volume Table of Contents.....	258
Edited Format.....	259
Input and Output.....	260
Control.....	261
Job Control Statements.....	261
Utility Control Statements.....	262
IEHLIST Examples.....	264
Example 1: List Partitioned Directories Using DUMP and FORMAT.....	265
Example 2: List Non-indexed Volume Table of Contents.....	265
Chapter 15. IEHMOVE (Move System Data) Program.....	267
Considering Volume Size Compatibility.....	268
Allocating Space for a Moved or Copied Data Set.....	269
Reblocking Data Sets.....	270
Using IEHMOVE with RACF®.....	270
Moving or Copying a Data Set.....	270
Sequential Data Sets.....	271
Partitioned Data Sets.....	271
BDAM Data Sets.....	274
Multivolume Data Sets.....	274
Unloaded Data Sets.....	274
Unmovable Data Sets.....	275
Moving or Copying a Group of Cataloged Data Sets.....	275
Moving or Copying a Volume of Data Sets.....	276

Input and Output.....	276
Control.....	277
Job Control Statements.....	277
Utility Control Statements.....	280
IEHMOVE Examples.....	289
Chapter 16. IEHPROGM (Program Maintenance) Program.....	295
Scratching or Renaming a Data Set or Member.....	295
Maintaining Data Set Passwords.....	296
Adding Data Set Passwords.....	298
Replacing Data Set Passwords.....	298
Deleting Data Set Passwords.....	298
Listing Password Entries.....	298
Input and Output.....	299
Control.....	299
Job Control Statements.....	299
Utility Control Statements.....	301
IEHPROGM Examples.....	306
Example 1: Scratch Temporary System Data Sets.....	307
Example 2: Scratch and Uncatalog Two Data Sets.....	307
Example 3: Rename a Multi-Volume Data Set Catalog.....	307
Example 4: Uncatalog Three Data Sets.....	308
Example 5: Rename a Data Set and Define New Passwords.....	308
Example 6: List and Replace Password Information.....	309
Example 7: Rename a Partitioned Data Set Member.....	309
Chapter 17. IFHSTATR (List ESV Data) program.....	311
Assessing the quality of tapes in a library.....	311
Input and output.....	311
Control.....	313
IFHSTATR example.....	313
Appendix A. Invoking Utility Programs from an Application Program.....	315
IEBCOPY User-Exit Parameter.....	316
Invocation of IEBCOPY.....	319
IEBCOPY User Exit Routines.....	319
Building Parameter Lists.....	325
Options List.....	326
ddname List.....	326
Page Header Parameter.....	327
Return Codes.....	328
IEBCOMPR Return Codes.....	328
IEBCOPY Return Codes.....	328
IEBDG Return Codes.....	329
IEBEDIT Return Codes.....	330
IEBGENER Return Codes.....	330
IEBIMAGE Return Codes.....	330
IEBPDSE Return Codes.....	331
IEBPTPCH Return Codes.....	331
IEBUPDTE Return Codes.....	332
IEHINITT Return Codes.....	332
IEHLIST Return Codes.....	332
IEHMOVE Return Codes.....	333
IEHPROGM Return Codes.....	333
Appendix B. Unload partitioned data set format.....	335
Introduction.....	335

Records present in an unload data set.....	335
Different unload data set formats.....	336
Detailed record descriptions.....	336
Appendix C. Specifying User Exits with Utility Programs.....	343
General Guidance.....	343
Register Contents at Entry to Routines from Utility Programs.....	344
Programming Considerations.....	344
Returning from an Exit Routine.....	344
Parameters Passed to Label Processing Routines.....	346
Parameters Passed to Nonlabel Processing Routines.....	346
Processing User Labels.....	347
Processing User Labels as Data Set Descriptors.....	347
Exiting to a Totaling Routine.....	348
Processing User Labels as Data.....	348
Using an Exit Routine with IEBDG.....	349
Appendix D. IEHLIST VTOC Listing.....	351
IEHLIST Sample Output	351
IEHLIST Sample Output for Data Sets Supporting Extended Attribute DSCBs	352
IEHLIST Sample Output— a table of contents of a volume that has data sets that support extended attribute DSCBs.....	352
IEHLIST DUMP Sample Output— format 8 and 9 DSCBs present for data sets that support the extended attribute DSCBs.....	355
IEHLIST Sample Output Showing Vendor Data.....	356
Explanation of fields in IEHLIST formatted VTOC listing.....	357
Appendix E. Accessibility.....	361
Notices.....	363
Terms and conditions for product documentation.....	364
IBM Online Privacy Statement.....	365
Policy for unsupported hardware.....	365
Minimum supported hardware.....	365
Programming Interface Information.....	366
Trademarks.....	366
Glossary.....	367
Index.....	379

Figures

1. Partitioned directories whose data sets can be compared by using IEBCOMPR.....	31
2. Partitioned directories whose data sets cannot be compared by using IEBCOMPR.....	32
3. Copying a partitioned data set—full copy.....	71
4. Copying from three input partitioned data sets.....	72
5. Selective copy with Replace specified on the member level.....	74
6. Compress-in-Place following full copy with “replace”specified.....	77
7. Multiple copy operations/copy steps (Part 1 of 2).....	80
8. Multiple copy operations/copy steps (Part 2 of 2).....	81
9. Multiple copy operations/copy steps within a job step (Part 1 of 3).....	84
10. Multiple copy operations/copy steps within a job step (Part 2 of 3).....	85
11. Multiple copy operations/copy steps within a job step (Part 3 of 3).....	86
12. IEBDG actions.....	97
13. Field selected from the input record for use in the output record.....	101
14. Default placement of fields within an output record using IEBDG.....	106
15. Placement of fields with specified output locations.....	106
16. Placement of fields with only some output locations specified.....	107
17. Creating output records with utility control statements.....	107
18. Output Records at job step completion.....	112
19. Output partitioned member at job step completion.....	114
20. Partitioned data set members at job step completion.....	116
21. Contents of output records at job step completion.....	117
22. Using IEBGENER to create a partitioned data set or PDSE from sequential input	128
23. Using IEBGENER to add members to a partitioned data set or PDSE	129

24. Editing a sequential data set using IEBGENER.....	130
25. How a sequential data set is edited and copied.....	146
26. 3800 General Module Header.....	151
27. 3800 FCB module structure.....	153
28. 4248 FCB module structure.....	153
29. 4248 FCB module control byte.....	154
30. 4248 FCB module data byte.....	154
31. IEBIMAGE listing of a forms control buffer module.....	155
32. Structure of the copy modification module	156
33. IEBIMAGE listing of three segments of a copy modification module.....	157
34. Module structure of the character arrangement table.....	158
35. Graphic character modification modules.....	158
36. IEBIMAGE listing of a character arrangement table module.....	159
37. 3800 graphic character modification module structure for one character.....	161
38. IEBIMAGE listing of two segments of a graphic character modification module.....	162
39. 3800 Model 3 library character set module structure for one character.....	163
40. IEBIMAGE Listing of two segments of a library character set.....	164
41. Partitioned data set before and after an IEHMOVE copy operation.....	272
42. Merging two data sets using IEHMOVE.....	273
43. Merging three data sets using IEHMOVE.....	273
44. Relationship between the protection status of a data set and its passwords.....	297
45. Sample output from IFHSTATR.....	312
46. Directory record layout.....	339
47. Attribute record layout.....	340
48. Note list record layout.....	341

49. Member data record layout.....	341
50. Member data block layout.....	342
51. End-of-file block layout.....	342
52. System action at OPEN, EOVS, or CLOSE time.....	348
53. IEHLIST sample output—VTOC (for extended format sequential data sets).....	351
54. IEHLIST sample output—VTOC (for sequential, partitioned data sets and PDSEs).....	352
55. IEHLIST sample output—VTOC showing vendor data.....	356
56. DUMP sample output showing vendor data.....	357

Tables

1. Tasks and utility programs.....	1
2. System Utility Programs.....	5
3. Data set utility programs.....	5
4. Example directory.....	9
5. Job control statements for GDKUTIL.....	11
6. Job control statements for IEBCOMPR.....	32
7. IEBCOMPR utility control statements.....	33
8. Syntax of LABEL statement.....	34
9. IEBCOMPR example directory.....	35
10. Job control statements for IEBCOPY.....	55
11. IEBCOPY utility control statements.....	60
12. IEBCOPY utility control statements (continued).....	60
13. Multiple copy operations within a job step.....	60
14. Member name filter pattern examples.....	69
15. IEBCOPY example directory.....	69
16. IBM-supplied test data patterns.....	95
17. Job control statements for IEBDG.....	98
18. Syntax of EXEC statement.....	99
19. IEBDG utility control statements.....	100
20. Compatible IEBDG Operations.....	105
21. IEBDG example directory.....	110
22. Job control statements for IEBEDIT.....	119
23. IEBEDIT example directory.....	121

24. Job control statements for IEBGENER.....	131
25. Effect of output DD statements.....	134
26. IEBGENER utility control statements.....	136
27. IEBGENER example directory.....	142
28. Members per track (T) for various devices.....	150
29. Job control statements for IEBIMAGE.....	165
30. Utility control statements for IEBIMAGE.....	166
31. IEBIMAGE listing of a copy modification module with overrun notes.....	180
32. IEBIMAGE example directory.....	181
33. Job control statements for IEBPDSE.....	201
34. Job control statements for IEBTPCH.....	205
35. IEBTPCH utility control statements.....	206
36. IEBTPCH example directory.....	214
37. Job control statements for IEBUPDTE.....	222
38. IEBUPDTE utility control statements.....	224
39. NEW, MEMBER and NAME parameters of the function statements.....	228
40. IEBUPDTE example directory.....	232
41. Example of reordered sequence numbers.....	238
42. Reordered sequence numbers.....	239
43. IEHINITT job control statements.....	247
44. IEHINITT example directory.....	251
45. IEHLIST job control statements.....	261
46. IEHLIST utility control statements.....	262
47. IEHLIST example directory.....	265
48. Move and copy operations—DASD receiving volume with size compatible with source volume.....	268

49. Move and copy operations—DASD receiving volume with size incompatible with source volume.....	268
50. Move and copy operations—Non-DASD receiving volume.....	268
51. Moving and copying sequential data sets.....	271
52. Moving and copying partitioned data sets.....	271
53. Moving and copying a group of cataloged data sets.....	275
54. Moving and copying a volume of data sets.....	276
55. IEHMOVE job control statements.....	277
56. IEHMOVE utility control statements.....	280
57. IEHMOVE example directory.....	289
58. IEHPROGM job control statements.....	299
59. IEHPROGM utility control statements.....	301
60. IEHPROGM example directory.....	306
61. IFHSTATR job control statements.....	313
62. IEBCPARM mapping macro.....	316
63. IEBCREAS mapping macro.....	329
64. Contents of the COPYR1 descriptor record.....	337
65. Contents of the COPYR2 descriptor record.....	338
66. User-exit routines specified with utilities.....	343
67. Return codes that must be issued by user exit routines.....	345
68. Parameter lists for nonlabel processing exit routines.....	347

About this book

This book is intended to help system and application programmers use the z/OS™ DFSMS utility programs to manipulate system and user data and data sets. Most programs are discussed according to the following pattern:

1. Introduction and description of the functions that can be performed by the program. This description typically includes an overview of the program's use, definitions of terms, and illustrations.
2. Functions that are supported by the utility and the purpose of each function.
3. Input and output that are used and produced by the program.
4. Control of the program through job and utility control statements. Job control statements are described only insofar as their use in the utility program is peculiar to that program. Utility control statements are discussed fully.
5. Examples of using the program, including the job and utility control statements.

Use of the following utilities is not recommended:

- ICAPRTBL—The 3211 printer is no longer supported.
- IEBISAM is no longer distributed. VSAM should be used instead. For information on converting ISAM data sets to VSAM key-sequenced data sets, see *z/OS DFSMS Using Data Sets* and the REPRO command in *z/OS DFSMS Access Method Services Commands*.
- IEHMOVE—DFSMSdss and IEBCOPY are recommended.
- IEHPROGM—IDCAMS is recommended for all catalog and delete functions.
- IEHATLAS—The IEHATLAS program is no longer distributed. Use Device Support Facilities (ICKDSF) instead.

The information about these utilities is provided for compatibility only.

Several specialized utilities are not discussed in this book. The following list shows their names and functions, and indicates which book contains their explanation.

Utility	Function	Reference
IDCAMS	Allows users to define, manipulate, or delete VSAM data sets, define and manipulate integrated catalog facility catalogs, and copy, print, or convert SAM and ISAM data sets to VSAM data sets.	<i>z/OS DFSMS Access Method Services Commands</i>
Device Support Facilities (ICKDSF)	Used for the initialization and maintenance of DASD volumes.	<i>Device Support Facilities (ICKDSF) User's Guide and Reference</i>
DFSMSdss	DASD utility functions such as dump/restore and reduction of free space fragmentation.	<i>z/OS DFSMSdss Storage Administration</i> , <i>z/OS DFSMSdfp Storage Administration</i> ,
Offline IBM 3800 Utility (CIPOPS)	Printer library function, which is used with the IBM 3800 Tape-to-Printing Subsystem Feature.	<i>Offline 3800 Utility</i>
IEFBR14	Performs no action other than give return code 0 but the job scheduler checks JCL statements for syntax errors, allocates space for data sets and performs disposition processing.	<i>z/OS MVS JCL User's Guide</i>

Utility	Function	Reference
AMASPZAP (super zap)	Used to inspect and modify disk data sets.	z/OS MVS Diagnosis: Tools and Service Aids
IPCS, interactive program control system	For diagnosis, analysis, and printing of system and application problems by using system dumps and GTF tracing.	z/OS MVS IPCS User's Guide
EDGINERS	Write IBM and ISO/ANSI standard labels on magnetic tape volumes and automatic erasure and labeling of magnetic tapes.	z/OS DFSMSrmm Implementation and Customization Guide
SuperC	Compare and report on differences between data set contents.	<i>HLASM Toolkit Feature User's Guide</i> , z/OS ISPF User's Guide Vol I

Required product knowledge

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

- Applications that use tape at your installation
- DFSMS
- Method of allocation in MVS™
- Job control language (JCL)
- Data management
- System management facilities (SMF)
- Tape and DASD hardware
- Tape mount management

You should also be familiar with the information presented in the following referenced publications:

Referenced documents

The following publications are referenced in this book:

Publication Title	Order Number
z/OS MVS Programming: Authorized Assembler Services Guide	SA23-1371
z/OS MVS Programming: Assembler Services Reference ABE-HSP	SA23-1369
<i>IBM 3800 Printing Subsystem Programmer's Guide</i>	GC26-3846
<i>IBM 3800 Printing Subsystem Model 3 Programmer's Guide: Compatibility</i>	SH35-0051
z/OS MVS JCL Reference	SA23-1385
z/OS MVS JCL User's Guide	SA23-1386
z/OS DFSMS Access Method Services Commands	SC23-6846
z/OS DFSMS Installation Exits	SC23-6850
z/OS MVS Program Management: User's Guide and Reference	SA23-1393
z/OS DFSMS Macro Instructions for Data Sets	SC23-6852
z/OS DFSMS Managing Catalogs	SC23-6853
z/OS DFSMSdfp Storage Administration	SC23-6860
z/OS DFSMSdfp Advanced Services	SC23-6861

Publication Title	Order Number
<i>z/OS DFSMS Using Data Sets</i>	SC23-6855
<i>z/OS DFSMS Using Magnetic Tapes</i>	SC23-6858
<i>Reference Manual for the IBM 3800 Printing Subsystem Model 1</i>	GA26-1635
<i>z/OS MVS Diagnosis: Tools and Service Aids</i>	GA32-0905
<i>z/OS TSO/E Programming Services</i>	SA32-0973
<i>z/OS MVS Programming: Assembler Services Guide</i>	SA23-1368
<i>z/OS MVS System Messages, Vol 7 (IEB-IEE)</i>	SA38-0674
<i>z/OS MVS System Messages, Vol 8 (IEF-IGD)</i>	SA38-0675

z/OS information

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

When possible, this information uses cross-document links that go directly to the topic in reference using shortened versions of the document title. For complete titles and order numbers of the documents for all products that are part of z/OS, see *z/OS Information Roadmap*.

Notational Conventions

A uniform notation describes the syntax of utility control statements. This notation is not part of the language; it is merely a way of describing the syntax of the statements. The statement syntax definitions in this book use the following conventions:

[]

Brackets enclose an optional entry. You may, but need not, include the entry. Examples are:

```
[length]
[MF=E]
```

|

An OR sign (a vertical bar) separates alternative entries. You must specify one, and only one, of the entries unless you allow an indicated default. Examples are:

```
[REREAD | LEAVE]
[length | 'S']
```

{ }

Braces enclose alternative entries. You must use one, and only one, of the entries. Examples are:

```
BFTEK={S | A}
{K | D}
{address | S | O}
```

Sometimes alternative entries are shown in a vertical stack of braces. An example is:

```
MACRF={ { (R[C|P]) }
        { (W[C|P|L]) }
        { (R[C],W[C]) } }
```

In the example above, you must choose only one entry from the vertical stack.

...

An ellipsis indicates that the entry immediately preceding the ellipsis may be repeated. For example:

```
(dcbaddr, [(options)], . . .)
```

‘ ’

A ‘ ’ indicates that a blank (an empty space) must be present before the next parameter.

UPPERCASE BOLDFACE

Uppercase boldface type indicates entries that you must code exactly as shown. These entries consist of keywords and the following punctuation symbols: commas, parentheses, and equal signs. Examples are:

- CLOSE , , , ,TYPE=T
- MACRF=(PL,PTC)

UNDERSCORED UPPERCASE BOLDFACE

Underscored uppercase boldface type indicates the default used if you do not specify any of the alternatives. Examples are:

- [EROPT={ACC|SKP|ABE}]
- [BFALN={F|D}]

Lowercase *Italic*

Lowercase italic type indicates a value to be supplied by you, the user, usually according to specifications and limits described for each parameter. Examples are:

- *number*
- *image-id*
- *count*

keyword=device=list

The term *keyword* is replaced by VOL, FROM or TO.

The term *device* is replaced by a generic name, for example, 3380, or an esoteric name, for example, SYSDA.

For DASD, the term *list* is replaced by one or more volume serial numbers separated by commas. When there is more than one volume serial number, the entire *list* field must be enclosed in parentheses.

For tapes, the term *list* is replaced by either one or more "volume serial number, data set sequence number" pairs. Each pair is separated from the next pair by a comma. When there is more than one pair, the entire *list* field must be enclosed in parentheses; for example:
FROM=3480=(tapeA,1,tapeB,1).

REQUIRED KEYWORDS AND SYMBOLS

Entries shown IN THE FORMAT SHOWN HERE (notice the type of highlighting just used) must be coded exactly as shown. These entries consist of keywords and the following punctuation symbols: commas, parentheses, and equal signs. Examples are:

- CLOSE , , , ,TYPE=T
- MACRF=(PL,PTC)

Note: The format (the type of highlighting) that is used to identify this type of entry depends on the display device used to view a softcopy book. The published hardcopy version of this book displays this type of entry in uppercase boldface type.

DEFAULT VALUES

Values shown IN THE FORMAT SHOWN HERE (notice the type of highlighting just used) indicate the default used if you do not specify any of the alternatives. Examples are:

- [EROPT={ACC|SKP|ABE}]
- [BFALN={F|D}]

Note: The format (the type of highlighting) that is used to identify this type of entry depends on the display device used to view a softcopy book. The published hardcopy version of this book displays this type of value in underscored uppercase boldface type.

User Specified Value

Values shown *in the format shown here* (notice the type of highlighting just used) indicate a value to be supplied by you, the user, usually according to specifications and limits described for each parameter. Examples are:

- *number*
- *image-id*
- *count*

Note: The format (the type of highlighting) that is used to identify this type of entry depends on the display device used to view a softcopy book. The published hardcopy version of this book displays this type of value in lowercase italic type.

How to provide feedback to IBM

We welcome any feedback that you have, including comments on the clarity, accuracy, or completeness of the information. For more information, see [How to send feedback to IBM](#).

Summary of changes

This information includes terminology, maintenance, and editorial changes. Technical changes or additions to the text and illustrations for the current edition are indicated by a vertical line to the left of the change.

Note: IBM z/OS policy for the integration of service information into the z/OS product documentation library is documented on the z/OS Internet Library under [IBM z/OS Product Documentation Update Policy](http://www.ibm.com/docs/en/zos/latest?topic=zos-product-documentation-update-policy) (www.ibm.com/docs/en/zos/latest?topic=zos-product-documentation-update-policy).

Summary of changes for z/OS 3.2

The following content is new, changed, or no longer included in z/OS 3.2.

New

The following content is new.

September 2025 release

- None.

Changed

The following content is changed.

September 2025 release

- Updated [“SYSPRINT DD Statement”](#) on page 57.

Deleted

The following content is deleted.

September 2025 release

- None.

Summary of changes for z/OS 3.1

The following content is new, changed, or no longer included in z/OS 3.1.

New

The following content is new.

May 2025 release

- Added Keywords to support GDKUTIL program. For more information, see [“SYSIN Statement” on page 16](#) in support of APAR OA66579 (www.ibm.com/support/pages/apar/OA66579)(APAR OA66579 applies to z/OS 3.1 and above).
- Added CDA symbols to support GDKUTIL program. For more information, see [“SYSIN Statement” on page 16](#) in support of APAR OA66579 (www.ibm.com/support/pages/apar/OA66579)(APAR OA66579 applies to z/OS 3.1 and above).
- Added Metadata Key table to support GDKUTIL program. For more information, see [“SYSIN Statement” on page 16](#) in support of APAR OA66579 (www.ibm.com/support/pages/apar/OA66579)(APAR OA66579 applies to z/OS 3.1 and above).

- Added Example 14 in GDKUTIL Examples. For more information, see [“Example 14: Upload a data set, keeping record information”](#) on page 30 in support of APAR OA66579 (www.ibm.com/support/pages/apar/OA66579)(APAR OA66579 applies to z/OS 3.1 and above).

March 2025 release

- Added Keywords to support GDKUTIL program. For more information, see [“SYSIN Statement”](#) on page 16 in support of OA66536: NEW FUNCTION - DFSMSdftp CDA Compression Support (www.ibm.com/support/pages/apar/OA66536) (APAR OA66536 applies to z/OS 3.1 and above).
- Added Examples 12 and 13 in GDKUTIL Examples. For more information, see [“Example 12: Compress an object”](#) on page 29 and [“Example 13: Decompress and download the object”](#) on page 30 in support of OA66536: NEW FUNCTION - DFSMSdftp CDA Compression Support (www.ibm.com/support/pages/apar/OA66536) (APAR OA66536 applies to z/OS 3.1 and above).

December 2024 release

- New GDKUTIL updates and examples are added. For more information, see [“CREDSNAM Statement”](#) on page 13, [“Example 9: New default credentials for the CLOUD1 provider”](#) on page 29, [“Example 10: Delete Credentials”](#) on page 29, [“Example 11: List Credential Entries”](#) on page 29, [“Job Control”](#) on page 11, [“SYSIN Statement”](#) on page 16 and [“OBJNAMEX Statement”](#) on page 14 (APAR OA65990, which applies to both z/OS 2.5 and 3.1).

May 2024 release

- New GDKUTIL examples are added. For more information, see [“Example 3: Create a Bucket”](#) on page 27, [“Example 4: List Objects in a cloud storage bucket with filtering”](#) on page 27, [“Example 5: Upload multiple files to cloud storage”](#) on page 27, [“Example 6: Download multiple objects from cloud storage”](#) on page 27, and [“Example 7: Delete multiple objects from cloud storage”](#) on page 28.

September 2023 release

- SYSDUMP DD Statement is new and added to job control statements for IEBCOPY. For more information, see [“Job Control Statements”](#) on page 54 and [“SYSDUMP DD Statement”](#) on page 57.

Changed

The following content is changed.

May 2025 release

- Updated Download Command to support GDKUTIL program. For more information, see [“SYSIN Statement”](#) on page 16 in support of APAR OA66579 (www.ibm.com/support/pages/apar/OA66579)(APAR OA66579 applies to z/OS 3.1 and above).

May 2024 release

- GDKUTIL is updated. For more information, see [“BUCKET Statement”](#) on page 12, [“LOCNAME Statement”](#) on page 15, [“OBJNAME Statement”](#) on page 14, [“Job Control”](#) on page 11 and [“SYSIN Statement”](#) on page 16.

Chapter 1. Introduction

DFSMS provides utility programs to assist you in organizing and maintaining data. Utilities are simple programs that perform commonly needed functions. [“Guide to utility program functions” on page 1](#) will help you find the program that performs the function that you need.

Guide to utility program functions

You can use the DFSMS utility programs to perform a variety of tasks, as shown in [Table 1 on page 1](#). The "Task" column shows tasks that you may want to perform. The "Options" column more specifically defines the tasks. The "Primary Utility" column identifies the utility that is especially suited for the task. The "Secondary Utilities" column identifies other utilities that can be used to perform the task.

Table 1. Tasks and utility programs

Task	Options	Primary Utility	Secondary Utilities
Add	a member to a partitioned data set	IEBUPDATE, IEBGENER	IEBDG
	a password	IEHPROGM	
Alter in place	a load module	IEBCOPY	
Catalog	a data set in a catalog	IEHPROGM	
Change	data set organization	IEBUPDTE	IEBGENER, IEBPTPCH
	logical record length	IEBGENER	
Compare	z/OS UNIX System Services (z/OS UNIX) files such as HFS files	IEBCOMPR	
	partitioned data sets	IEBCOMPR	
	sequential data sets	IEBCOMPR	
	PDSEs	IEBCOMPR	
Compress	a partitioned data set	IEBCOPY	
Compress in place	a partitioned data set	IEBCOPY	
Convert to partitioned data set	an unloaded PDSE containing program objects cannot be loaded into a PDS. An unloaded PDSE containing data objects can be loaded into a PDS but all extended attributes will be lost.	IEBCOPY	
	sequential data sets	IEBGENER	IEBUPDTE
	a PDSE	IEBCOPY	
Convert to PDSE	a partitioned data set	IEBCOPY	
	an unloaded copy of a partitioned data set or PDSE	IEBCOPY	
	sequential data sets	IEBGENER	IEBUPDTE
Convert to sequential data set	a partitioned data set or PDSE	IEBGENER	IEBUPDTE

Table 1. Tasks and utility programs (continued)

Task	Options	Primary Utility	Secondary Utilities
Copy	a load module or load module library	IEBCOPY	
	a partitioned data set	IEBCOPY	IEHMOVE
	a volume of data sets (on tape or disk)	IEHMOVE	
	job steps	IEBEDIT	
	selected members of a partitioned data set	IEBCOPY	IEHMOVE
	sequential data sets	IEBGENER	IEHMOVE, IEBUPDTE, IEBPTPCH
	a PDSE	IEBCOPY	
	a group of PDSE members	IEBCOPY	
	selected members of a PDSE	IEBCOPY	
Create	a backup copy of a partitioned data set or PDSE	IEBCOPY	
	a character arrangement table module	IEBIMAGE	
	a copy modification module	IEBIMAGE	
	a 3800 or 4248 forms control buffer module	IEBIMAGE	
	a graphic character modification module	IEBIMAGE	
	a library character set module	IEBIMAGE	
	a library of partitioned members	IEBGENER	IEBUPDTE
	a member of a partitioned data set or PDSE	IEBGENER	IEBDG, IEBUPDTE
	a sequential output data set	IEBDG	IEBGENER, IEBPTPCH
	an indexed sequential data set	IEBDG	
	an output job stream	IEBEDIT	
Delete	a data set or member of a partitioned data set	IEHPROGM	
	password	IEHPROGM	
	catalog entries	IEHPROGM	
	records in a partitioned data set or PDSE member	IEBUPDTE	
Edit and convert to partitioned data set or PDSE	a sequential data set	IEBGENER	IEBUPDTE
Edit and copy	a job stream	IEBEDIT	
	a sequential data set	IEBGENER	IEBUPDTE, IEBPTPCH
Edit and list	error statistics by volume (ESV) records	IFHSTATR	
Edit and print	a sequential data set	IEBPTPCH	IEBGENER
Edit and punch	a sequential data set	IEBPTPCH	IEBGENER
Enter	a procedure into a procedure library	IEBUPDTE	
Exclude	a partitioned data set member from a copy operation	IEBCOPY	IEHMOVE
	a PDSE member from a copy operation	IEBCOPY	

Table 1. Tasks and utility programs (continued)

Task	Options	Primary Utility	Secondary Utilities
Expand	a partitioned data set or PDSE	IEBCOPY	
	a sequential data set	IEBGENER	
Generate	test data	IEBDG	
Include	changes to members or sequential data sets	IEBUPDTE	
	a partitioned data set member from a copy operation	IEBCOPY	IEHMOVE
	a PDSE member from a copy operation	IEBCOPY	
Indicate	double-byte character set string by supplying enclosing shift-out/shift-in characters	IEBGENER	IEBPTPCH
Insert records	into a partitioned data set or PDSE	IEBUPDTE	
Label	magnetic tape volumes	IEHINITT	
List	a password entry	IEHPROGM	
	a volume table of contents	IEHLIST	
	number of unused directory blocks and tracks	IEHLIST	IEBCOPY
	partitioned data set or PDSE directories	IEHLIST	IEHPROGM
	CVOL entries	IEHLIST	
Load	an unloaded partitioned data set to a partitioned data set	IEBCOPY	
	an unloaded data set	IEHMOVE	
	an unloaded partitioned data set to a PDSE (for non-load modules only)	IEBCOPY	
	an unloaded PDSE to a partitioned data set (for non-load modules only)	IEBCOPY	
	an unloaded PDSE to a PDSE	IEBCOPY	
Merge	partitioned data sets	IEBCOPY	IEHMOVE
	PDSEs	IEBCOPY	
	partitioned data sets and PDSEs	IEBCOPY	
Modify	a partitioned or sequential data set, or a PDSE	IEBUPDTE	
Move	a volume of data sets	IEHMOVE	
	partitioned data sets	IEHMOVE	
	sequential data sets	IEHMOVE	
Number records	in a new or old member of a partitioned data set or PDSE	IEBUPDTE	
Password protection	add a password	IEHPROGM	
	delete a password	IEHPROGM	
	list passwords	IEHPROGM	
	replace a password	IEHPROGM	

Table 1. Tasks and utility programs (continued)

Task	Options	Primary Utility	Secondary Utilities
Print	sequential data sets	IEBTPCH	IEBGENER, IEBUPDTE
	partitioned data sets or PDSEs	IEBTPCH	
	selected records	IEBTPCH	
	mixed strings of double-byte and single-byte character set data	IEBTPCH	IEBGENER
	double-byte character set data	IEBTPCH	IEBGENER
Punch	a partitioned data set member	IEBTPCH	
	a sequential data set	IEBTPCH	
	selected records	IEBTPCH	
	mixed strings of double-byte and single-byte character set data	IEBTPCH	IEBGENER
	Double-byte character set data	IEBTPCH	IEBGENER
Reblock	a load module	IEBCOPY	
	a partitioned data set or PDSE	IEBCOPY	
	a sequential data set	IEBGENER	IEBUPDTE
Re-create	a partitioned data set or PDSE	IEBCOPY	
Rename	member of a partitioned data set or PDSE	IEBCOPY	IEHPROGM
	a sequential or partitioned data set, or PDSE	IEHPROGM	
	moved or copied members of a partitioned data set	IEHMOVE	
Renumber	logical records	IEBUPDTE	
Remove	indication of a double-byte character set string by stripping off enclosing shift-out/shift-in characters	IEBGENER	
Replace	a password	IEHPROGM	
	logical records	IEBUPDTE	
	records in a member of a partitioned data set or PDSE	IEBUPDTE	
	selected members of a PDSE	IEBCOPY	IEBUPDTE
	selected members of a partitioned data set	IEBCOPY	IEBUPDTE, IEHMOVE
Scratch	data sets	IEHPROGM	
Uncatalog	data sets	IEHPROGM	
Unload	a partitioned data set	IEBCOPY	IEHMOVE
	a sequential data set	IEHMOVE	
	a PDSE	IEBCOPY	
Update in place	a partitioned data set or PDSE	IEBUPDTE	

System utility programs

System utility programs are used to list or change information that is related to data sets and volumes, such as data set names, catalog entries, and volume labels. Most functions that system utility programs can perform are performed more efficiently with other programs, such as IDCAMS, ISMF, or DFSMSrmm.

Table 2 on page 5 is a list of system utility programs and their purpose.

Table 2. System Utility Programs

System Utility	Alternate Program	Purpose
*IEHINITT	DFSMSrmm EDGINERS	To write standard labels on tape volumes
IEHLIST	ISMF, PDF 3.4	To list system control data
*IEHMOVE	DFSMSdss, IEBCOPY	To move or copy collections of data
IEHPROGM	Access method services, PDF 3.2	To build and maintain system control data
*IFHSTATR	DFSMSrmm, EREP	To select, format, and write information about tape errors from the IFASMFDG tape

*These programs provide functions that are better performed by newer applications, such as ISMF or DFSMSrmm or DFSMSdss. IBM continues to ship these programs for compatibility with the *supported* older system levels.

The system utilities listed next support user data sets allocated with any valid combination of the following dynamic allocation options: XTIO, UCB NOCAPTURE, and DSAB above the line.

- IEHINITT
- IEHLIST
- IEHPROGM

The system utilities listed next do not support user data sets allocated with any of the following dynamic allocation options: XTIO, UCB NOCAPTURE, and DSAB above the line.

- IEHMOVE
- IFHSTATR

Data set utility programs

You can use data set utility programs to reorganize, change, or compare data at the data set or record level. These programs are controlled by JCL statements and utility control statements.

These utilities allow you to manipulate partitioned, sequential or indexed sequential data sets, or partitioned data sets extended (PDSEs), which are provided as input to the programs. You can manipulate data ranging from fields within a logical record to entire data sets.

The data set utilities included in this publication cannot be used with VSAM data sets. Information about VSAM data sets can be found in [z/OS DFSMS Using Data Sets](#).

Table 3 on page 5 is a list of data set utility programs and their use.

Table 3. Data set utility programs

Data Set Utility	Use
*IEBCOMPR, SuperC, (PDF 3.12)	Compare records in sequential or partitioned data sets, or PDSEs
IEBCOPY	Copy, compress, or merge partitioned data sets or PDSEs; add RLD count information to load modules; select or exclude specified members in a copy operation; rename or replace selected members of partitioned data sets or PDSEs
IEBDG	Create a test data set consisting of patterned data

Table 3. Data set utility programs (continued)

Data Set Utility	Use
IEBEDIT	Selectively copy job steps and their associated JOB statements
IEBGENER or ICEGENER	Copy records from a sequential data set or convert a data set from sequential organization to partitioned organization
IEBIMAGE	Modify, print, or link modules for use with the IBM 3800 Printing Subsystem, the IBM 3262 Model 5, or the 4248 printer
IEBPTPCH or PDF 3.1 or 3.6	Print or punch records in a sequential or partitioned data set
IEBUPDTE	Incorporate changes to sequential or partitioned data sets, or PDSEs

*These programs provide functions that are better performed by newer applications, such as ISMF or DFSMSrmm or DFSMSdss. IBM continues to ship these programs for compatibility with the *supported* older system levels.

The data set utilities listed next support user data sets allocated with any valid combination of the following dynamic allocation options: XTIO, UCB NOCAPTURE, and DSAB above the line.

- IEBCOPY
- IEBGENER

The data set utilities listed next do not support user data sets allocated with any of the following dynamic allocation options: XTIO, UCB NOCAPTURE, and DSAB above the line.

- IEBCOMPR
- IEBDG
- IEBEDIT
- IEBIMAGE
- IEBPTPCH
- IEBUPDTE

Control

System and data set utility programs are controlled by job control and utility control statements. The job control and utility control statements necessary to use utility programs are provided in the major discussion of each utility program.

Job Control Statements

You can start a system or data set utility program in the following ways:

- Place job control statements in a file and give the file to JES to run, for example, by the TSO SUBMIT command.
- Place job control statements, placed in a procedure library and run them with the MVS operator START command or include them in a JOB with the EXEC job control statement.
- Use TSO CALL command.
- Use another program which uses the CALL, LINK, or ATTACH macro.

Most JCL examples shown in this publication specify parameters used in locating uncataloged data sets. With cataloged data sets, the UNIT and VOL=SER parameters are not necessary.

Related reading:

- For information about allocating SMS-managed data sets, see [z/OS DFSMS Using Data Sets](#).
- For information about coding JCL statements, see [z/OS MVS JCL Reference](#).

Sharing data sets

Except for VSAM data sets or PDSEs, a data set cannot be updated by more than one job or user at a time without the risk of damaging the data set. Some data sets, particularly system data sets (identified by "SYS1"), are always in use. In order to safely update shared data sets, all but one user must stop updating the data set. After the update is finished, all users will have to re-access the data set. Re-accessing the data set is a function of the program using the data set and may involve closing and reopening the data set, or even freeing and reallocating the data set. Not all programs may be capable of doing this, so it is not always possible to safely update a shared data set. Unfortunately, this serialization mechanism does distinguish between data sets with the same name on different volumes.

You can use the DISP parameter on the DD statement, or the TSO ALLOCATE command, or the equivalent text unit for dynamic allocation (SVC 99) to put a lock on a data set so that you can update the data set. Specify DISP=OLD or DISP=MOD whenever you update a data set other than a PDSE.

If you code DISP=SHR in your JCL or the equivalent on the TSO ALLOCATE command or dynamic allocation, realize that the data set you are updating may be simultaneously updated by another user, resulting in an unusable data set, unless it is a PDSE. Another exception is a PDS. If a second program within the GRSplex tries to open the shared PDS for output (not the update in place option), that program gets a 213 ABEND. This detection of a violation of sharing protocol works even with two data sets with the same name on different volumes or two data sets with the same name and volume serial but on different systems. For these two cases the system does not issue ABEND 213.

This problem has been addressed by some components such as program management (binder, linkage editor), MVS allocation (JCL, SVC 99) and ISPF/PDF. Each component provides its own separate interlock and none of them recognize all the other interlocks. Therefore there is no totally safe way to update a data set allocation with DISP=SHR.

Partitioned data sets further complicate sharing because they have a directory and individual members. These sub-parts are generally protected inside the system ISPF/PDF does provide good protection against changes to members and the directory made by other ISPF/PDF user).

PDSE (partitioned data sets extended) are designed to avoid sharing problems. Consider using them in place of partitioned data sets.

When a volume is shared between unlike operating systems (such as between an MVS system and a VM system with shared DASD), DD statements or TSO ALLOC commands may not be able to stop a volume from being simultaneously updated from the two different systems.

Related reading: For more information about sharing data sets see [z/OS DFSMS Using Data Sets](#).

Utility control statements

Utility control statements are used to identify a particular function to be performed by a utility program and, when required, to identify specific volumes or data sets to be processed. The utility control statements that a particular utility uses are discussed in the topic for that utility.

Utility control statements are usually included in the input stream. However, they may also be placed in a sequential data set, in a member of a partitioned data set or PDSE, or in a z/OS UNIX System Services (z/OS UNIX) file such as a HFS file. In any case, the data set must have fixed or fixed blocked records with a logical record length of 80. For a z/OS UNIX file, the records can have various lengths; code FILEDATA=TEXT on the DD statement.

Exception: Some utilities allow exceptions to these rules.

The control statements for the utility programs have the following standard format:

label operation operand comments

The *label* symbolically identifies the control statement and, with the exception of system utility program IEHINITT, can be omitted. When included, a name must begin in the first position of the statement and must be followed by one or more blanks. The label can contain from 1 to 8 alphanumeric characters. IEBUPDTE control statements are an exception to this rule. They begin with "/" in positions 1 and 2, with an optional label beginning in position 3.

The *operation* identifies the type of control statement. It must be preceded and followed by one or more blanks.

The *operand* is made up of one or more keyword parameters, separated by commas. The operand field must be preceded and followed by one or more blanks. Commas, parentheses, and blanks can be used only as delimiting characters.

Comments can be written in a utility statement, but they must be separated from the last parameter of the operand field by one or more blanks.

Continuing utility control statements

Utility control statements are contained in columns 1 through 71. A statement that exceeds 71 characters must be continued on one or more additional records. A nonblank character must be placed in column 72 to indicate continuation.

Exception: Some utilities allow exceptions to this rule. In those cases, a utility statement can be interrupted either in column 71 or after any comma.

The continued portion of the utility control statement must begin in column 16 of the following record.

Exception: The IEBTPCH and the IEBGENER utility programs permit certain exceptions to these requirements.

Related reading: For more information, see:

- For information about the IEBTPCH utility, see [Chapter 11, “IEBTPCH \(Print-Punch\) Program,” on page 203](#),
- For information about the IEBGENER utility, see [Chapter 7, “IEBGENER \(Sequential Copy/Generate Data Set\) Program,” on page 127](#).

Restrictions

- Utility control statements do not support temporary data set names that begin with an ampersand. You can code the complete name generated for the data set by the system (for example, DSN=SYS95296.T000051.RP001.JOBTEMP.TEMPMOD). For utilities where you identify data sets only on DD statements or the dynamic allocation equivalent, you can use a temporary data set name that begins with an ampersand.
- The utility programs described in this publication do not normally support VSAM data sets. For certain exceptions, refer to the various program descriptions.
- You identify ASCII tape data sets with LABEL=(,AL) or OPTCD=Q.

The utilities that read or write binary (non-text) data on tape do not support ASCII tapes. This is because ASCII tapes require all data to be text. The utilities that do not support ASCII tapes include IEBCOPY, IEHMOVE, and IFHSTATR.

Related reading: For more information, see [z/OS DFSMS Using Data Sets](#).

Special referencing aids

To help you locate the correct utility program for your needs and locate the correct example of the program for reference, two special referencing aids are included in this publication.

To locate the correct utility program, refer to [Table 1 on page 1](#) under [“Guide to utility program functions” on page 1](#).

To locate the correct example, use the figure (called an "example directory") that precedes each program's examples. [Table 4 on page 9](#) shows a portion of the example directory for IEBCOPY. The figure shows that IEBCOPY Example 1 is an example of copying a partitioned data set from one disk volume to another and that IEBCOPY Example 2 is an example of copying from three input partitioned data sets to an existing output partitioned data set.

Table 4. Example directory

Operation	Device	Comments	Example
COPY	Disk	Full Copy. The input and output data sets are partitioned.	1
COPY	Disk	Multiple input partitioned data sets. Fixed-blocked and fixed-record formats.	2

Chapter 2. GDKUTIL (Cloud Object Utility) Program

GDKUTIL is a utility that is used to copy objects in cloud object storage to z/OS. It can also be used to copy z/OS data sets or UNIX files to cloud object storage.

Cloud object storage providers supported by DFSMSdftp cloud data access can be targeted by the GDKUTIL program. This list is present in the *MVS Programming: Callable Services for High Level Languages*. The utility allows upload and download data, with optional EBCDIC to UTF-8 conversion on the upload and UTF-8 to EBCDIC conversion on the download.

Environment requirements: See the **Cloud Data Access Configuration** section in the z/OS MVS Programming: Callable Services for High Level Languages for details regarding the environment required for the RACF user invoking the GDKUTIL program.

Saving the Cloud Credentials to be Used: Refer to the **Cloud Data Access Cloud Credential storage** section in the z/OS MVS Programming: Callable Services for High Level Languages for details regarding storing the Cloud Credentials to be used when communicating with the Cloud Object Storage.

Control

GDKUTIL is controlled by job control and utility control statements. The job control statements define the local z/OS UNIX file or data set, as well as the Cloud Object name to be used in the processing. The utility control statements are used to control the functions of the program.

Job Control

Table 5 on page 11 shows the job control statements for GDKUTIL.

Table 5. Job control statements for GDKUTIL

Statement	Use
JOB	Starts the job.
EXEC	Specifies the program name (PGM=GDKUTIL) , or if the job control statements reside in a procedure library, the procedure name.
BUCKET	Defines a sequential data set or UNIX file that contains the Cloud bucket name that is involved in the operation. It can be in-stream data (DD *), or point to a sequential data set or UNIX file. (Required for multi-object operations).
SYSPRINT DD	Defines a sequential data set or UNIX file for Utility output messages. The data set or UNIX file can be written on a system output device, a tape volume, or a DASD volume.
SYSIN DD	Defines the control data set, which contains the utility control statements. The data set normally resides in the input stream; however, it can be defined as a sequential data set, a member of a partitioned data set or PDSE, or a UNIX file. Any text in the last 8 characters of each record is ignored in case sequence numbers are used.

Table 5. Job control statements for GDKUTIL (continued)

Statement	Use
SYSOUT DD	Defines a sequential data set or UNIX file for DFSMSdfp CDA logging and error messages. The data set or UNIX file can be written on a system output device, a tape volume, or a DASD volume.
OBJNAME DD	Defines a sequential data set or UNIX file that contains the Cloud Object name to be used in the operation. It can be in-stream data (DD *), or point to a sequential data set or UNIX file. (Required)
LOCAL	Defines a data set or UNIX file that will be used in the operation. (Optional)
LOCNAME DD	Defines a sequential data set or UNIX file that contains the name of a z/OS data set or UNIX file to be used in the operation. It can be in-stream data (DD *), or point to a sequential data set or UNIX file. (Required if LOCAL does not exist.)
CREDSNAM DD	Defines the name of a UNIX file or z/OS data set that contains the credentials JSON to be used in the CREDENTIALS(ADD) command. It can be in-stream data (DD*) that identifies a z/OS UNIX file absolute path, or z/OS data set name. Or it can identify a z/OS UNIX PATH or Data Set that contains the z/OS UNIX file absolute path or z/OS Data Set name.
OBJNAMEX DD	Defines a sequential data set or UNIX file that contains the Cloud Object name to be used in the operation. It can be in-stream data (DD *), or point to a sequential data set or UNIX file. (Required if OBJNAME DD is not specified)

BUCKET Statement

The BUCKET statement describes the cloud bucket for a multi object operation. A multi object operation may be:

- An operation where multiple z/OS UNIX files are uploaded to a bucket.
- An operation where multiple cloud objects are downloaded to a z/OS UNIX directory.
- An operation where multiple cloud objects are deleted from a bucket.

A bucket name is define to be a name that starts with a forward slash character, and ends with a forward slash character with no other forward slash characters in between. If the UPLOAD command is used where the LOCNAME describes a UNIX directory, this bucket name is where the objects will be placed. The object names may be prefixed with the value of the PREFIX keyword as well if specified. If the DOWNLOAD command is used, the bucket name is the bucket where object names are listed. The PREFIX keyword may also adjust the object names that are considered for processing.

The first non-blank record in a data set, member, or in-stream data, or non-blank line in a UNIX file is read from the DD and the content is used as the bucket name. The bucket name must start with a forward slash, followed by the bucket or container name ending in a forward slash. The cloud bucket or container must already exist. Leading blanks and trailing blank characters in the name are trimmed.

Most characters are acceptable. However, the following characters are unacceptable and may cause the request to fail:

- Backslash ("\\")
- Left curly brace ("{"
- Non-printable ASCII characters (128-255 decimal characters)
- Caret ("^")
- Right curly brace ("}")
- Percent character ("%")
- Grave accent / back tick ("`")
- Right square bracket ("]")
- Quotation marks
- 'Greater Than' symbol (">")
- Left square bracket ("["
- Tilde ("~")
- 'Less Than' symbol ("<")
- 'Pound' character ("#")
- Vertical bar / pipe ("|")

CREDSNAM Statement

The CREDSNAM statement identifies the location of JSON that contains the credentials to be used by the CREDENTIALS(ADD) command. The location may be a z/OS UNIX absolute path, or a Data Set. The DD will be opened and all records/lines are read, concatenating them together after trimming leading and trailing whitespace to create the z/OS data set or UNIX file to open.

The JSON document may contain the following keys with your credential values as needed:

- accessKey – The value used in authentication for:
 - AWS4 authentication type providers: S3 HMAC Access Key ID
 - Azure authentication type providers: Storage Account name
- secretAccessKey – The value used in authentication for:
 - AWS4 authentication type providers: S3 HMAC Secret Access Key
 - Azure authentication type providers: Key
- tenant – The tenant value used in the authentication.
- userid – The userid value used for authentication.
- password – The password value used for authentication.
- accessURL – The Access URL used during authentication.
- credentials-file – The absolute path to the UNIX file that contains the JSON credentials obtained from the cloud provider

Note: this is only used with OAUTH_2 authentication models. This value may override a specified accessURL. Both may not be specified in a JSON credentials document.

Note:
The first occurrence of each key is the value used. Subsequent duplicate keys are ignored.

Example JSON credentials found in /u/user1/tempcreds.json

```
{
  "accessKey": "myIBN_Cloud_AccessKey",
  "secretAccessKey": "myIBM_Cloud_SecretAccessKey"
}
```

Example JSON credentials found in /u/user/temp_credentials.json

```
{
```

```
  "credentials-file": "/u/ibmuser/GCPcreds/credentials.json"
}
```

EXEC Statement

The EXEC statement specifies PGM=GDKUTIL. Any PARM parameter is ignored.

SYSOUT Statement

If the SYSOUT statement is omitted, one will be dynamically created if logging is requested.

SYSPRINT Statement

If the SYSPRINT statement is omitted, one will be dynamically created for any messages written.

OBJNAME Statement

The OBJNAME statement describes the location that holds the object name in Cloud Storage that will be used in the utility operation. The first non-blank record in a data set, member, or in-stream data, or non-blank line in a UNIX file is read and the content is used as the object name. The object name must start with a forward slash, followed by the bucket or container name. The bucket or container must already exist. The rest of the object name follows, starting with a forward slash. Additional forward slashes may be used to create a pseudo-directory structure. Refer to the cloud provider documentation for additional details. Leading blanks and trailing blank characters in the name are trimmed.

Most characters are acceptable. However, the following characters are unacceptable and may cause the request to fail:

- Backslash ("\")
- Left curly brace ("{"
- Non-printable ASCII characters (128-255 decimal characters)
- Caret ("^")
- Right curly brace ("}")
- Percent character ("%")
- Grave accent / back tick ("`")
- Right square bracket ("]")
- Quotation marks
- 'Greater Than' symbol (">")
- Left square bracket ("["
- Tilde ("~")
- 'Less Than' symbol("<")
- 'Pound' character("#")
- Vertical bar / pipe ("|")

The OBJNAME DD may contain simply a bucket name. A bucket name is defined to be a name that starts with a forward slash character, and ends with a forward slash character with no other forward slash characters in between. The LIST command will return a list of objects found within a bucket in this case. If the OBJNAME is simply the forward-slash character, and the LIST command is used, then a list of the accessible buckets is displayed. The PREFIX keyword may also adjust the object or bucket names that are retrieved.

OBJNAMEX Statement

The OBJNAMEX statement describes the location that holds the object name in Cloud Storage that will be used in the utility operation. All records in the data set, member or in-stream data, or lines in a UNIX file is

read and the content is used as the object name. Leading and trailing whitespace is ignored when building the object name. Note that sequence numbers will be considered as part of the record and included in the text.

The object name must start with a forward slash, followed by the bucket or container name. The bucket or container must already exist. The rest of the object name follows, starting with a forward slash. Additional forward slashes may be used to create a pseudo-directory structure. Refer to the cloud provider documentation for additional details regarding object names.

Most characters are acceptable. However, the following characters are unacceptable and may cause the request to fail:

- Backslash ("\\")
- Left curly brace ("{"
- Non-printable ASCII characters (128-255 decimal characters)
- Caret ("^")
- Right curly brace ("}")
- Percent character ("%")
- Grave accent / back tick ("`")
- Right square bracket ("]")
- Quotation marks
- 'Greater Than' symbol (">")
- Left square bracket ("["
- Tilde ("~")
- 'Less Than' symbol("<")
- 'Pound' character("#")
- Vertical bar / pipe ("|")

The OBJNAMEX DD may contain simply a bucket name. A bucket name is defined to be a name that starts with a forward slash character, and ends with a forward slash character with no other forward slash characters in between. The LIST command will return a list of objects found within a bucket in this case.

If the OBJNAMEX is simply the forward-slash character, and the LIST command is used, then a list of the accessible buckets is displayed. The PREFIX keyword may also adjust the object or bucket names that are retrieved.

LOCAL Statement

The LOCAL statement describes a sequential z/OS data set or member of a partitioned data set or PDSE, or a z/OS UNIX file. For an UPLOAD command, the data in the z/OS UNIX file or data set will be read and placed into the cloud object. The z/OS data set or member must have a record format (RECFM) of F, FB, V, or VB, and logical record length greater than zero (LRECL). A RECFM=U data set may also be used. For a DOWNLOAD command, the data will be appended if DISP=MOD, or PATHOPTS=(OAPPEND) is used. If DISP=OLD, or PATHOPTS=(OTRUNC), the existing data will be overwritten.

Note: While a PDSE or PDS data set name may be specified for an UPLOAD command, this only opens the directory for read, and none of the members are processed.

Note: When specifying a RECFM=U data set for an UPLOAD, the CONVERT keyword should not be specified. When the CONVERT keyword is not specified, the raw data from each block is sent, and any record or block boundaries are lost. Likewise, during a DOWNLOAD command targeting a RECFM=U data set, the byte stream is not examined to guess at the record boundaries.

LOCNAME Statement

The LOCNAME statement describes the location that holds the z/OS sequential data set, PDS or PDS/E member, or version of a Generation Data Group (GDG) that will be used in the utility operation. The first

non-blank record in a data set, member, or in-stream data, or non-blank line in a UNIX file is read and the content is used as the local name.

If the name does not start with a forward-slash, it is assumed to be a fully qualified Data Set name. UNIX file names must be an absolute path.

Additionally, the data set name notation for `fopen()` can be used. e.g.

```
// 'FULLY.QUALIFD.NAME'
// 'PDSE.DATASET.NAME(MEMBER1)'
// 'GENER.ATION.DATASET(+1)'
```

Details can be found in the z/OS XL C/C++ Programming Guide, Input and Output chapter, performing OS I/O operations -> Opening Files → Using `fopen()` or `freopen()`.

Using a data set name in *z/OS XL C/C++ Programming Guide*

Note: If using a generation data set relative name and you want to create a new (+1) generation, this must be done by a previous step in the JCL JOB. The utility will not create a new generation from the name in the LOCNAME statement.

Note: Data Set attribute considerations

When specifying a z/OS Data Set for LOCAL or LOCNAME for a download command without the CONVERT keyword, the Record Format should not be Variable. (RECFM=V or RECFM=VB) This is because in binary mode, there are no indicators in the data where a record ends, so the data is placed up to the maximum logical record length. If a Data Set with variable records was uploaded to a Cloud Object in binary mode, the indicators of where a record begins and ends are lost. Therefore, a download operation of that same data to a Variable record data set cannot tell when to start a new record in the data set. During a DOWNLOAD command, if the data set name specified on the LOCNAME DD does not exist, a data set will be created automatically using the defaults described in the `fopen()` defaults section of the "Input and Output" chapter of the x/OS XL C/C++ Programming Guide. These defaults may not meet your expectations for containing the data.

The LOCNAME DD may contain in the absolute path of a UNIX directory. When specified with the UPLOAD command, this indicates that all non-hidden regular files within the directory tree may be selected for upload to the cloud provider. Symbolic links are followed to examine the target of the link. When specified with the DOWNLOAD command, this indicates that all objects downloaded should be placed in that directory. If any pseudo-directory characters (forward slash /) are found in the object name, then a subdirectory with that name will be attempted to be created.

Note: z/OS UNIX file names may contain unprintable characters that will not be url-encoded, resulting in a failure to upload that file. Not all valid z/OS UNIX file names can be used as cloud object names. Some characters may be deemed unacceptable by the cloud provider, and may result in a failure even if url-encoded as specified in the `urlEncodeChars` key:value pair in the cloud provider file.

SYSIN Statement

GDKUTIL is controlled by the utility control statements shown in the table below.

Statement	Use
Commands	
Upload	Requests the data in LOCAL or the z/OS UNIX file / Data Set named in the LOCNAME DD be sent to the Cloud object Storage provider referenced on the PROVIDER keyword. The data will be stored in the object named by the OBJNAME DD. If the Cloud already has an object with that name, it will be overwritten without warning. The bucket must already exist. You may request the data be processed differently by specifying the CONVERT keyword. Minimum length UP.

Statement	Use
Commands	
Download	<p>Requests the data in the object named by the OBJNAME DD for the provider reference on the PROVIDER keyword be written to the z/OS UNIX file, or Data Set/member referenced on the LOCAL or LOCNAME DD. If the LOCAL DD is used, and DISP=OLD is used, the data will be overwritten. If DISP=MOD is used, the data will be appended to the end. The request will fail if DISP=SHR is used.</p> <p>If LOCNAME is used and the z/OS UNIX file or data set exists, its data will be overwritten. Likewise, if an error occurs before data is written to the data set, it may be left empty, with the previous data no longer accessible.</p> <p>If the z/OS UNIX file or data set does not exist and the object does not have the , it will be created using the defaults described in the fopen() defaults section of the "Input and Output" chapter of the z/OS XL C/C++ Programming Guide. These defaults may not meet your expectations for containing the data. Minimum length DOWN.</p> <p>Note: If not transferring in text mode, and the target data set is RECFM=V or RECFM=VB, all records will be the same length, except possibly the last.</p>
Delete	Requests the object named by the OBJNAME DD for the provider referenced on the PROVIDER keyword be removed from the Cloud Object Storage bucket.
List	<p>Requests a list of appropriate information from the cloud server according to the specified object name specified on the OBJNAME DD:</p> <ul style="list-style-type: none"> • / - Requests a list of the accessible buckets from the cloud object server. The LISTBUCKETS operation must exist in the provider file. • /<bucket_name>/ - Requests a list of the objects in the named bucket. The LISTOBJECT operation must exist in the provider file. • /<bucket_name>/<object_name> - Requests a listing of the metadata and other information from the time of the request. The HEADOBJECT operation must exist in the provider file.
OPERATION(<oper_name>)	Requests the operation named <oper_name> to be performed. <oper_name> is an operation found in the supportedOperations array in the provider file. minimum length: OPER(<oper_name>)

Statement	Use
Commands	
CREDENTIALS(ADD DELETE LIST)	<p>Requests Credential management functions:</p> <p>ADD – Add the cloud credentials found in the JSON document found in the z/OS UNIX file or data set named by the CREDSNAM DD statement. The credentials will be saved for the named PROVIDER name. The optional BUCKET DD statement may be used to describe the bucket name the credentials apply to. If the BUCKET DD is not provided, the credentials will be saved for the generic / bucket, meaning all buckets. If credentials already exist for the provider, they will be deleted before the new credentials are added.</p> <p>DELETE – Delete the saved credentials for the named PROVIDER and bucket name. If the optional BUCKET DD wasn't specified to indicate the specific bucket name for the credentials, the generic / bucket is defaulted to.</p> <p>LIST – Display a list of credentials (by bucket name) for the PROVIDER specified. A provider name of asterisk (*) may be specified to indicate credentials for all providers should be displayed.</p>
Keywords	
PROVIDER(NAME)	<p>Required. Indicates the name of a ~/gdk/providers/<provider_name>.json file that describes the Cloud Object Storage provider that will be used. The specified name is case sensitive and must match the case of the file in the providers directory. See the Provider file in <i>z/OS MVS Programming: Callable Services for High-Level Languages</i> manual for more details about the provider file. Alias: PROV(<provider_name>).</p>
CONVERT	<p>Optional: Indicates that the data should be converted from EBCDIC (IBM-1047) to ASCII (ISO8859-1) on UPLOAD, or from ASCII (ISO8869-1) to EBCDIC (IBM-1047) on DOWNLOAD. The data is read or written to the z/OS data set or UNIX file in text mode. In text mode during UPLOAD, when reading from a z/OS data set, each record has a newline character appended at the end of the record data. In text mode during DOWNLOAD, the newline characters are used as record delimiters, and are stripped before writing that record to the output data set. If not specified, the data is read/written in binary mode. No bytes are added or removed from the data during I/O. Minimum text CONV.</p>
CONVERT(<localCCSID>{,<remoteCCSID>})	<p>Optional: Indicates that DFSMSdfp CDA should convert the data the local CCSID (code page) to the remote CCSID (code page) on UPLOAD, or from the remote code page to the local code page on DOWNLOAD. The remote CCSID is optional, and will default to ASCII (ISO8859-1) if not specified. Minimum text CONV(<localized>).</p>
DELSRC	<p>Optional. Indicates that the source should be deleted after a successful processing. If specified on an UPLOAD command, the utility attempts to remove the z/OS UNIX file or data set. If specified on a DOWNLOAD command, a request to delete the cloud object is sent. Only valid for UPLOAD and DOWNLOAD commands.</p>

Statement	Use
Commands	
LOG(level)	<p>Optional: Indicates an override of the logging level. The default level is ERROR. <level> may be one of:</p> <ul style="list-style-type: none"> • NONE - No logging done, including errors. • ERROR (default) - Only Errors are logged. • WARNING - Only Warnings and above are logged. • NOTICE - Only Notification types and above are logged. • INFO - Only Information types and above are logged. • DEBUG - All information is logged.
WEBTOOLKITLOG	<p>Optional. Request logging from the z/OS Client Web Enablement Toolkit. This output is written to the SYSPRINT DD. This logging is useful in debugging errors occurring during communication with the Cloud Object Storage server. See the z/OS MVS Programming: Callable Services for High-Level Languages, z/OS client web enablement toolkit chapter for information about the HWTH_OPT_VERBOSE option. Abbreviation: WTKLOG</p>
WEBTOOLKITLOGU	<p>Optional: Requests the Unredacted logging output from the z/OS Client Web Enablement Toolkit. Abbreviation: WTKLOGU</p>
PREFIX(<list_prefix>)	<p>Requests that only object names that begin with <list_prefix> are returned for the LIST command. <list_prefix> is a string of characters. The provider file must include the URL_PARM requestParameter definition for "prefix". minimum length: PRE(<list_prefix>)</p> <p>Requests that only bucket names that begin with <list_prefix> are returned for the LIST command when the object name is only /.</p> <p>When specifies with the UPLOAD command and the LOCNAME is for a directory, each object name created in the cloud provider will contain the requested prefix.</p> <p>When specified with the DOWNLOAD or DELETE command the list of objects considered for processing is modified with the prefix.</p>
DELIM(<single_char>)	<p>Requests that only the portion of the object name that is common up to the <single_char> is returned. This is the most commonly used with the forward-slash character (/) to utilize a pseudo directory structure to object names. The provider file must include the URL_Parm requestParameter for "delimiter"</p>
REGEX(<reg_expr>)	<p>Indicates that the command should process multiple things, and selection of the object name or local z/OS UNIX file should be done according to whether it matches the <reg_expr>. Only applies for UPLOAD where LOCNAME indicates a UNIX directory, DOWNLOAD or DELETE where OBJNAME indicates a bucket name.</p>
TEST	<p>Indicates that processing should not be performed for a multiple object command. Only selection is performed. This can be used to determine what would be processed in a multiple object command. For example, when the UPLOAD command is specified with TEST and the LOCNAME indicates a z/OS UNIX directory, messages will be issued indicating the source z/OS UNIX file name, and target cloud object name after applying the prefix if PREFIX was specified, and applying the REGEX regular expression to select a file for upload processing.</p>

Statement	Use
Commands	
METADATA	<p>Optional. Specifies a DD name that contains descriptions of the metadata to be saved with the cloud object during an UPLOAD command. The contents of the metadata DD can specify key:value pairs, separated by commas, where the key is folded to lowercase and used in a provider appropriate HTTP header. The key cannot contain blank or space characters.</p> <p>The value specified must be less than 2048 characters in length and may be:</p> <ul style="list-style-type: none"> • A constant value, which will be added as-is. Contents are not examined for sensitive data. • A System symbol specified with the ampersand character. i.e. &SYSNAME. • A JCL symbol specified with the ampersand character (&). • A CDA Symbol specified with the ampersand character (&). See table below for list of supported CDA symbols. <p>If a symbol does not resolve to a value, it is ignored and not included in the metadata attached to the cloud object.</p> <p>The WRITEOBJECT, or WRITELARGEOBJECT operation in the provider file must have a mechanism:METAHEADER object in the requestParameters that describes the prefix specific to that Cloud Storage provider. The sample provider files include this. If the provider file does not have the METAHEADER object, the request will fail.</p>
ZUSERID(<user>)	<p>Used with the CREDENTIALS command to indicate that the credentials management request should be performed for the named user. For ADD and DELETE, the user ID associated with the JOB must have ALTER access to the <user>'s GDK.<user> key label in the CSFKEYS class. Additionally, the JOB userid must have write access to the <user>'s home directory. For LIST requests, the user ID associated with the JOB must have read access to the <user>'s ~/gdk/gdkkeyf.json file.</p>
COMPRESSION(zEDC gzip)	<p>When specified on the UPLOAD command, this indicates that DFSMSdfp CDA should perform the requested compression algorithm on the data sent to the cloud object. If a METAHEADER description exists in the requestParameters array for the WRITEOBJECT and WRITELARGEOBJECT operations, then a metadata tag for “zos-compression” with the value of the requested algorithm.</p> <ul style="list-style-type: none"> • zEDC – Compress using the z Enterprise Data Compression algorithm. • gzip – Compress using the gzip compression algorithm. <p>May be shortened to COMPRESS.</p>

Statement	Use
Commands	
COMPLEVEL(MAX SPEED DEFAULT {1-9})	<ul style="list-style-type: none"> • MAX indicates that CDA should try to get the most compression for data being sent. It can result in extra CPU usage as the best compression result is attempted. • SPEED indicates that CDA should request the compression performed be done as quickly as possible, even if the compression ration is not that good. • DEFAULT indicates that the default compression level should be used. It is a mid-point between SPEED and MAX. <p>1-9 – A number that indicates the specific compression level to be given to the zlib interface.</p>
DECOMPRESS(zEDC gzip zlib NONE)	<p>When specified on the DOWNLOAD command, this indicates that DFSMSdfp CDA should decompress the cloud object data using the requested algorithm, regardless of the existence of the “zos-compression” metadata tag.</p> <ul style="list-style-type: none"> • zEDC – Decompress using the zEDC algorithm. • gzip – Decompress using the gzip algorithm. • zlib – Use the zlib headers imbedded in the object data to determine the algorithm to use. • NONE – Do not perform any decompression, regardless of the existence of the zos-compression metadata tag on the object.
FORMAT(RECORD <u>NONE</u>)	<p>When specified on the UPLOAD command for a z/OS data set, FORMAT(RECORD) indicates that DFSMSdfp CDA should imbed record prefixes into the user data so that the cloud object may be downloaded to z/OS while re-creating the records in the data set. Metadata will be associated with the object to identify the object as containing the embedded record prefix data. FORMAT(NONE) is the default.</p> <p>The DOWNLOAD command does not require specification of this keyword, since DFSMSdfp CDA will recognize the zos-filedata:record metadata tag, and process the data accordingly. However, if you know that the data within an object contains the imbedded record prefixes, but the object does not have the metadata tag, you may specify FORMAT(RECORD) to force processing of the imbedded record prefix data. Likewise, if you want to override automatic processing of the imbedded record prefix data, you may specify FORMAT(NONE).</p> <p>When creating a data set, the metadata tags must contain at least zos-dsorg with zos-recfm and zos-lrecl when zos-dsorg is PS, PS_EXT, or PS_LARGE. When zos-dsorg is VSAMKSDS or VSAMESDS, the zos-vs-recordsize and zos-vs-keys for VSAMKSDS.</p> <p>Note: z/OS UNIX files are not supported for UPLOAD with FORMAT(RECORD)</p>
STORCLAS(<storage_class>)	<p>This keyword may be specified on the DOWNLOAD command, and allows you to override the SMS storage class name used when creating the data set when it does not exist. This keyword may be specified with nothing between the parentheses to indicate that no name should be used when creating the data set.</p>

Statement	Use
Commands	
MGMTCLAS(<management_class>)	The MGMTCLAS keyword is recognized with the DOWNLOAD command, and allows an override of the SMS management class name used when creating the data set when it does not exist. This keyword may be specified with nothing between the parentheses to indicate that no name should be used when creating the data set
DATACLAS(<data_class>)	The DATACLAS keyword is recognized with the DOWNLOAD command, and allows an override of the data class name used when creating the data set when it does not exist. This keyword may be specified with nothing between the parentheses to indicate that no name should be used when creating the data set.
VOLUMES(<volume_list>)	The VOLUMES keyword is recognized with the DOWNLOAD command and allows an override of the volume serials used when creating the local data set. If more than one volume is passed, the volume serials should be separated by commas. If the local data set is to be a sequential data set, only the first volume serial is used. If the local data set is to be a VSAM data set, all volume serials will be passed when creating the data set.
UNIT(<unit_name>)	The UNIT keyword is recognized with the DOWNLOAD command, and allows an override to the unit name passed on the DYNALLOC request when creating a non-VSAM data set. The string can be simply the empty string, which indicates that no UNIT name should be passed. When not specified, the UNIT name used is SYSALLDA.
LOCSIZE(<bytes>)	The LOCSIZE keyword may be specified to override the size of the data set created. The size specified is in number of bytes of data to be written to the data set.
LISTOUTDD(<ddname>) LISTDD(<ddname>)	The output from a LIST command may be directed to a different DD statement using the LISTOUTDD keyword. Specify the name of a DD statement for the step where the object list information should be written to. The record length of the data should allow for the full records to be written. (Maximum object name length is 1024 characters.) If the named DD does not exist, SYSPRINT will be used for the LIST output.
LISTDATEFMT('<format>')	The format of the date fields in the LIST output may be tailored to your needs using the LISTDATEFMT keyword. Use syntax conforming to the <code>strftime()</code> conversion specifiers to indicate how the object date should be formatted. For more information about <code>strftime</code> , see the description in strftime — Convert to formatted time in z/OS C/C++ Runtime Library Reference All text between the single quote characters is considered the format text.

CDA Symbol	Meaning
GDK_BLKSIZE	Block size value from the z/OS data set
GDK_LRECL	LRECL value for the z/OS data set

CDA Symbol	Meaning
GDK_RECFCM	Record Format value for the z/OS data set <ul style="list-style-type: none"> • F – Fixed records • FB – Fixed Blocked • FBA – Fixed Blocked ASA print-control characters • FBS – Fixed Blocked Standard • FS – Fixed Standard • V – Variable • VB – Variable Blocked • VBA – Variable Blocked with ASA print-control Characters • VBS – Variable Blocked Spanned records • U - Unknown
GDK_DSORG	Data Set organization <ul style="list-style-type: none"> • PS – Physical Sequential • PDS – Partitioned Data set • LIBRARY – Partitioned Data Set Extended
GDK_EXPDATE	Expiration Date of the cataloged z/OS data set
GDK_CREDATE	Creation Date of the cataloged z/OS data set or UNIX file
GDK_REFDATE	Last Reference Date of the cataloged z/OS data set or last backup date of a z/OS UNIX file.
GDK_ISDATASET	True if z/OS data set. False if z/OS UNIX file
GDK_DATACLAS	Data Class associated with data set
GDK_MGMTCLAS	SMS Management Class associated with data set
GDK_STORCLAS	SMS Storage Class associated with data set
GDK_ONAME	Original Data Set or File name
GDK_JOBNAME	JCL Job name that put this object into the cloud
GDK_STEPNAME	JCL Step name that put this object into the cloud
GDK_CTIME	Time the z/OS UNIX file metadata changed
GDK_MTIME	Time the z/OS UNIX file data changed
GDK_SECONDARY	Secondary allocation amount for the z/OS data set in the format: <nnn>-<unit>, where <i>nnn</i> is the number or amount, and unit is the allocation unit. e.g. CYL, TRK, BLK, REC, MB, KB, B

CDA Symbol	Meaning
GDK_DSORGE	<ul style="list-style-type: none"> • PDSE for PDS/E dataset. • PDS for PDS dataset. • PS_LARGE when is a large format data set. (DS1LARGE). • PS_EXT when is extended format. (DS1STRP). • PS when data set is basic format sequential. • VSAMESDS when data set is Entry Sequenced VSAM. • VSAMKSDS when data set is Key Sequenced VSAM.
GDK_VSACCOUNT	32 bytes of accounting information and user data for a VSAM data set. Resolves to the value originally specified on the ACCOUNT() parameter when the VSAM data set was defined.
GDK_VSBUFFSPACE	The BUFFERSPACE value for the VSAM data set.
GDK_VSBWOTYPE	The backup-while-open (BWO) value for the VSAM data set. Will resolve to TYPECICS, TYPEIMS, or NO.
GDK_VSDCISIZE	The size of the control interval for the VSAM data set data component.
GDK_VSICISIZE	The size of the control interval for the VSAM data set index component.
GDK_EATTR	The Extended Attribute value for the VSAM data set. Resolved values are: OPT or NO
GDK_VSERASE	Indicator whether the cluster's components are to be erased when its entry in the catalog are deleted. Resolved values are: ERASE or NOERASE
GDK_VSFREESPACE	Value of the free space to be set aside after the initial load of the VSAM data set. Values are: cinnn-cannn
GDK_VSKEYLABEL	The key label name associated with the VSAM data set.
GDK_VSKEYS	Primary key field information for the VSAM data set. Resolved values are in the form of len_nnn-off_nnn, where len_nnn is the length of the key, and off_nnn is the zero based offset in the record that the primary key starts at.
GDK_VSLOG	Value from the LOG keyword for the VSAM data set. Resolved values are: NONE, UNDO, and ALL
GDK_VSLOGREPLICATE	Value for the VSAM data set eligibility for VSAM replication. Resolved values are: LOGREPLICATE, and NOLOGREPLICATE.
GDK_VSLOGSTREAMID	Name for the forward recovery log stream for the VSAM data set.

CDA Symbol	Meaning
GDK_VSOWNER	Resolves to the cluster's owner userid.
GDK_VSRECORDSIZE	The average and maximum lengths of the records in the data component. Values are in the format: avg_nnn-max_nnn, where avg_nnn is the average record size, and max_nnn is the maximum record size.
GDK_VSREUSE	Resolves to the indicator whether the VSAM cluster can be opened again and again as a reusable cluster. Values are: REUSE or NOREUSE
GDK_VSSHAREOPTIONS	The share options value for the VSAM data set. Values are in the format: reg_n-sys_n, where reg_n is the cross region option value, and sys_n is the cross system option value. i.e. 1-3.
GDK_VSSPANNED	Attribute of the VSAM data set indicating whether it can contain records that cross control interval boundaries. Resolved values are: SPANNED, or NON_SPANNED.
GDK_VSPREFORMAT	Attribute of the VSAM data set indicating whether the control areas of the VSAM data component should be pre-formatted during loading. Resolved values are: SPEED, or RECOVERY.
GDK_VOLSER	This will resolve to a hyphen separated list of volume serials for the current data set. e.g. VOL1-VOL002-VOLA1E.

Metadata key	Value
zos-lrecl	Set to the logical record length of the data set.
zos-recfm	Set to the record format of the data set.
zos-blksize	Set to the block size of the data set.
zos-dsorg	Set to the data set organization of the data set.
zos-dataclas	Set to the DATACLAS for the data set.
zos-mgmtclas	Set to the SMS Management Class for the data set.
zos-storclas	Set to the SMS Storage Class for the data set.
zos-secondary	Set to the secondary allocation amount in the format. <nnnn><type>where type is CYLS, TRKS, BLKS.
zos-vs-account	Set to the value of the ACCOUNT for the VSAM data set if it exists.
zos-vs-buffspace	Set to the value of the BUFFSPACE for the VSAM data set if not default.
zos-vs-bwotype	Set to the backup-while-open value for the VSAM data set.
zos-vs-dcsize	Set to the VSAM cluster Data Component CISIZE

Metadata key	Value
zos-vs-icisize	Set to the Indexed VSAM cluster Index Component CISIZE.
zos-eattr	Set to the Extended Attribute value for the VSAM data set.
zos-vs-erase	Set to the ERASE setting for the VSAM data set.
zos-vs-freespace	Set to the VSAM data set FREESPACE value.
zos-vs-keylabel	Set to the VSAM data set KEYLABEL name if it exists.
zos-vs-keys	Set to the Primary Key field information for the VSAM data set.
zos-vs-log	Set to the value of the LOG keyword for the VSAM data set.
zos-vs-logreplicate	Set to the value of the LOGREPLICATE keyword for the VSAM Data set.
zos-vs-logstreamid	Set to the LOGSTREAMID value for the VSAM data set if it exists.
zos-vs-owner	Set to the OWNER value for the VSAM data set if it exists.
zos-vs-recordsize	Set to the RECORDSIZE value for the VSAM data set.
zos-vs-spanned	Set to SPANNED when the VSAM data set was defined as having spanned records.
zos-volumes	Set to a hyphen separated list of volume serials for the data set.

GDKUTIL Examples

Example 1: Retrieve an object into a z/OS UNIX file

In this example a binary Cloud object is retrieved and stored into a z/OS UNIX file.

```
//DOWNCLD EXEC PGM=GDKUTIL,REGION=0M
//SYSPRINT DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//SYSIN DD *
  DOWNLOAD PROVIDER(IBM COS)
/*
//OBJNAME DD *
  /cloudbucket01/file1.jar
/*
//LOCNAME DD *
  /prod/input-files/cloud/file1.jar
/*
```

The file1.jar object in the cloudbucket01 bucket is specified as the object name on the in stream DD for OBJNAME. The local name specified using the LOCNAME in stream DD is in the local UNIX directory/prod/input-file/cloud/ with the filename being file1.jar. The SYSIN directs the action to be a Download of the object to the Local name from the Cloud provider specified in the IBM COS.json file in the gdk/providers/ directory.

Example 2: Send a z/OS Sequential data set to a Cloud Object

In this example, a sequential data set with EBCDIC text data is sent to a cloud object with conversion to UTF-8. Note that the /cloudbucket01/ bucket must have been created previously.

```
//UPCLOUD EXEC PGM=GDKUTIL,REGION=0M
//SYSPRINT DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//SYSIN DD *
```

```

UPLOAD PROVIDER(IBM COS) CONVERT
/*
//OBJNAME DD *
      /cloudbucket01/processing-report-20211018.txt
/*
//LOCAL DD DISP=SHR,DSN=REPORTS.CICSPROC.D211018

```

Example 3: Create a Bucket

In this example, the OPERATION command is used to create a new bucket in the cloud provider.

```

//CRBUCKET EXEC PGM=GDKUTIL,REGION=0M
//SYSPRINT DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//SYSIN DD *
      OPERATION(CREATEBUCKET) PROVIDER(IBM COS)
/*
//OBJNAME DD *
      /newbucket05/
/*

```

Example 4: List Objects in a cloud storage bucket with filtering

In this example, the GDKUTIL utility is used to list the objects within a specific bucket. The PREFIX keyword is used to filter the results to only those object names that begin with the prefix, "images/". The DELIM keyword is used to further filter those results to only the portion that has a forward slash next. Some applications like to use a forward slash character in the object name as a pseudo-directory character in order to group objects by 'directory' order.

```

//LISTDIR1 EXEC PGM=GDKUTIL,REGION=0M
//SYSPRINT DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//SYSIN DD *
      LIST PROVIDER(IBM COS)
      PREFIX(2023-jan/webapp/) DELIM(/)
/*
//OBJNAME DD *
      /appimages05/
/*

```

Example 5: Upload multiple files to cloud storage

In this example, the GDKUTIL utility is used to upload an entire directory tree of files to objects in cloud storage. The LOCNAME DD indicates that the /u/user01/images/ directory is the place to start looking for files to upload. The PREFIX keyword is used here to request that every cloud object begin with /cloudbucket01/user01/images/. Individual files found will have their path name relative to the LOCNAME value appended to /cloudbucket01/user01/images/. The REGEX keyword is used here to only select filename that end in ".png".

```

//UPMULTI EXEC PGM=GDKUTIL,REGION=0M
//SYSPRINT DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//SYSIN DD *
      UPLOAD PROVIDER(IBM COS)
      PREFIX(user01/images) REGEX(.*\.png)
/*
//BUCKET DD *
      /cloudbucket01/
/*
//LOCNAME DD *
      /u/user01/images/
/*

```

Example 6: Download multiple objects from cloud storage

In this example, the GDKUTIL utility is used to download multiple objects from the cloud provider described in the IBM COS.json provider file. The CONVERT keyword requests translation of each file from

UTF-8 to EBCDIC. The LOCNAME DD describes the target directory to place the download objects into. The BUCKET DD specifies the bucket name where the objects are to be found. The PREFIX keyword limits the selection of object names to only the ones that begin with webapp/logs/. The REGEX keyword further filters that list so that only objects with names ending in ".txt" are selected for download processing. Pseudo-directory characters (forward slash /) will cause the object to be placed in the z/OS UNIX directory with the same name.

```
//DOWNMULT EXEC PGM=GDKUTIL,REGION=0M
//SYSPRINT DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//SYSIN DD *
  DOWNLOAD PROVIDER(IBMCO) CONVERT
  PREFIX(webapp/logs) REGEX(.*)\.txt)
/*
//BUCKET DD *
  /cloudbucket01/
/*
//LOCNAME DD *
  /u/approc01/logs/
/*
```

Example 7: Delete multiple objects from cloud storage

In this example, the GDKUTIL utility is used to delete multiple objects from the cloud provider described in the IBMCO.json provider file. The BUCKET DD describes the bucket name contain the objects to be deleted. The PREFIX keyword limits the selection of object names to only the ones that begin with webapp/logs/. The REGEX keyword further filters that list so that only objects with names ending in ".log" are selected for deletion.

```
//DELMULT EXEC PGM=GDKUTIL,REGION=0M
//SYSPRINT DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//SYSIN DD *
  DELETE PROVIDER(IBMCO)
  PREFIX(webapp/logs) REGEX(.*)\.log)
/*
//BUCKET DD *
  /cloudbucket01/
/*
```

Example 8: Send a GDG version to a Cloud Object with Metadata tags

In this example, a sequential data set with EBCDIC data is sent to cloud object storage with conversion to UTF-8. Metadata tags are associated with the cloud object according to the name and symbols specified.

```
//UPCLOUD EXEC PGM=GDKUTIL,REGION=0M
//SYSPRINT DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//SYSIN DD *
  UPLOAD PROVIDER(IBMCO) CONVERT
  METADD(DSMETA)
/*
//OBJNAME DD *
  /cloudbucket01/previous-report.txt
/*
//LOCNAME DD *
  APPLICAT.SMS.REPORT(-2)
/*
//DSMETA DD *
  blksize:&GDK_BLKSIZE,lrecl:&GDK_LRECL,dsorg:&GDK_DSORG,
  sysname:&SYSNAME,jobname:&GDK_JOBNAME,stepname:&GDK_STEPNAME,
  recfm:&GDK_RECFM,expiredate:&GDK_EXPDATE,createdate:&GDK_CREDATE,
  dataset:&GDK_ISDATASET,dclass:&GDK_DATACLAS,mclass:&GDK_MGMTCLAS,
  sclass:&GDK_STORCLAS,origname:&GDK_ONAME,
  referenceDate:&GDK_REFDATE,sample:MySampleJCL
/*
```


Example 9: New default credentials for the CLOUD1 provider

The example below shows the default S3 credentials being added for the CLOUD1 provider. (Default means all buckets). The S3 credentials are found in a separate UNIX file with the absolute path: /u/user1/add_S3_creds.json. The content of the UNIX credentials file is a JSON document as displayed below.

```
//EXEC      PGM=GDKUTIL,REGION=0M
//SYSPRINT  DD SYSOUT=*
//SYSOUT    DD SYSOUT=*
//SYSIN     DD *
CREDENTIALS(ADD) PROVIDER(CLOUD1)
/*
//CREDSNAME DD *
/u/user1/add_S3_creds.json
/*
```

```
Contents of /u/user1/add_S3_creds.json
{
  "accessKey": "S3Cloud_access_key",
  "secretAccessKey": "S3Cloud_secret_key"
}
```

Example 10: Delete Credentials

The example below shows the JCL used to delete the default credentials entry for the CLOUD1 provider.

```
//EXEC      PGM=GDKUTIL,REGION=0M
//SYSPRINT  DD SYSOUT=*
//SYSOUT    DD SYSOUT=*
//SYSIN     DD *
CREDENTIALS(DELETE) PROVIDER(CLOUD1)
/*
```

Example 11: List Credential Entries

The first example below shows the JCL used to list the credential entries for the CLOUD1 provider for the current JOB userid. The second example below shows the JCL used to list the credential entries for all of the provider for the z/OS userid: USER256.

```
//EXEC      PGM=GDKUTIL,REGION=0M
//SYSPRINT  DD SYSOUT=*
//SYSOUT    DD SYSOUT=*
//SYSIN     DD *
CREDENTIALS(LIST) PROVIDER(CLOUD1)
/*
```

```
//EXEC      PGM=GDKUTIL,REGION=0M,USERID=USER1
//SYSPRINT  DD SYSOUT=*
//SYSOUT    DD SYSOUT=*
//SYSIN     DD *
CREDENTIALS(ADD) PROVIDER(*) ZUSERID(USER256)
/*
```

Example 12: Compress an object

In this example, the GDKUTIL utility is used to compress the object with the zEDC algorithm and upload it to a cloud.

```
//CRBUCKET EXEC PGM=GDKUTIL,REGION=0M
//SYSPRINT  DD SYSOUT=*
//SYSOUT    DD SYSOUT=*
//SYSIN     DD *
  UPLOAD COMPRESSION(ZEDC)
  PROVIDER(IBM COS)
  LOG(DEBUG)
/*
//OBJNAME   DD *
  /bucket-name/multi01/dir1/CavesofTerrorpg18970.txt
/*
```

```
//LOCNAME DD *
/OA66536/books/CavesofTerrorpg18970.txt
/*
```

Example 13: Decompress and download the object

In this example, the GDKUTIL utility is used to download an object. The user knows that the object has data compressed with the zEDC algorithm, and the object does not need have the zos-compression metadata tag attached to it.

```
//DOWNCOMP EXEC PGM=GDKUTIL,REGION=0M
//SYSPRINT DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//SYSIN DD *
  DOWNLOAD PROVIDER(IBMCO) DECOMPRESS(zEDC)
/*
//OBJNAME DD *
  /bucket-name/multi01/dir1/CavesofTerrorpg18970.txt
/*
//LOCNAME DD *
  /OA66536/downloads/CavesofTerrorpg18970.txt
/*
```

Example 14: Upload a data set, keeping record information

In this example, the GDKUTIL utility is used to upload a z/OS data set to object storage, keeping the record boundaries intact.

```
//CRBUCKET EXEC PGM=GDKUTIL,REGION=0M
//SYSPRINT DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//SYSIN DD *
  UPLOAD FORMAT(RECORD)
  PROVIDER(IBMCO)
/*
//OBJNAME DD *
  /bucket-name/backups/20250314/sysb/DFSMSDSS.DUMP
/*
//LOCNAME DD *
  BACKUPS.DFSMSDSS.D140325.DUMP
/*
```

Chapter 3. IEBCOMPR (Compare Data Sets) Program

Important: Use the SuperC utility instead of IEBCOMPR. SuperC is part of ISPF/PDF and the High Level Assembler Toolkit Feature. SuperC can be processed in the foreground as well as in batch and its report is more useful.

IEBCOMPR is a data set utility that is used to compare two sequential data sets, two partitioned data sets or two partitioned data sets (PDSEs) at the logical record level to verify a backup copy. Fixed, variable, or undefined records from blocked or unblocked data sets or members can also be compared. However, you should not use IEBCOMPR to compare load modules.

Note: If you compare PDS or PDSE members, IEBCOMPR cannot handle more than about 290 thousand members.

Two sequential data sets are considered *equal*, that is, are considered to be identical under the following conditions:

- The data sets contain the same number of records.
- Corresponding records and keys are identical.

Two partitioned data sets or two PDSEs are considered equal under all of the following conditions:

- Corresponding members contain the same number of records.
- Note lists are in the same position within corresponding members.
- Corresponding records and keys are identical.
- Corresponding directory user data fields are identical.

If all these conditions are not met for a specific type of data set, those data sets are considered unequal. If records are unequal, the record and block numbers, the names of the DD statements that define the data sets, and the unequal records are listed in a message data set. Ten successive unequal comparisons stop the job step, unless you provide a routine for handling error conditions.

Load module partitioned data sets that reside on different types of devices should not be compared. Under most circumstances, the data sets will not compare as equal.

Partitioned data sets or PDSEs can be compared only if all the names in one or both of the directories have counterpart entries in the other directory. The comparison is made on members that are identified by these entries and corresponding user data.

Figure 1 on page 31 shows the directories of two partitioned data sets. Directory 2 contains corresponding entries for all the names in Directory 1; therefore, the data sets can be compared.

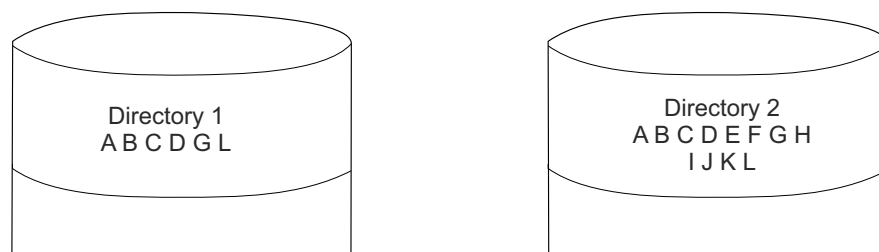


Figure 1. Partitioned directories whose data sets can be compared by using IEBCOMPR

Figure 2 on page 32 shows the directories of two partitioned data sets. Each directory contains a name that has no corresponding entry in the other directory; therefore, the data sets cannot be compared, and the job step will be ended.

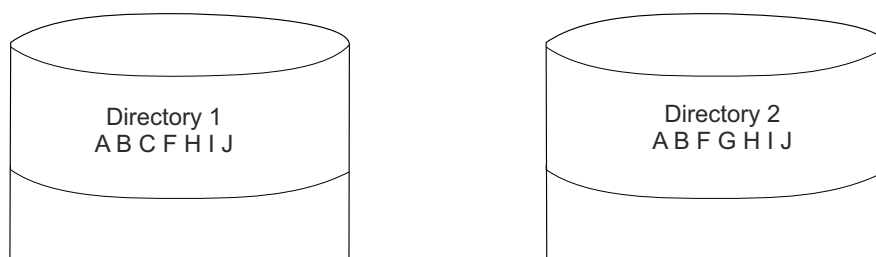


Figure 2. Partitioned directories whose data sets cannot be compared by using IEBCOMPR

User exits are provided for optional user routines to process user labels, handle error conditions, and modify source records.

If IEBCOMPR is invoked from an application program or TSO, you can dynamically allocate the data sets by calling dynamic allocation (SVC 99) or the TSO ALLOCATE command before calling IEBCOMPR.

Related reading: For information about the linkage conventions for user routines, see [Appendix C, “Specifying User Exits with Utility Programs,”](#) on page 343.

Input and Output

IEBCOMPR uses the following input:

- Two sequential data sets, partitioned data sets, PDSEs, or z/OS UNIX files to be compared.
- A control data set that contains utility control statements. This data set is required if the input data sets are partitioned or PDSEs, or if user routines are used.

IEBCOMPR produces as output a message data set that contains informational messages (for example, the contents of utility control statements), the results of comparisons, and error messages.

Related reading: For information about IEBCOMPR return codes, see [Appendix A, “Invoking Utility Programs from an Application Program,”](#) on page 315.

Control

IEBCOMPR is controlled by job and utility control statements. The job control statements are required to process IEBCOMPR and to define the data sets that are used and produced by IEBCOMPR. The utility control statements are used to indicate the input data set organization (that is, sequential, partitioned, or PDSE), to identify any user routines that may be provided, and to indicate if user labels are to be treated as data.

Job Control Statements

Table 6 on page 32 shows the job control statements for IEBCOMPR.

Table 6. Job control statements for IEBCOMPR

Statement	Use
JOB	Starts the job.
EXEC	Specifies the program name (PGM=IEBCOMPR) or, if the job control statements reside in a procedure library, the procedure name.
SYSPRINT DD	Defines a sequential data set which will be used for messages produced by IEBCOMPR. This data set can be written to a system output device (SYSOUT), a tape volume, a direct access volume, TSO terminal, or dummy (DUMMY DD).
SYSUT1 DD	Defines an input data set or z/OS UNIX file to be compared.

Table 6. Job control statements for IEBCOMPR (continued)

Statement	Use
SYSUT2 DD	Defines an input data set or z/OS UNIX file to be compared.
SYSIN DD	Defines the control data set or specifies DUMMY if the input data sets are sequential and no user routines are used. The control data set normally resides in the input stream; however, it can be defined as a member within a library of partitioned members.

One or both of the input data sets can be passed from a preceding job step.

You can compare data sets that reside on different device types. However, you should not compare load module libraries that reside on different device types. You can also compare sequential data sets that were written at different densities.

The SYSPRINT DD statement must be present for each use of IEBCOMPR. The block size specified in the SYSPRINT DD statement must be a multiple of 121.

The SYSIN DD statement is required. The block size specified in the SYSIN DD statement must be a multiple of 80.

The input data sets must have the same logical record length. Otherwise, a comparison of the two data sets will show them to be unequal. The block sizes of the input data sets can differ. For fixed block (FB) data sets, block sizes must be multiples of the logical record length. The block sizes cannot exceed 32760 bytes.

Utility Control Statements

Table 7 on page 33 shows the utility control statements that are used to control IEBCOMPR.

Table 7. IEBCOMPR utility control statements

Statement	Use
COMPARE	Indicates the organization of a data set.
EXITS	Identifies user exit routines to be used.
LABELS	Indicates if user labels are to be treated as data by IEBCOMPR.

Continuation requirements for utility control statements are described in [“Continuing utility control statements”](#) on page 8.

COMPARE Statement

You use the COMPARE statement to indicate the organization of the data sets you want to compare.

The COMPARE statement, if included, must be the first utility control statement. COMPARE is required if you use the EXITS or LABELS statement or if the input data sets are partitioned data sets or PDSEs.

The syntax of the COMPARE statement is:

Label	Statement	Parameters
[<i>label</i>]	COMPARE	TYPORG={ <i>PS</i> <i>PO</i> }

where:

TYPORG={*PS*|*PO*}

specifies the organization of the input data sets. The values that can be coded are:

PS

specifies that the input data sets are sequential data sets. This is the default.

PO

specifies that the input data sets are partitioned data sets or PDSEs.

EXITS Statement

You use the EXITS statement to identify any exit routines you want to use. If an exit routine is used, the EXITS statement is required. If you use more than one EXITS statement, IEBCOMPR only uses the last EXITS statement. All others are ignored.

The syntax of the EXITS statement is:

Label	Statement	Parameters
[<i>label</i>]	EXITS	[INHDR= <i>routinename</i>] [,INTLR= <i>routinename</i>] [,ERROR= <i>routinename</i>] [,PRECOMP= <i>routinename</i>]

where:

INHDR=*routinename*

specifies the name of the routine that processes user input header labels.

INTLR=*routinename*

specifies the name of the routine that processes user input trailer labels.

ERROR=*routinename*

specifies the name of the routine that is to receive control for error handling after each unequal comparison. If this parameter is omitted and ten consecutive unequal comparisons occur while IEBCOMPR is comparing sequential data sets, processing is stopped; if the input data sets are partitioned or PDSE, processing continues with the next member.

PRECOMP=*routinename*

specifies the name of the routine that processes logical records (physical blocks in the case of variable spanned (VS) or variable blocked spanned (VBS) records longer than 32K bytes) from either or both of the input data sets before they are compared.

Related reading: For information about processing of user labels as data set descriptors, see [“Processing User Labels”](#) on page 347.

LABELS Statement

You use the LABELS statement to specify whether user labels are to be treated as data by IEBCOMPR. For a discussion of this option, refer to [“Processing User Labels”](#) on page 347.

If you use more than one LABELS statement, IEBCOMPR will only use the last LABELS statement. All others will be ignored.

Table 8 on page 34 shows the syntax of the LABELS statement.

Table 8. Syntax of LABEL statement		
Label	Statement	Parameters
[<i>label</i>]	LABELS	[DATA={ <i>YES</i> <i>NO</i> <i>ALL</i> <i>ONLY</i> }]

DATA={*YES* | *NO* | *ALL* | *ONLY*}

specifies if user labels are to be treated as data. The values that can be coded are:

YES

specifies that any user labels that are not rejected by a user's label processing routine are to be treated as data. Processing of labels as data stops in compliance with standard return codes. YES is the default.

NO

specifies that user labels are not to be treated as data.

ALL

specifies that all user labels are to be treated as data. A return code of 16 causes IEBCOMPR to complete processing of the remainder of the group of user labels and to end the job step.

ONLY

specifies that only user header labels are to be treated as data. User header labels are processed as data regardless of any return code. The job ends upon return from the OPEN routine.

Requirement: LABELS DATA=NO must be specified to make IBM standard user label (SUL) exits inactive when input/output data sets with nonstandard labels (NSL) are to be processed.

IEBCOMPR Examples

The examples in [Table 9 on page 35](#) illustrate some of the uses of IEBCOMPR. The numbers in the "Example" column refer to examples that follow.

Examples that use **disk** or **tape** in place of actual device names or numbers must be changed before use. The actual device names or numbers depend on how your installation has defined the devices to your system.

Table 9. IEBCOMPR example directory

Operation	Data Set Organization	Devices	Comments	Example
COMPARE	Partitioned	Disk	No user routines. Blocked input.	“Example 6: Compare Two Partitioned Data Sets” on page 38
COMPARE	PDSE	Disk	No user routines. SMS-managed data sets.	“Example 8: Compare Two PDSEs” on page 39
COMPARE	Sequential	9-track Tape	No user routines. Blocked input.	“Example 1: Compare Data Sets that Reside on Tape” on page 35
COMPARE	Sequential	7-track Tape	No user routines. Blocked input.	“Example 2: Compare Sequential Data Sets that Reside on Tape” on page 36
COMPARE	Sequential	7-track Tape and 9-track Tape	User routines. Blocked input. Different density tapes.	“Example 3: Compare Sequential Data Sets Written at Different Densities” on page 36
COMPARE	Sequential	System input stream, 9-track Tape	No user routines. Blocked input.	“Example 4: Compare Sequential Data Sets—Input Stream and Tape Input” on page 37
COPY (using IEBGENER) and COMPARE	Sequential	Disk or Tape	No user routines. Blocked input. Two job steps; data sets are passed to second job step.	“Example 5: Copy and Compare Sequential Data Set in Two Job Steps” on page 37
COPY (using IEBCOPY) and COMPARE	Partitioned	Disk	User routine. Blocked input. Two job steps; data sets are passed to second job step.	“Example 7: Copy and Compare Partitioned Data Set in Two Job Steps” on page 38

Example 1: Compare Data Sets that Reside on Tape

In this example, two sequential data sets that reside on 9-track tape volumes are compared.

```
//TAPETAPE JOB ...
// EXEC PGM=IEBCOMPR
//SYSPRINT DD SYSOUT=A
//SYSUT1 DD DSN=SET1,UNIT=tape,LABEL=(,NL),
// DCB=(RECFM=FB,LRECL=80,BLKSIZE=2000),
// DISP=(OLD,KEEP),VOLUME=SER=001234
```

```
//SYSUT2 DD DSN=SET2,UNIT=tape,LABEL=(,NL),
//          DCB=(RECFM=FB,LRECL=80,BLKSIZE=1040),
//          DISP=(OLD,KEEP),VOLUME=SER=001235
//SYSIN DD DUMMY
/*
```

The job control statements are discussed, as follows:

- SYSUT1 DD defines an input data set (SET1), which resides on an unlabeled 9-track tape volume.
- SYSUT2 DD defines an input data set (SET2), which resides on an unlabeled 9-track tape volume.
- SYSIN DD defines a dummy data set. Because no user routines are used and the input data sets have a sequential organization, utility control statements are not necessary.

Example 2: Compare Sequential Data Sets that Reside on Tape

In this example, two sequential data sets that reside on 7-track tape volumes are compared.

```
//TAPETAPE JOB ...
//          EXEC PGM=IEBCOMPR
//SYSPRINT DD SYSOUT=A
//SYSUT1 DD DSN=SET1,LABEL=(2,SUL),DISP=(OLD,KEEP),
//          VOL=SER=001234,DCB=(DEN=2,RECFM=FB,LRECL=80,
//          BLKSIZE=2000,TRTCH=C),UNIT=tape
//SYSUT2 DD DSN=SET2,LABEL=(,SUL),DISP=(OLD,KEEP),
//          VOL=SER=001235,DCB=(DEN=2,RECFM=FB,LRECL=80,
//          BLKSIZE=2000,TRTCH=C),UNIT=tape
//SYSIN DD *
//          COMPARE TYPORG=PS
//          LABELS DATA=ONLY
/*
```

The control statements are discussed, as follows::

- SYSUT1 DD defines an input data set, SET1, which resides on a labeled, 7-track tape volume. The blocked data set was originally written at a density of 800 bits per inch (DEN=2) with the data converter on (TRTCH=C).
- SYSUT2 DD defines an input data set, SET2, which is the first or only data set on a labeled, 7-track tape volume. The blocked data set was originally written at a density of 800 bits per inch (DEN=2) with the data converter on (TRTCH=C).
- SYSIN DD defines the control data set, which follows in the input stream.
- COMPARE TYPORG=PS specifies that the input data sets are sequentially organized.
- LABELS DATA=ONLY specifies that user header labels are to be treated as data and compared. All other labels on the tape are ignored.

Example 3: Compare Sequential Data Sets Written at Different Densities

In this example, two sequential data sets that were written at different densities on different tape units are compared.

```
//TAPETAPE JOB ...
//          EXEC PGM=IEBCOMPR
//SYSPRINT DD SYSOUT=A
//SYSUT1 DD DSN=SET1,LABEL=(,SUL),DISP=(OLD,KEEP),
//          VOL=SER=001234,DCB=(DEN=1,RECFM=FB,LRECL=80,
//          BLKSIZE=320,TRTCH=C),UNIT=tape
//SYSUT2 DD DSN=SET2,LABEL=(,SUL),DISP=(OLD,KEEP),
//          DCB=(RECFM=FB,LRECL=80,BLKSIZE=640),
//          UNIT=tape,VOLUME=SER=001235
//SYSIN DD *
//          COMPARE TYPORG=PS
//          EXITS INHDR=HDRS,INTLR=TLRS
//          LABELS DATA=NO
/*
```

The control statements are discussed, as follows::

- SYSUT1 DD defines an input data set, SET1, which is the first or only data set on a labeled, 7-track tape volume. The blocked data set was originally written at a density of 556 bits per inch (DEN=1) with the data converter on (TRTCH=C).
- SYSUT2 DD defines an input data set, SET2, which is the first or only blocked data set on a labeled tape volume. In this example, assume SYSUT2 is on a 9-track tape drive.
- SYSIN DD defines the control data set, which follows in the input stream.
- COMPARE TYPORG=PS specifies that the input data sets are sequentially organized.
- EXITS identifies the names of routines to be used to process user input header labels and trailer labels.
- LABELS DATA=NO specifies that the user input header and trailer labels for each data set are not to be compared.

Example 4: Compare Sequential Data Sets—Input Stream and Tape Input

In this example, two sequential data sets (input stream and tape) are compared.

```
//CARDTAPE JOB ...
//          EXEC PGM=IEBCOMPR
//SYSPRINT DD SYSOUT=A
//SYSIN    DD DUMMY
//SYSUT2   DD UNIT=tape,VOLUME=SER=001234,LABEL=(,NL),
//          DCB=(RECFM=FB,LRECL=80,BLKSIZE=2000),
//          DISP=(OLD,KEEP)
//SYSUT1   DD DATA

(input data set)

/*
```

The control statements are discussed, as follows::

- SYSIN DD defines a dummy control data set. Because no user routines are provided and the input data sets are sequential, utility control statements are not necessary.
- SYSUT2 DD defines an input data set, which resides on an unlabeled, tape volume.
- SYSUT1 DD defines a system input stream data set.

Example 5: Copy and Compare Sequential Data Set in Two Job Steps

In this example, a sequential disk or tape data set is copied and compared in two job steps.

```
//TAPETAPE JOB ...
//STEP A    EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=A
//SYSUT1   DD DSN=WAREHOUS.COPYSET1,DISP=(OLD,PASS),
//SYSUT2   DD DSN=WAREHOUS.COPYSET2,DISP=(,PASS),LABEL=(,SL),
//          DCB=(RECFM=FB,LRECL=80,BLKSIZE=640),
//          UNIT=tape,VOLUME=SER=001235
//SYSIN    DD DUMMY
//STEP B    EXEC PGM=IEBCOMPR
//SYSPRINT DD SYSOUT=A
//SYSUT1   DD DSN=*.STEP A.SYSUT1,DISP=(OLD,KEEP)
//SYSUT2   DD DSN=*.STEP A.SYSUT2,DISP=(OLD,KEEP)
//SYSIN    DD DUMMY
```

The first job step copies the data set and passes the original and copied data sets to the second job step. The second job step compares the two data sets.

The control statements for the IEBCOMPR job step are discussed, as follows:

- SYSUT1 DD defines an input data set passed from the preceding job step (COPYSET1). The data set resides on a labeled tape volume.
- SYSUT2 DD defines an input data set passed from the preceding job step (COPYSET2). The data set, which was created in the preceding job step, resides on a labeled tape volume.
- SYSIN DD defines a dummy control data set. Because the input is sequential and no user exits are provided, no utility control statements are required.

Example 6: Compare Two Partitioned Data Sets

In this example, two partitioned data sets are compared.

```
//DISKDISK JOB ...
//STEP1 EXEC PGM=IEBCOMPR
//SYSPRINT DD SYSOUT=A
//SYSUT1 DD DSN=PDSET1,UNIT=disk,DISP=SHR,
//          DCB=(RECFM=FB,LRECL=80,BLKSIZE=2000),
//          VOLUME=SER=111112
//SYSUT2 DD DSN=PDSET2,UNIT=disk,DISP=SHR,
//          DCB=(RECFM=FB,LRECL=80,BLKSIZE=2000),
//          VOLUME=SER=111113
//SYSIN DD *
//          COMPARE TYPORG=PO
/*
```

The control statements are discussed, as follows:

- SYSUT1 DD defines an input partitioned data set, PDSSET1. The blocked data set resides on a disk volume.
- SYSUT2 DD defines an input partitioned data set, PDSSET2. The blocked data set resides on a disk volume.
- SYSIN DD defines the control data set, which follows in the input stream.
- COMPARE TYPORG=PO indicates that the input data sets are partitioned.

Example 7: Copy and Compare Partitioned Data Set in Two Job Steps

In this example, a partitioned data set is copied and compared in two job steps.

```
//DISKDISK JOB ...
//STEP1 EXEC PGM=IEBCOPY
//SYSPRINT DD SYSOUT=A
//SYSUT1 DD DSN=MAINDB.LOG.OLDSET,DISP=SHR
//SYSUT2 DD DSN=NEWMEMS,UNIT=disk,DISP=(,PASS),
//          VOLUME=SER=111113,SPACE=(TRK,(5,5,5)),
//          DCB=(RECFM=FB,LRECL=80,BLKSIZE=640)
//SYSUT3 DD UNIT=SYSDA,SPACE=(TRK,(1))
//SYSUT4 DD UNIT=SYSDA,SPACE=(TRK,(1))
//SYSIN DD *
//          COPY OUTDD=SYSUT2,INDD=SYSUT1
//          SELECT MEMBER=(A,B,D,E,F)
/*
//STEP2 EXEC PGM=IEBCOMPR
//SYSPRINT DD SYSOUT=A
//SYSUT1 DD DSN=OLDSET,DISP=(OLD,KEEP)
//SYSUT2 DD DSN=NEWMEMS,DISP=(OLD,KEEP)
//SYSIN DD *
//          COMPARE TYPORG=PO
//          EXITS ERROR=SEEERROR
/*
```

The first job step copies the data set and passes the original and copied data sets to the second job step. The second job step compares the two data sets.

The control statements for the IEBCOMPR job step are discussed, as follows:

- SYSUT1 DD defines a blocked input data set (MAINDB.LOG.OLDSET) that is passed from the preceding job step. The data set resides on a disk or tape volume.
- SYSUT2 DD defines a blocked input data set (MAINDB.LOG.NEWMEMS) that is passed from the preceding job step. The data set resides on a disk volume.
- SYSIN DD defines the control data set, which follows in the input stream.
- COMPARE TYPORG=PO specifies partitioned organization.
- EXITS specifies that a user error routine, SEEERROR, is to be used.

Because the input data set names are not identical, the data sets can be retrieved by their data set names.

Example 8: Compare Two PDSEs

In this example, two PDSEs are compared.

```
//DISKDISK JOB ...  
// EXEC PGM=IEBCOMPR  
//SYSPRINT DD SYSOUT=A  
//SYSUT1 DD DSN=PDSE1,DISP=SHR  
//SYSUT2 DD DSN=PDSE2,DISP=SHR  
//SYSIN DD *  
 COMPARE TYPORG=PO  
/*
```

PDSEs no longer must be SMS managed. Because these PDSEs are cataloged, you need not specify the UNIT or VOLUME parameters.

The control statements are discussed, as follows:

- SYSUT1 DD and SYSUT2 DD define input PDSEs, PDSE1, and PDSE2. Because no DCB values are specified, the DCB values that were specified in creating the data sets will be used.
- SYSIN DD defines the control data set, which follows in the input stream.
- COMPARE TYPORG=PO indicates that the input data sets are PDSEs.

Chapter 4. IEBCOPY (Library Copy) Program

IEBCOPY is a data set utility that is used to copy or merge members between one or more partitioned data sets, or partitioned data sets extended (PDSEs), in full or in part. You can also use IEBCOPY to create a backup of a partitioned data set into a sequential data set (called an unload data set or PDSU), and to copy members from the backup into a partitioned data set.

Starting in z/OS V1R13, IEBCOPY is no longer APF-authorized. It does not need to be run from an authorized library, and calling programs do not need to be APF-authorized.

You can use IEBCOPY to perform the following tasks:

- Make a copy of a partitioned data set or PDSE.
- Merge partitioned data sets (except when unloading).
- Create a sequential form of a partitioned data set or PDSE for a backup or transport.
- Reload one or more members from a PDSU into a partitioned data set or PDSE. (Note that an unloaded load module cannot be converted to a program object in this process.)
- Select specific members of a partitioned data set or PDSE to be copied, loaded, or unloaded.
- Replace members of a partitioned data set or PDSE.
- Rename selected members of a partitioned data set or PDSE when copied.
- Exclude members from a data set to be copied, unloaded, or loaded. (Except on COPYGRP).
- Compress a partitioned data set in place.
- Upgrade a load module for faster loading by MVS program fetch.
- Copy and reblock load modules.
- Convert load modules in a partitioned data set to program objects in a PDSE when copying a partitioned data set to a PDSE.
- Convert a partitioned data set to a PDSE or a PDSE to a partitioned data set.
- Copy to or from a PDS or PDSE data set, a member and its aliases together as a group (COPYGROUP).
- Copy the member generations of a PDSE V2 data set to another (COPYGRP, COPYGROUP).

In addition, IEBCOPY automatically lists the number of unused directory blocks and the number of unused tracks that are available for member records if the output data set is a partitioned data set.

Related reading: For important information on shared partitioned data sets see [“Job Control Statements”](#) on page 6.

Converting Load Modules to Program Objects or the Reverse

Program objects are created automatically when load modules are copied into a PDSE. Likewise, program objects are automatically converted back to load modules when they are copied into a partitioned data set. Note that some program objects cannot be converted into load modules because they use features of program objects that do not exist in load modules.

IEBCOPY is not able to *directly* convert between program objects and load modules when loading or unloading a PDSE or partitioned data set. A load operation can only reload load modules into partitioned data sets or reload program objects into a PDSE. Unloading a partitioned data set can only place load modules into the unload data set. Similarly, unloading a PDSE can only place program objects into the unload data set.

Therefore, to convert an unloaded load module into a program object, reload the load module into a partitioned data set and then copy the partitioned data set to a PDSE. To convert a program object into an unloaded load module, copy the PDSE to a partitioned data set and then unload the partitioned data set.

If your partitioned data set contains both load modules and data members, then you will have to convert the partitioned data set into two separate PDSEs - one for program objects and the second for data members.

Related reading: For more information about differences between program objects and load modules, see *z/OS MVS Program Management: User's Guide and Reference*.

Converting Partitioned Data Sets to PDSEs

You can use IEBCOPY to convert partitioned data sets to PDSEs.

To convert a partitioned data set to a PDSE, create a PDSE and copy the partitioned data set into the new PDSE. This can be accomplished with one use of IEBCOPY.

You cannot convert a partitioned data set that has any of the following features:

- Both load modules and nonload modules in the same partitioned data set. Individual members of a partitioned data set can be converted by copying them into two separate PDSEs, the first for data and the second for program objects.
- Note lists. Load modules that contain note lists can be placed into PDSEs because they are converted automatically into program objects.
- Nonzero key lengths in the members.

Copying Data Sets

IEBCOPY can be used to totally or partially copy a partitioned data set from one direct access volume to another. In addition, a data set can be copied to its own volume, provided its data set name is changed. (If the data set name is not changed, IEBCOPY interprets the request as a compress-in-place, provided the SELECT / EXCLUDE statement is not coded.)

When you use IEBCOPY to copy a PDSE to a PDSE, either volume to volume or to its own volume, all DDM attributes are also copied.

Members copied into a partitioned data set are not physically reordered; members are copied in the physical order in which they occur in the original data set.

Merging Data Sets

Merging data sets is done by copying or loading the additional members to an existing partitioned data set. The merge operation (ordering of the directory of the output data set) is automatically performed by IEBCOPY.

Increasing Directory Space for a Partitioned Data Set

IEBCOPY cannot increase the number of directory blocks in a partitioned data set. (A PDSE directory automatically expands as needed.) If you are not sure there will be enough directory blocks in the output partitioned data set you are merging to, then you should expand the output data set directory space before beginning the merge operation.

Use IEHLIST to determine how much directory space remains in a partitioned data set. If more blocks are needed, this procedure will extend most data sets:

1. Rename the data set.
2. Allocate a new data set with enough space and directory blocks.
3. Copy the renamed data set to the newly allocated data set.
4. Delete the renamed data set (or save it as a backup).

Unloading (Backing up) Data Sets

IEBCOPY can be used to create a backup copy of a partitioned data set by copying (unloading) it to a sequential data set on DASD, tape, or other device supported by QSAM.

IEBCOPY creates an unload data set when you specify physical sequential organization (DSORG=PS) for the output data set. To create a partitioned data set, specify DSORG=PO and DSNTYPE=PDS or DSNTYPE=LIBRARY.

Requirement: If you do not explicitly state the DSORG, it might be set opposite to your intention by ACS routines or other JCL parameters. For example, if you specify LIKE= or DCB= or a model DSCB and omit the DSORG, then whatever DSORG the referenced object has will be implicitly added to your DD statement. Always specify a DSORG if you are not sure what will be taken from a referenced object.

Attention: Do not change the DCB parameters of an unload data set after IEBCOPY finishes creating it, or IEBCOPY might not be able to reload it.

To unload more than one partitioned data set to the same tape volume in one execution of IEBCOPY, multiple copy operations must be used and multiple sequential data sets must be allocated to successive files on the tape.

IEBCOPY can copy a PDSU to a PDSU directly without the need to reload it to a PDS and then unload the PDS to create the new PDSU. If a selective copy is not required then it will be faster to use IEBGENER to copy the PDSU to a new PDSU.

Only a COPY operation can create an unload data set; COPYMOD cannot.

Copying Directory Information between a Partitioned Data Set and a PDSE

The PDSE directory can contain attributes in addition to those traditionally kept in a partitioned data set directory entry.

Some PDSE extended attributes are recorded on an unload data set and will be reloaded when the target is a PDSE.

If you reload an unloaded PDSE that contains *program objects* to a partitioned data set, an error message is issued and the operation fails.

Information that is kept as user data in a partitioned data set directory, such as PDF statistics, will move to a PDSE directory and back from a PDSE directory to a partitioned data set directory without change.

Loading or Copying Unload Data Sets

Using IEBCOPY, you can re-create a partitioned data set from an unloaded copy of a partitioned data set by copying the sequential (unloaded) data set to a partitioned data set. A partitioned data set may be loaded from an unloaded PDSE if it does not contain program objects, and a PDSE may be loaded from an unloaded partitioned data set if it does not contain load modules.

Note: <A load operation cannot convert unloaded load modules to program objects or program objects to unloaded load modules.

You can create a single partitioned data set from multiple input sequential (unloaded) data sets.

For unload and load operations, requests are handled in the same way as for a copy operation. You can choose to process specific members and to rename them.

A partitioned data set in the unload format will have a variable spanned record format. When the unload data set is subsequently loaded, the output data set will have the same characteristics it had before the unload operation, unless you provide overriding characteristics when you reload the data set.

IEBCOPY Unload Data Set DCB Parameters

An unload data set is always a variable spanned record format with sequential organization, (RECFM=VS and DSORG=PS).

The logical record length of the unload data set is intended to hold a block from the input data set plus a header, with these considerations:

1. The LRECL is calculated as being the larger of:
 - a. 280 bytes, or
 - b. 16 bytes + the block size + the key length of the input data set.
2. If the LRECL exceeds 32760, it is reduced to 32760.

Note: Applications reading an unload data set should be aware that for RECFM=VS data sets, the actual length of an assembled logical record could exceed 32760 bytes, even if the LRECL was reduced to 32760 by this step.

Note: When the input data set is a PDSE and has a block size greater than 32744, the LRECL of the unload data set is set to X, i.e. LRECL=32768. LRECL=X is used to process logical records that exceed 32760 bytes.

3. If the user supplies an LRECL larger than the one calculated here, it will be placed in the data set label; however, the size of the logical record that IEBCOPY creates will not be increased.

The block size (BLKSIZE) for an unload data set is determined by the following steps:

1. The initial block size is set to the block size supplied by the user, or if the user did not supply a block size, it is calculated as the LRECL plus 4.
2. If the block size is less than 284, it is increased to 284.
3. If the block size exceeds 32760, it is reduced to 32760.
4. The block size value is then compared with the largest block size acceptable to the output device. If the output device capacity is smaller than the block size, it is set to the maximum allowed for the output device.

Because the unload data set is unblocked, increasing the block size beyond LRECL plus 4 will not result in longer physical records or better utilization.

The block size is stored in the first control record (COPYR1) and used at load time. If the block size of the unload data set is changed after it is created, IEBCOPY might not be able to reload it.

Recommendation: Do not set the PDSU block size equal to the PDS block size or your PDSU will have very poor space utilization and performance. Let IEBCOPY pick the block size or choose a PDSU block size 20 bytes greater than the PDS block size.

Selecting Members to be Copied, Unloaded, or Loaded

Select specific members to be processed from one or more data sets by coding a SELECT statement to name the members. Alternatively, all members but a specific few can be designated by coding an EXCLUDE statement to name members not to be processed.

You cannot use both a SELECT and an EXCLUDE statement in the same copy operation (same set of input ddnames).

A maximum of eight characters can be given for the member or alias name on a copy operation.

Selected members are searched for in a low-to-high (a-to-z) collating sequence, regardless of the order in which they are specified on the SELECT statement; however, they are copied in the same physical sequence in which they appear on the input partitioned data set or PDSE.

Once a member designated in a SELECT statement is found in an input data set, no search is made for it on any subsequent input data set. When all of the selected members are found, the operation ends, even if all data sets have not yet been processed.

Example:

If members A and B are specified and A is found on the first of three input data sets, it is not searched for again when the second and third data sets are searched.

If B is found on the second input data set, the operation is successfully ended after the second input data set has been processed, and the third input data set is never examined.

When you copy, you can rename members unless the input and output data sets are the same.

When using COPYGROUP, if you specify SELECT MEMBER name filter pattern masking, you can also specify EXCLUDE MEMBER name filter pattern masking. Otherwise, EXCLUDE is not supported with COPYGROUP.

When using COPYGROUP, you can use SELECT MEMBER name filter pattern masking instead of specifying a full member name. The filter characters asterisk (*) and percent sign (%) provide the member name filtering capability. With SELECT MEMBER name filter pattern masking, all of the members of the input data set that match the filter mask pattern criteria will be selected.

You cannot use both a SELECT and an EXCLUDE statement in the same COPY operation (same set of input ddnames); they are mutually exclusive.

Excluding Members from a Copy Operation

Members from one or more input data sets can be excluded from a copy, unload, or load operation. The excluded member is searched for on every input data set in the copy, unload, or load operation and is always omitted. Members are excluded from the input data sets named on an INDD statement that precedes the EXCLUDE statement.

A maximum of eight characters can be given for the member or alias name on a copy operation.

Restriction: EXCLUDE is not allowed for COPYGRP or for COPYGROUP. It is allowed for COPYGROUP when a SELECT MEMBER statement uses member name filter pattern masking, and the EXCLUDE statement does also.

The replace option can be specified on the data set level in an exclusive copy or load, in which case, nonexcluded members on the input data set replace identically named members on the output data set.

Related reading: For more information about the replace option, see [“Replacing Members in a Data Set”](#) on page 46.

Copying Members That Have Alias Names (COPY Statement)

This topic discusses using the COPY statement for copying a PDS, PDSU, or PDSE that has members with alias names. The COPYGRP or COPYGROUP statement is recommended for copying *program objects*.

Tip: If the COPY statement is used to copy program objects, errors can occur.

- If you are copying an entire data set to a new data set (one that has no members before the copy operation), all members and their aliases will be copied, and they will have the same relationship to one another as they had on the original data set.
- If you are merging a data set with another data set, no members or aliases on the output data set will be changed unless you specify that input members are to replace output members.

Example:

In all instances, if you have a member A with alias B on your input data set, and a member C with alias B on your output data set, if you do *not* indicate replacement, member A will be copied over, but the alias name B will continue to refer to C. If you *do* indicate replacement, B will be copied as an alias of member A in the newly merged data set.

- When selectively copying from a partitioned data set, you must specify every name that you want copied, including their aliases.

Example:

If you are selecting member A, and member A has the aliases B and C, to copy all three names you must specify SELECT MEMBER=(A,B,C). This will result in *one* copy of the data for that member, and all three names placed in the directory and associated with that member data.

If you specify `SELECT MEMBER=(A,C)`. This will result in *one* copy of the data for that member, and two names, A and C, placed in the directory and associated with that member data.

- When copying to or from a PDSE, you must specify the member's name. An alias will not become the member name in the output data set.
- If you are exclusively copying a data set (using the `EXCLUDE` statement), you must specify not only a member's name, but also all of its alias names to exclude the member data from the copy operation.

Example:

If you want to exclude member A from the copy operation, and A has the alias names B and C, you must specify `EXCLUDE MEMBER=(A,B,C)`. If you specify only `MEMBER=A`, then the member is copied to the output data set with the alias names B and C.

- The rules for replacing or renaming members apply to both aliases and members; no distinction is made between them.

Related reading: For information about copying program objects, see [“Copying Program Objects \(COPYGRP and COPYGROUP Statements\)”](#) on page 47.

Replacing Members in a Data Set

You can use IEBCOPY's `COPY` or `COPYGRP` or `COPYGROUP` statement to replace members on an output partitioned data set or PDSE. The explanations in this topic are for the `COPY` statement.

With the `COPY` statement you can perform the following tasks:

- You can specify replacement on the data set level. In this case, every member of an input data set will be copied to the output data set. Each member on the output data set that has a name identical to the name of a member on the input data set will be replaced.
- You can indicate replacement on the member level. In this case, you can indicate that a particular member of the input data set is to replace an identically named member of the output data set, and indicate that another member is to be copied only if it is not already present on the output data set.

When you specify replacement on the member level, you can also rename an input member. The output data set directory is searched for the new name to see if the member should be copied. For instance, you could rename member A to B, and have it replace member B on the output data set.

Related reading: For information about replacing members using `COPYGRP`, See [“Copying Program Objects \(COPYGRP and COPYGROUP Statements\)”](#) on page 47.

Specifying Replacement on the Data Set Level

When you merge partitioned data sets, or load an unload data set into a partitioned data set that already has members, the input and output data sets might have members with the same names. Under normal processing, these input members will not replace the output members that have their names. To specify that all input members are to be copied to the output data set, and thus replace any output members of the same name, use the replace (**R**) option on an `INDD` or `COPY` statement.

When replace (**R**) is specified on the data set level, the input data is processed as follows:

- In a full copy or load process, all members in an input data set are copied to an output partitioned data set; members whose names already exist in the output partitioned data set are replaced by members copied or loaded from the input data set.
- In a selective copy or load process, all selected input members will be copied to the output data set, replacing any identically named output data set members. Specifying replace (**R**) on the data set level when performing a selective copy relieves you of the need to specify replace (**R**) for each member you want copied.
- In an exclusive copy process, all nonexcluded members on the input data sets are copied or loaded to an output partitioned data set, replacing those members with duplicate names on the output partitioned data set.

Specifying Replacement on the Member Level

When you specify the name of a member to be copied in the MEMBER operand of the SELECT statement, specify the replace (R) option for input members which have identically named members in the output data set. In this way, you can copy many members from an input data set, but allow only a few of them to replace members in the output data set.

Processing Considerations for Replacing Members

There are differences between full, selective, and exclusive copy or load processing. These differences should be remembered when specifying the replace (R) option on either the data set or member level, and when the output data set contains member names common to some or all the input data sets being copied or loaded. These differences are:

- When a full copy or load is performed, the output partitioned data set contains the replacing members that were on the *last* input data set copied.
- When a selective copy or load is performed, the output partitioned data set contains the selected replacing members that were found on the *earliest* input data set searched. After a selected member is found, it is not searched for again. Therefore, after it is found, a selected member is copied or loaded. If the same member exists on another input data set, it is not searched for, and hence, not copied or loaded.
- When an exclusive copy or load is performed, the output partitioned data set contains all members, except those specified for exclusion, that were on the *last* input data set copied or loaded.

Renaming Selected Members

Using the SELECT statement to rename members

To use the SELECT statement to rename members, you should place the old name, the new name, and optionally a replace (R) indicator together inside parentheses as an operand of the MEMBER parameter on a SELECT statement. MEMBER parameter operands may consist of as many old name/new name/replace sets as you need.

The replace (R) option must be used if the new name matches a name of a member in the output data set, or the member will not be copied. It does not matter if replace is indicated globally for all members by using the INDD parameter, or if it is indicated for individual members in the MEMBER parameter.

The selected members are not renamed in the input data set directory. They are just added to the output data set with the new name.

Copying Program Objects (COPYGRP and COPYGROUP Statements)

It is recommended that you use the COPYGRP or COPYGROUP statement to copy PDS load modules or PDSE program objects and their aliases from or to a PDSE data set. Program objects can have aliases that are longer than eight characters. Using the COPYGRP or COPYGROUP statement will ensure that these longer aliases are copied along with their member.

Use the COPYGRP or COPYGROUP statement to begin a *group* copy, unload, or load. A *group* consists of a member and all of its aliases. COPYGRP or COPYGROUP treats the group as a single entity.

COPYGRP can be used to copy a data set when either the input data set or the output data set, or both, are PDSE:

- PDSE to PDSE
- PDSE to PDS
- PDS to PDSE

COPYGROUP can be used to copy a data set when either the input data set or the output data set, or both, are PDS or PDSE:

PDSE to PDSE
PDSE to PDS
PDS to PDSE
PDS to PDS

For unloading groups:

Using COPYGRP: PDSE to PS
Using COPYGROUP: PDSE to PS, or PDS to PS

For loading groups:

Using COPYGRP: PS to PDSE
Using COPYGROUP: PS to PDSE, or PS to PDS

If neither data set is a PDSE, the request is treated as a COPY operation subject to the syntax requirements of COPYGRP.

When using the COPYGRP or COPYGROUP statement:

- All aliases in a group will be copied with the member or neither the aliases or the member in a group will be copied.
- There can be only one INDD per copy operation.
- You can use the SELECT statement to selectively copy members. Either the member name or an alias can be specified to copy the member and all of its aliases.
- Do not indicate replace (R) on the SELECT statement.
- The EXCLUDE statement is not supported.

COPYGROUP

The COPYGROUP statement has the same effect as the COPYGRP statement when either the input or the output data set or both are partitioned format, that is either PDS or PDSEs. The function of a COPYGROUP statement differs from COPYGRP only if both of the data sets are PDSs. COPYGROUP performs a full group copy operation when both data sets are PDSs. By contrast, a COPYGRP statement with two PDSs is the same as a COPY statement with those data sets.

To abbreviate a COPYGROUP statement, use CP. The abbreviation for COPYGRP remains CG.

When using the COPYGROUP statement:

- Member name pattern masking can be used, instead of a full member name specification, to select multiple member from the input data set.
- If member name pattern masking is used, the member rename option will not be honored. Member name pattern masking and member rename are mutually exclusive.

Replacing Program Objects

If the replace (R) option is indicated on the INDD parameter,

- The output data set members and their aliases will be replaced if they have the same member and alias names as the input data set's members and aliases.

Example:

The input data set has member A with alias B; the output data set has member A with alias B. The input data set's member and alias will replace the output data set member and alias.

- The copy will fail if a member's alias in the output data is the same as a differently named member's alias in the input data set.

Example:

The input data set has member A with alias B; the output data set has member C with alias B. The copy will fail because the alias B points to a member with a different name on the output data set.

- If the output data set's members and aliases do not match the input data set's members and aliases, then all of the input data set's members and aliases are copied.

Example:

The input data set has member A with alias B; the output data set has member C with alias D. After the copy, the output data set will contain A with alias B and C with alias D.

The EXCLUDE statement is not supported.

Compressing a Partitioned Data Set

A partitioned data set will contain unused areas (sometimes called gas) where a deleted member or the old version of an updated member once resided. This unused space is only reclaimed when a partitioned data set is copied to a new data set, or after a compress-in-place operation successfully completes. It has no meaning for a PDSE and is ignored if requested.

The simplest way to request a compress-in-place operation, is to specify the same ddname for both the OUTDD and INDD parameters of a COPY statement, without specifying the SELECT / EXCLUDE statement.

However, a compress is actually performed when both the input and output is the same data set on the same volume. For example, this job step will compress data set Pacanowska:

```
//COMPRESS EXEC      PGM=IEBCOPY
//A          DD      DSN= 'Pacanowska',DISP=OLD
//B          DD      DSN= 'Pacanowska',DISP=OLD
//SYSPRINT   DD      SYSOUT=*
//SYSIN      DD      *
              COPY    OUTDD=B,INDD=A
```

If multiple entries are made on the INDD statement, a compress-in-place occurs when any of the input ddnames matches the OUTDD name. The compress operation is performed in the same relative order as the ddnames in the INDD list.

For example, consider the COPY statement:

```
COPY    OUTDD=B,INDD=(A,B,C,B)
```

- The data set for ddname A is copied to ddname B
- The data set B is compressed
- ddname C is copied to ddname B
- The data set B is compressed again.

It is a good idea to make a copy of the data set that you intend to compress-in-place before you actually do so. You can use the same execution of IEBCOPY to do both, and a following job step could delete the backup copy if the IEBCOPY job step ends with a return code of 0.

Attention: A partitioned data set can be destroyed if IEBCOPY is interrupted during processing, for example, by a power failure,abend, TSO attention, or I/O error. Keep a backup of the partitioned data set until the successful completion of a compress-in-place.

Attention: Do not compress a partitioned data set currently being used by more than one user. If you do, the other users will see the data set as damaged, even though it is not. If any of these other users update the partitioned data set after you compress it, then their update will actually damage the partitioned data set.

Processing Considerations for Compress

- If you try to perform a compress-in-place on a PDSE, IEBCOPY will ignore your request and continue processing with the next control statement.
- A compress-in-place does not release extents assigned to the data set.

- During a compress operation, you cannot include, exclude, or rename selected members.
- You may not change any data set DCB parameters, such as block size, when compressing a data set.

Altering Load Modules

ALTERMOD is designed as a one-time update operation against load modules from old systems. You can use IEBCOPY to update partitioned data set load modules that were written by a linkage editor prior to MVS/370, so that they will load faster. Load modules processed with the linkage editor in MVS/370 and subsequent versions of MVS and DFSMS do not require alterations, nor do program objects in a PDSE.

ALTERMOD will place correct relocation dictionary (RLD) counts and segment block counts into control records inside the module. ALTERMOD performs this update without making a new copy of the load module. It can be used to alter modules that might have erroneous RLD counts. Examples of modules that might have erroneous RLD counts are modules that were created by a program other than the linkage editor or copied by a program other than IEBCOPY.

The ALTERMOD statement will not function under these conditions:

- The load modules are in scatter-load format or link edited with the noneditable (NE) attribute.
- The data set is a PDSE. (It is ignored.)

When you use a SELECT statement to identify members to be processed by ALTERMOD, you cannot rename them.

Copying and Reblocking Load Modules

The COPYMOD statement lets you COPY and reblock the load modules to a block size appropriate for the device to which you are copying the data set.

The text records, relocation dictionary (RLD)/control records, and note list records of overlay load modules will be rebuilt when you use COPYMOD. Other records such as SYM and CESD records will be copied without any changes. The load modules processed by COPYMOD can be link edited again.

- Load modules in page-aligned format are copied without reblocking, as if the operation was COPY not COPYMOD, and the functions of ALTERMOD are performed against the copy that was made.
- Load modules in scatter-load format and modules that were link-edited with the noneditable (NE) attribute will be copied, but not reblocked or altered.
- Members that are not recognized as load modules will be copied, but not reblocked or altered.
- Load modules that have the downward compatible (DC) linkage editor attribute are reblocked to a maximum block size of 1024 (1K) regardless of the value specified on the MINBLK or MAXBLK parameter.

The block size in the output data set label is increased by COPYMOD as needed to match the MAXBLK value.

COPYMOD does not write records longer than the output data set block size. However, if COPYMOD cannot process a member, and COPY is used instead, COPY will copy all records, including those records longer than the output data set block size.

The reblocking function of COPYMOD lets you specify the following block sizes:

- A maximum block size for compatibility with other systems or programs
- A minimum block size to specify the smallest block that should be written on the end of a track

IEBCOPY will determine the amount of space remaining on a track before assigning a size to the next block to be written. If this amount is smaller than the output block size, IEBCOPY will try to determine if a smaller block can be written to use the remaining space on the track. The maximum block size produced by the COPYMOD function is 32760 bytes.

Changed COPYMOD operation

Before MVS/DFP Version 3, Release 2, the default MAXBLK size for COPYMOD was 32760. The intention was to make text blocks as long as possible to reduce the number of text blocks read to fetch the module. This often resulted in physical records created in the data set which were longer than the BLKSIZE in the data set label. (These blocks are called *fat blocks*.) This condition leads to I/O errors or a violation of data integrity.

Starting with Version 3, Release 2, the default MAXBLK value is the data set block size, not 32760. Further, if a MAXBLK value is specified that is larger than the block size in the data set label, then the value in the data set label is increased to the specified MAXBLK value.

How IEBCOPY Uses Virtual Storage for Tables and Buffers

Starting with IEBCOPY ESCON® Performance Updates the minimum work area size is about 208K. If your job has a small REGION size, you might have to increase it.

The recommended minimum REGION sizes for IEBCOPY jobs are 1M when only partitioned data sets are copied, and 2M when a PDSE is copied.

The WORK=nnn parameter in the JCL EXEC PARM field controls how much space IEBCOPY requests for a work area. The REGION size must exceed the work size plus the program size before an increase in the value of the work parameter will have an effect.

From this work area comes tables, buffers, and storage for partitioned data set directories. When there is not enough work area, the partitioned data set directories spill to SYSUT3 and SYSUT4. If more storage is still needed, IEBCOPY will stop with a message. Larger WORK and REGION values allow larger directories to be processed without opening the spill data sets assigned to SYSUT3 and SYSUT4.

Large WORK and REGION sizes should be accompanied by a SIZE parameter when small data sets are copied. This is to prevent the buffers from becoming too large and causing a degradation in performance or a shortage in real storage.

How IEBCOPY Allocates Tables and Buffers

1. IEBCOPY obtains as large a work area as possible, up to the value of the WORK parameter.
2. An initial minimum sized buffer and channel program construction area is allocated from the work area.

The size of this area is approximately 4 times the size of the largest input or output device track, plus about 3%.

If IEBCOPY cannot allocate a minimum sized work area, it will stop with a message.

3. IEBCOPY processes control statements, reads the partitioned data set directories, and builds a table of members to be copied. When the work area cannot hold the table and all directories, the directory entries are spilled to SYSUT3 and SYSUT4 to allow the table to grow. When no room remains and all directories are spilled, then IEBCOPY ends with a message.
4. If enough storage remains unused in the work area, it is used as a second buffer.

If a SIZE= parameter is specified, the size of the second buffer is limited so that the total size of both buffers does not exceed the specified value. If the SIZE= value will not allow a minimum sized second buffer, it is not allocated.

Tip: When SIZE=999999 (or any number that is a few thousand less than the WORK= value) is coded, IEBCOPY might be able to allocate buffers in the work area but not have enough room remaining for tables. If this happens, increase the REGION= and WORK= values, or remove the SIZE= parameter.

When data sets with huge directories are copied, make the largest amount of virtual storage available to retain directory information. Specify WORK=8M (or another large value) and a correspondingly large REGION.

When data sets with small directories are copied with large work area sizes, the second I/O buffer can become very large (megabytes) and cause real storage shortages. This could result in increased system paging and system sluggishness, because most of the buffer is backed by real frames which are fixed for duration of the I/O. In this case, specify SIZE=1M or a smaller value to limit the amount of storage used for buffers, but allow lots of storage to be used for directory information.

Avoiding the Need to Supply Control Statements

When the SYSIN DD statement is a DD DUMMY, points to an empty file, or is omitted, IEBCOPY will generate a control statement that allows you to run IEBCOPY without supplying a control statement data set for SYSIN.

The generated statement can take several forms, depending on what parameters are specified in the JCL EXEC PARM field.

1. When COMPRESS and REPLACE are not specified, the generated statement is: xxxxx OUTDD=SYSUT2,INDD=SYSUT1. The xxxxx will be ALTERMOD, COPYGROUP, COPYGRP or COPYMOD if you code one of those PARM options or a valid abbreviation. If you do not code one of them, the generated control statement will be COPY. The COPY, COPYGROUP, COPYGRP or COPYMOD will copy without replacing from the data set designated by the SYSUT1 DD statement to the data set designated by the SYSUT2 DD statement.
2. When COMPRESS is not specified but REPLACE is specified, the generated statement is: COPY OUTDD=SYSUT2,INDD=(SYSUT1,R) which will copy with replace from the data set designated by the SYSUT1 DD statement to the data set designated by the SYSUT2 DD statement.
3. When COMPRESS is specified, then the generated statement is: COPY OUTDD=SYSUT2,INDD=SYSUT2 which will compress in place the data set designated by the SYSUT2 DD statement.

When you specify any of the preceding parameters in the JCL EXEC PARM field, then IEBCOPY opens SYSIN and automatically generates a control statement.

Restrictions

To use IEBCOPY, see the following characteristics, rules, and restrictions:

- IEBCOPY *does* support VIO (virtual I/O) data sets.
- IEBCOPY uses the EXCP access method. Therefore:
 - Some common DCB parameters such as BUFNO are ignored.
- Starting in z/OS V1R13, IEBCOPY is no longer APF-authorized.
- IEBCOPY must not be loaded in supervisor state or in protection key zero. It is an application program that does not use the special system interfaces that are assumed for the system kernel running in supervisor state or in protection key zero.
- Variable spanned format record (VS or VBS) are not supported for a partitioned data set.
- Shared or in use data sets should not be compressed in place or updated unless the subject data set is made *nonsharable*. See [“Sharing data sets”](#) on page 7.
- When a PDSE is involved and only a small amount of virtual storage is available to the PDSE processing routines, then messages about the shortage might only appear on the console and not in the SYSPRINT data set.
- Load modules having the downward compatible (DC) linkage editor attribute is reblocked to a maximum block size of 1024 (1K) when encountered during COPYMOD processing, regardless of the number specified on the MINBLK and MAXBLK parameters.
- Reblocking cannot be performed if either the input or the output data set has:
 - Undefined format records
 - Keyed records
 - Track overflow records

- Note lists or user TTRNs

Or if compress-in-place is specified. (Load modules, with undefined record formats and note lists, might be reblocked using the COPYMOD statement.)

- The compress-in-place function cannot be performed for the following:
 - Unload data sets.
 - Data sets with track overflow records.
 - Data sets with keyed records.
 - Unmovable data sets.
 - PDSEs (request is ignored).
- PDSEs cannot contain members with note lists, keys, or track overflow. You cannot mix load modules and nonload modules in the same PDSE.
- Using OPTCD=W for any DASD might dramatically slow down a COPY operation. OPTCD is in the data set label, and therefore can be active when OPTCD is *not* coded on the DD statement.
- OPTCD=W is honored when coded in the JCL. OPTCD=W, if present in the data set label, is deleted from the label.

Note: If IEBCOPY copies a record which is physically longer than the block size of the output partitioned data set, message IEB175I (return code 4) is issued to warn you that the data set contains *fat blocks*, which are physical records that are created in the data set that are longer than the BLKSIZE in the data set label.

- COPYMOD size limits
 - The load modules used as input cannot have more than 60 CSECTS in a single text block.
 - Overlay load modules cannot have more than 255 segments.
- IEBCOPY user TTR limits
 - There are up to three user TTRN fields in the directory.
 - Only one of these fields might have n>0.
 - The maximum length of the note list record identified by the user TTRN with n>0 is 1291 bytes including any block and record descriptor word.
 - No TTRN fields in a note list record might have n>0.
- A load module from an unload data set cannot be reloaded into a PDSE as a program object. The load modules should be reloaded into a partitioned data set and then the partitioned data set should be copied to a PDSE to convert the unloaded load module into a program object.
- IEBCOPY size limits
 - The maximum number of renames allowed on all SELECT or EXCLUDE cards for one copy operation is $(2 * \text{max_trk}) / 16$, where "max_trk" is the cylinder length of the larger device used in the copy operation. For a 3380, this limit is about 5925. For a 3390 the limit is about 7050, and for a 9345, the limit is about 5800.
 - Do not use a PDSU block size smaller than the PDS block size +20.
 - SYSUT4 space must be a single contiguous extent.
 - A PDSE program object must have a block size of at least 4000 bytes.
- IEBCOPY specifies the REREAD or LEAVE parameters for the CLOSE macro. If you specify FREE=CLOSE in the JCL, the data set is not unallocated until the end of the job step, and informational message IEC988I is issued to indicate that the data sets are not unallocated during close.
- IEBCOPY can require significant amounts of virtual storage when the input data set (INDD) is a PDSE. The amount of private area extended virtual storage required when using a PDSE as the input data set is approximately one megabyte per two hundred PDSE members that are included. Since the PDSE copy or unload operation requires a sizable amount of private area extended virtual storage, large PDSE data set storage requirements might exceed the total amount of available private area storage.

The actual amount of private area virtual storage available varies based on local system configuration and loading requirements. Some PDSE copy operations might fail because of excessive virtual storage requirements.

Following are some estimates of storage requirements, for use as examples:

- If an INDD PDSE includes 500,000 members:

`500,000 members/200 per megabyte = 2,500; 2,500 X 1 megabyte = 2.6+ gigabytes`

- If an INDD PDSE includes 300,000 members:

`300,000 members/200 per megabyte = 1,500; 1,500 X 1 megabyte = 1.5+ gigabytes`

- Some PDS data members, for example in IMS ACBLIBs, have attributes that resemble load module attributes. Using COPYGROUP to copy such members from a PDS to a PDS might result in message IGW0155T. Instead, use COPYGRP and specify the member and alias names to be copied.

Input and Output

IEBCOPY uses the following input:

- A partitioned data set, or a PDSE, or unload data set that contains members to be copied, merged, altered, reblocked, loaded, or unloaded.
- An optional control data set that contains utility control statements. The control data set is required when:

There is more than one input or one output data set to be processed,
Designated members are to be selected or excluded, or
A load module library is to be altered or reblocked (ALTERMOD or COPYMOD is needed).

IEBCOPY produces the following output:

- Output data sets, which contain the copied, merged, altered, reblocked, or unloaded members. The output data set is either a new data set (from a copy, reblock, load, or unload) or an old data set (from a merge, compress-in-place, copy, alter, or load).
- A message data set, which lists control statements, IEBCOPY activities, and error messages, as applicable.

IEBCOPY might require:

- Optional spill data sets, which are temporary data sets used to provide space when not enough virtual storage is available for the input or output partitioned data set directories. These data sets are opened only when needed.

If IEBCOPY is invoked from an application program, you can dynamically allocate the data sets by issuing SVC 99 before calling IEBCOPY.

For IEBCOPY return codes, see [Appendix A, “Invoking Utility Programs from an Application Program,”](#) on page 315.

Control

IEBCOPY is controlled by JCL and utility control statements.

Job Control Statements

[Table 10 on page 55](#) shows the job control statements for IEBCOPY.

Table 10. Job control statements for IEBCOPY

Statement	Use
JOB	Starts the job.
EXEC	Starts IEBCOPY.
SYSMDUMP DD	Defines the dump data set when an abnormal end is issued.
SYSPRINT DD	Defines a sequential data set used for listing control statements and messages.
SYSUT1 or anyname1 DD	Defines a partitioned data set or unload data set for input. A partitioned data set must reside on DASD or be a VIO data set. The unload data set is a sequential data set created as the result of an unload operation and may reside on DASD or tape or any other device supported by the QSAM access method. The unload data set can be basic format, large format, extended format, compressed format or tape.
SYSUT2 or anyname2 DD	Defines a partitioned data set or unload data set for output. A partitioned data set must reside on DASD or be a VIO data set. The unload data set is a sequential data set that was created as the result of an unload operation and may reside on DASD or tape or any other device supported by the QSAM access method.
SYSUT3 DD	Defines a spill data set on a DASD or VIO device. SYSUT3 is used when there is no space in virtual storage for some or all of the current input data set directory entries.
SYSUT4 DD	Defines a spill data set on a DASD or VIO device. SYSUT4 is used when there is no space in virtual storage for the output data set directory.
SYSIN DD	Defines the optional control data set.

EXEC Statement

The syntax of the EXEC statement is:

Label	Statement	Parameters
//[stepname]	EXEC	[, PGM=IEBCOPY. [, REGION={n nK nM}] [, PARM=<parms>]

where:

PGM=IEBCOPY

specifies that you want to run the IEBCOPY program.

REGION={n|nK|nM}

specifies the amount of storage to be made available to IEBCOPY by the operating system. Specify 1M if you are only using partitioned data sets. If you are using any PDSE, then specify 2M.

Notes:

1. The number *n* can be any number of digits, and is specified in decimal. The K causes the number to be multiplied by 1024 bytes (1 kilobyte) and M causes the number to be multiplied by 1024K or 1048576.
2. Specifying a larger REGION size might not have an effect unless the WORK= value is also increased.
3. Specifying a REGION size on the JOB statement will override any REGION specified on the EXEC statement.

PARM=

You may specify any of the parameters in any order to IEBCOPY. Separate multiple parameters with a comma between each one.

ALTERMOD or AM

If you direct IEBCOPY to generate a control statement, specify this parameter to make the control statement perform an ALTERMOD operation instead of a COPY operation.

BYPASS_AUTH

Specify this parameter if you wish to direct IEBCOPY to bypass RACF authorization checking for a DASD sequential or partitioned (PDS or PDSE) data set. This option does not apply to tape, spooled data sets, work data sets, or data set password checking. If the BYPASS_AUTH option is specified, IEBCOPY must be invoked by an APF-authorized caller. If the parameter is passed from a non APF-authorized caller, abend 913-80 will be issued.

CMWA=nK

Specify this parameter to increase the COPYMOD work area size of 120K if larger load modules are being processed. This will be evident because message IEB1133E will be issued in this instance.

COMPRESS

If you direct IEBCOPY to generate a control statement, specify this parameter to make the control statement perform a compress-in-place operation instead of a COPY operation.

COPY or C

If you wish to direct IEBCOPY to generate a control statement, specify this parameter to make the control statement perform a COPY operation.

COPYGROUP or CP

If you wish to direct IEBCOPY to generate a control statement, specify this parameter to make the control statement perform a COPYGROUP operation.

COPYGRP or CG

If you wish to direct IEBCOPY to generate a control statement, specify this parameter to make the control statement perform a COPYGRP operation.

COPYMOD

If you wish to direct IEBCOPY to generate a control statement, specify this parameter to make the control statement perform a COPYMOD operation.

LC=n**LPP=n****LINECOUNT=n**

n is the number of lines, including headings, to print on each page of the SYSPRINT output listing. Default is 60.

LIST=NO**LIST=YES**

sets the default value for the LIST= operand when it is omitted from the COPY, COPYMOD, or ALTERMOD statement. Default is **LIST=YES**.

RC4NOREP

Use of this parameter will cause IEBCOPY to set a return code of X'04' when a module is not copied from the source data set to the target data set because REPLACE was not specified. When 'RC4NOREP' is specified, message IEB1067W will be issued for each module NOT copied due to the REPLACE option not being specified. Note: Message IEB1067W will be issued regardless of the LIST option requested.

REPLACE

If you wish to direct IEBCOPY to generate a control statement, specify this parameter to make the control statement perform a copy with replace operation.

SIZE={n|nK|nM}

specifies the maximum number of bytes of virtual storage that IEBCOPY may use as a buffer.

It is best to let IEBCOPY choose buffer sizes by not using this parameter.

The minimum buffer size is approximately 4 times the largest track size of the devices being used, plus about 3%. There is no maximum for this value, but IEBCOPY cannot use more than the quantity available in the work area.

The number *n* can be any number of digits, and is specified in decimal. The K causes the number to be multiplied by 1024 bytes (1 kilobyte) and M causes the number to be multiplied by 1024K or 1048576.

SPCLCMOD

specifies that when a member is found to be ineligible for COPYMOD, processing (for example, the module is page aligned) and the module is scheduled for processing as a normal COPY, the detection of a record larger than the blocksize of the target data set will cause the entire operation to be terminated (no fat blocks will be created).

If the COPY operation resulting from COPYMOD ineligibility is successful, the return code of the operation will be set to zero instead of four (normally return code is set to 4 to indicate that a COPY instead of a COPYMOD was done for the affected modules, if LIST=YES is in effect).

WORK={*n*|*n*K|*n*M|1M}

specifies the number of bytes of virtual storage to request for a work area to hold directory entries, internal tables, and I/O buffers. The default request will be for 1M. The actual amount obtained will not exceed the space available in the REGION.

The number *n* can be any number of digits, and is specified in decimal. The K causes the number to be multiplied by 1024 bytes (1 kilobyte) and M causes the number to be multiplied by 1024K or 1048576.

Related reading: For more information, see:

- [“Avoiding the Need to Supply Control Statements” on page 52,](#)
- [“How IEBCOPY Allocates Tables and Buffers” on page 51,](#)
- [“How IEBCOPY Uses Virtual Storage for Tables and Buffers” on page 51.](#)

SYSMDUMP DD Statement

The SYSMDUMP DD is an optional statement and can be specified if IEBCOPY abends and a dump is required for diagnosis.

The SYSMDUMP DD data set can be basic, large, extended, or compressed format or a member of a PDS or PDSE.

These are the valid DCB parameters you may specify:

RECFM=[FBS]

The record format is fixed blocked standard.

LRECL=

The minimum logical record length that you may specify is 4160 for fixed length.

SYSPRINT DD Statement

IEBCOPY writes a log of the control statements and its actions to the SYSPRINT DD statement. IEBCOPY proceeds if SYSPRINT is unusable and issues summary messages to the console.

You may assign SYSPRINT to SYSOUT or to any QSAM data set. These are the valid DCB parameters you may specify:

DSORG=PS

The output always has sequential organization.

RECFM=[F|FB|V|VB|FA|FBA|VA|VBA]

The record format may be fixed, fixed blocked, and variable or variable blocked. Any of these record formats can be specified to include ISO/ANSI control characters (for example, VBA instead of VB).

LRECL=

The minimum logical record length that you may specify is 60 for fixed length, or 64 for variable length. The maximum is 250.

BLKSIZE=

If you are using fixed blocked records, the block size may be any multiple of the logical record length. If you are using variable or variable blocked records, the block size must be a minimum of the logical record length plus four. The block size cannot exceed 32760 bytes.

If you do not specify anything for DCB parameters, and the data set label has no DCB parameters, IEBCOPY will choose RECFM=FBA, LRECL=121, and a block size. When the output device is DASD or a standard labeled tape, the system will determine the block size to be used. When the output device is not DASD or a standard labeled tape, IEBCOPY will use the largest value that will work for the specific device, adjusted as needed for the RECFM.

If you specify any parameters other than those that IEBCOPY would choose, IEBCOPY will adapt to them. If you omit RECFM and specify an LRECL other than 121, then IEBCOPY will set RECFM=VBA and it will set an optimal BLKSIZE.

Tip: Giving LRECL a value that is 1-byte less than the width of your TSO terminal will allow you to view entire records from a listing without scrolling left or right.

SYSUT1 (anyname1) and SYSUT2 (anyname2) DD Statements

DD statements are required for input and output data sets. There must be one DD statement for each unique data set used in the job step. You must specify a unique output DD statement (for the unload data set) for every input data set that you unload, for example you cannot append two unload data sets to the same output data set. You cannot unload multiple data sets to the same unload data set.

Data sets that are used as input data sets in one copy operation can be used as output data sets in another copy operation.

Input data sets cannot be concatenated together on the same DD statement.

Fixed or variable records can be reblocked, but you cannot convert fixed format records into variable format records, or variable format records into fixed format records. Reblocking or deblocking is performed automatically when the block size of the input data set is not equal to the block size of the output data set.

For variable format data sets, the output LRECL must be greater than or equal to the input LRECL. For fixed format data sets the input and output LRECL must match. For undefined format data sets, the output BLKSIZE must be greater than or equal to the input BLKSIZE, and any LRECL is ignored.

A PDSE might require somewhat more space than a partitioned data set requires to hold the same data. When copying from a partitioned data set to a PDSE, IEBCOPY will override a secondary space allocation of zero for the output PDSE, rather than stop the job. The copy will obtain any additional space it requires in one unit increments of the primary allocation (tracks, cylinders, or blocks). The secondary space quantity in the data set label will not be changed.

When IEBCOPY must supply DCB parameters for the output data set, it will use the corresponding values from the input data set. In particular, system determined block size will not be used for the output data set block size, nor will the system determined block size or reblockable flags be set in the output data set label.

You should allow IEBCOPY to choose the DCB parameters for an unload data set. If you must specify a block size, specify a value that is at least 20 bytes greater than the input data set block size. (This avoids creating spanned records in the unload data set.)

OPTCD=W may be used to request write verification of the data written and causes DASD cached controllers (for instance, 3880-21, 3990-3) to perform a "write through" operation.

Starting with ESCON support, OPTCD=W also causes a different set of channel programs to be used for writing to ECKD™ capable DASD and for reading from all DASD. These programs could run much slower than the default ones; therefore, there is a significant performance cost for using OPTCD=W.

The write verification requested by OPTCD=W is usually unnecessary, because extensive error recovery occurs in both hardware and error recovery procedures (ERP) used by the MVS I/O supervisor. OPTCD=W does not cause the data read from the device to be compared with the data in virtual storage and cannot detect data garbled in transmission from virtual storage to the device control unit. OPTCD=W causes the control unit to read data from the device (after it has been written) to validate the error checking codes stored with the data.

Related reading: For more information, see [“IEBCOPY Unload Data Set DCB Parameters”](#) on page 43 .

SYSUT3 and SYSUT4 DD statements

In most uses of IEBCOPY, you do not need to provide space on spill data sets (SYSUT3 and SYSUT4). If IEBCOPY is running in a region size of 2M or more, neither of the spill data sets are needed if the output data set will have fewer than 1600 directory blocks.

In order to conserve space on DASD, you can use VIO for these data sets. However, it is more efficient to increase the region size (and WORK=parameter value) and not use the SYSUT3 and SYSUT4 data sets at all. You cannot use multivolume data sets for these data sets.

The space required for SYSUT3 depends on the number of members to be copied or loaded. The space to be allocated for SYSUT4 must be equal to or greater than the number of blocks allocated to the largest output partitioned data set directory in the IEBCOPY jobstep. Use a block size of 80 to calculate space requirements.

The space required depends on the number of directory blocks to be written to the output data set. SYSUT4 is only used if more than one data set is specified on a given INDD list. The data set contains one directory block per data block. Use a block size of 256 and a key length of 8 to calculate space requirements.

Requirement: SYSUT4 must be in a single contiguous extent, because SYSUT4 will contain a copy of the output partitioned data set directory. The design of data management requires that a partitioned data set directory be within a single extent.

IEBCOPY ignores all DCB information that is specified for SYSUT3 or SYSUT4.

SYSIN DD Statement

The SYSIN DD statement is optional. If it is omitted, or a DUMMY data set, or an empty data set, IEBCOPY will generate a control statement using options in the PARM field.

SYSIN will not be opened if COPY, COPYMOD, REPLACE, or COMPRESS appears in the PARM field.

If you are copying, loading, or unloading a *single* data set, or wish to compress a data set in place, you can avoid using utility control statements by using SYSUT1 for the input data set and SYSUT2 for the output data set.

Either fixed, variable, or undefined record format is acceptable, as is any BLKSIZE and LRECL consistent with the record format and each other. If the record format indicates carriage controls, the carriage control character in each record is ignored.

Sequence numbers are optional. IEBCOPY will look for sequence numbers (8 digits long) at the front of variable format records and at the end of fixed format records. If IEBCOPY finds them in the first control statement, it will ignore those columns in all control statements.

Related reading: For more information, see [“Avoiding the Need to Supply Control Statements”](#) on page 52 .

Utility Control Statements

IEBCOPY is controlled by the utility control statements in [Table 11](#) on page 60.

Table 11. IEBCOPY utility control statements. The minor statements (SELECT or EXCLUDE) can follow each major statement to restrict the scope of the major statements.

Statement	Use
Major Statements	
ALTERMOD	Indicates the beginning of an alter-in-place operation for load modules.
COPY	Indicates the beginning of a COPY operation.
COPYGRP	Indicates the beginning of a COPYGRP operation.
COPYGROUP	Indicates the beginning of a COPYGROUP operation.
COPYMOD	Indicates the beginning of a copy and load module reblock operation.
INDD=	Indicates the beginning of another copy step.

Table 12. IEBCOPY utility control statements (continued). The minor statements (SELECT or EXCLUDE) can follow each major statement to restrict the scope of the major statements.

Statement	Use
Minor Statements	
EXCLUDE	Specifies members in the input data set to be excluded from the copy step.
SELECT	Specifies which members in the input data set are to be copied.

COPY will accept the first letter as an abbreviation for all its key words, except MINBLK. COPYMOD may be abbreviated CM. COPYGRP may be abbreviated CG. COPYGROUP may be abbreviated CP.

To continue the copy control statement, stop at a comma. Put any nonblank character in column 72 and start in column 16 on the next record.

To make a comment statement, place an asterisk (*) in the left column where the label field goes. The record will be printed, then ignored. You may also place a comment on any control statement which also has an operand. Leave 1 or more spaces after the operand, and then start your comment.

- To request a *COPY* operation, specify partitioned data sets as input and output.
- To request an *UNLOAD* operation, specify a partitioned input data set and a sequential output data set.
- To request a *LOAD* operation, specify a sequential input data set and a partitioned output data set.

IEBCOPY uses a copy operation/copy step concept. A copy operation starts with a COPY, COPYGRP, COPYGROUP, COPYMOD, or ALTERMOD statement, and continues until another COPY, COPYGRP, COPYGROUP, COPYMOD, or ALTERMOD statement is found, or the end of the control data set is found. Within each copy operation, one or more copy steps are present. Any INDD statement directly following a SELECT or EXCLUDE statement marks the beginning of the next copy step and the end of the preceding copy step within the copy operation. If such an INDD statement cannot be found in the copy operation, the copy operation will consist of only one copy step.

Table 13 on page 60 shows the copy operation/copy step concept. Two copy operations are shown in the figure. The first begins with the statement named COOPER1, and the second begins with the statement named COOPER2.

Table 13. Multiple copy operations within a job step

Step	Label	Command	Parameters
First Copy Operation			

Table 13. Multiple copy operations within a job step (continued)

Step	Label	Command	Parameters
STEP 1	COPOPER1	COPY	OUTDD=AA INDD=ZZ INDD=(BB,CC) INDD=DD INDD=EE
		SELECT	MEMBER=(MEMA,MEMB)
		SELECT	MEMBER=(MEMC)
STEP 2			INDD=GG INDD=HH
		EXCLUDE	MEMBER=(MEMD,MEMH)
Second Copy Operation			
STEP 1	COPOPER2	COPY	OUTDD=YY INDD=(MM,PP) LIST=NO
		SELECT	MEMBER=MEMB
STEP 2			INDD=KK INDD=(LL,NN)

The first copy operation shown in Table 13 on page 60 is copying two groups of members. The first begins with the COPY statement and continues through the two SELECT statements. The second begins with the first INDD statement following the two SELECT statements and continues through the EXCLUDE statement preceding the second COPY statement.

The second copy operation has two steps. The first step begins with the COPY statement and continues through the SELECT statement. The second begins with the INDD statement immediately following the SELECT statement.

ALTERMOD Statement

The ALTERMOD statement is required to alter load modules in place. ALTERMOD will only work with a partitioned data set, not a PDSE.

The syntax of the ALTERMOD statement is:

Label	Statement	Parameters
[<i>label</i>]	ALTERMOD	OUTDD= <i>DDname</i> [, LIST={YES NO}]

where:

OUTDD=*DDname*

specifies the ddname of the partitioned data set that is to be altered.

LIST={YES|NO}

specifies whether the names of the altered members are to be listed in the SYSPRINT data set. When this parameter is omitted, the default from the EXEC PARM field applies.

COPY Statement

Use the COPY statement to begin one or more copy, unload, or load operations. Any number of operations can follow a single COPY statement; any number of COPY statements can appear within a single job step.

The syntax of the COPY statement is:

Label	Statement	Parameters
[<i>label</i>]	COPY	OUTDD= <i>DDname</i> , INDD=[([{ <i>DDname</i> (<i>DDname</i> , R) } [, ...] [])] [, LIST={YES NO}]

where:

OUTDD=*DDname*

specifies the name of a DD statement that locates the output data set.

INDD=[([{*DDname* | (*DDname*, R) } [, ...] [])]

specifies the names of DD statements that locate the input data sets.

When an INDD= appears in a record by itself (that is, not with a COPY keyword), it functions as a control statement and begins a new step in the current COPY operation.

These values can be coded:

DDname

the ddname of the DD statement for the input data set. For an unload operation, only one ddname should be specified per COPY statement. If more than one ddname is specified for a copy or load operation, the input data sets are processed in the same sequence as the ddnames are specified.

R

specifies that all members to be copied or loaded from this input data set are to replace any identically named members on the output partitioned data set. (In addition, members whose names are not on the output data set are copied or loaded as usual.) When this option is specified, the ddname and the R parameter must be enclosed in a set of parentheses; if it is specified with the first ddname in INDD, the entire field, exclusive of the INDD parameter, must be enclosed in a second set of parentheses.

LIST={YES|NO}

specifies whether the names of copied members are to be listed in the SYSPRINT data set at the end of each input data set. When this parameter is omitted, the default from the EXEC PARM field applies.

Usage Notes for COPY

The control statement operation and keyword parameters can be abbreviated to their first letters; for example, COPY can be abbreviated to C and OUTDD can be abbreviated to O.

If there are no keywords other than OUTDD on the COPY record, comments may not be placed on the record.

You can use COPY to copy *and* reblock data sets with fixed-blocked or variable-blocked records. You cannot use COPY to reblock undefined records or load module libraries. Instead, use COPYMOD to copy *and* reblock a load library.

Only one INDD and one OUTDD keyword may be placed on a single record. OUTDD must appear on the COPY statement. When INDD appears on a separate record, no other operands may be specified on that record, and INDD is not preceded by a comma.

A COPY statement must precede SELECT or EXCLUDE statements when members are selected for or excluded from a copy, unload, or load step. In addition, if an input ddname is specified on a separate INDD statement, it must follow the COPY statement and precede the SELECT or EXCLUDE statement which apply to it.

COPYGRP statement

Use the COPYGRP statement to begin a group copy, unload, or load. A group consists of a member and all of its aliases. COPYGRP treats the group as a single entity.

The syntax of the COPYGRP statement is:

Label	Statement	Parameters
[<i>label</i>]	COPYGRP	OUTDD= <i>DDname</i> , INDD={ <i>DDname</i> (<i>DDname</i> , <i>R</i>))} [Y, LIST={YES NO}] [Y, GENS={ALL NONE}]

Where:

OUTDD=*DDname*

Specifies the name of a DD statement that locates the output data set.

INDD={*DDname*| (*DDname*,*R*))}

Specifies the name of a DD statement that locates the input data set.

Restriction: Multiple INDD statements are not allowed for COPYGRP.

These values can be coded:

DDname

Specifies the ddname, which is specified on a DD statement, of an input data set.

R

Specifies that a group to be copied or loaded from this input data set is to replace a group in the output data set. When this option is specified, the ddname and the R parameter must be enclosed in a set of parentheses; and the entire field, except the INDD parameter, must be enclosed in a second set of parentheses.

LIST={YES|NO}

Specifies whether the names of copied members are to be listed in the SYSPRINT data set at the end of each input data set. When this parameter is omitted, the default from the EXEC PARM field applies.

GENS={ALL|NONE}

Indicates whether the member generations of a Version 2 PDSE are copied. This keyword is valid only when copying from a PDSE V2 with member generations to another.

GENS=ALL

Specifies that all members and generations to be copied from this input PDSE data set are to replace any identically named members on the output PDSE data set.

GENS=NONE

Specifies not to copy the member generations of the input PDSE data set.

Usage Notes for COPYGRP

The control statement can be abbreviated to CG. Only one input ddname can be used per COPYGRP control statement.

The INDD and OUTDD keyword must appear on the COPYGRP statement.

A COPYGRP statement must precede the SELECT statement when members are selected for a copy, unload, or load step. A SELECT statement following a COPYGRP cannot contain the R (replace) parameter.

An EXCLUDE statement cannot follow a COPYGRP statement.

For COPYGRP, either or both of the input and output data sets must be PDSEs; if both are PDSEs, no group copy function occurs.

COPYGROUP statement

Use the COPYGROUP statement to begin a group copy, unload, or load. A group consists of a member and all of its aliases. COPYGROUP treats the group as a single entity.

The syntax of the COPYGROUP statement is:

Label	Statement	Parameters
[label]	COPYGROUP	OUTDD=DDname , INDD={DDname ((DDname,R))} [Y,LIST={YES NO}] [Y,GENS={ALL NONE}]

Where:

OUTDD=DDname

Specifies the name of a DD statement that locates the output data set.

INDD={DDname|((DDname,R))}

Specifies the name of a DD statement that locates the input data set.

Restriction: Multiple INDD statements are not allowed for COPYGROUP.

These values can be coded:

DDname

Specifies the ddname, which is specified on a DD statement, of an input data set.

R

Specifies that a group to be copied or loaded from this input data set is to replace a group in the output data set. When this option is specified, the ddname and the R parameter must be enclosed in a set of parentheses; and the entire field, except the INDD parameter, must be enclosed in a second set of parentheses.

LIST={YES|NO}

Specifies whether the names of copied members are to be listed in the SYSPRINT data set at the end of each input data set. When this parameter is omitted, the default from the EXEC PARM field applies.

GENS={ALL|NONE}

Indicates whether the member generations of a Version 2 PDSE are copied. This keyword is valid only when copying from a PDSE V2 with member generations to another.

GENS=ALL

Specifies that all members and generations to be copied from this input PDSE data set are to replace any identically named members on the output PDSE data set.

GENS=NONE

Specifies not to copy the member generations of the input PDSE data set.

Usage Notes for COPYGROUP

The control statement can be abbreviated to CP.

Only one input ddname can be used per COPYGROUP control statement. The INDD and OUTDD keyword must appear on the COPYGROUP statement.

A COPYGROUP statement must precede the SELECT statement when members are selected for a copy, unload, or load step. A SELECT statement following a COPYGROUP cannot contain the R (replace) parameter.

An EXCLUDE statement cannot follow a COPYGROUP statement.

COPYGROUP cannot be used to load to a PDS or unload from a PDS.

When the INDD and OUTDD data sets are both PDSs, COPYGROUP can only be used if the INDD and OUTDD data sets have the same blocksize.

COPYMOD Statement

Use the COPYMOD statement to copy load module libraries that you want to reblock. The output data set must be partitioned. The input data set may be a partitioned or a sequential data set created by an unload operation. If the input to COPYMOD is not a valid load module, COPYMOD functions as a COPY.

The syntax of the COPYMOD statement is:

Label	Statement	Parameters
[label]	COPYMOD	OUTDD=DDname , INDD=([{DDname (DDname,R) } [, ...] []) [, MAXBLK={nnnnn nnK} [, MINBLK={nnnnn nnK} [, LIST={YES NO}

where:

OUTDD=DDname

specifies the name of a DD statement that locates the output data set.

INDD=([{DDname| (DDname,R) } [, ...] [])

specifies the names of DD statements that locate the input data sets.

When an INDD= appears in a record by itself (that is, not with a COPY keyword), it functions as a control statement and begins a new step in the current copy operation.

These values can be coded:

DDname

specifies the ddname, which is specified on a DD statement, of an input data set which is a load module library.

R

specifies that all members to be copied from this input data set are to replace any identically named members on the output load module library. (In addition, members whose names are not on the output data set are copied as usual.) When this option is specified, the ddname and the R parameter must be enclosed in a set of parentheses; if it is specified with the first ddname in INDD, the entire field, exclusive of the INDD parameter, must be enclosed in a second set of parentheses.

MAXBLK={nnnnn|nnK}

specifies the maximum block size for records in the output partitioned data set. MAXBLK is normally used to specify a smaller block size than the default, in order to make the record length of the data set compatible with different devices or programs.

The *nnnnn* value is specified as a decimal number; K indicates that the *nn* value is to be multiplied by 1024 bytes.

MAXBLK may be specified with or without MINBLK. Specifying a MAXBLK that is larger than the blocksize of the target data set will change the data set blocksize to the MAXBLK value.

Default: The COPYMOD MAXBLK parameter defaults to the output data set block size. If the output data set block size is zero, it defaults to the input data set block size.

MINBLK={nnnnn|nnK}

specifies the minimum block size for records in the output partitioned data set. MINBLK specifies the smallest block that should be written on the end of a track.

The MINBLK keyword is provided for compatibility with earlier MVS releases in which a larger, less-than-track-size MINBLK value could enhance program fetch performance for the module. Under normal circumstances, MINBLK should not be specified.

The *nnnnn* value is specified as a decimal number; K indicates that the *nn* value is to be multiplied by 1024 bytes.

MINBLK may be specified with or without MAXBLK.

Default: 1K (1024). If a value greater than MAXBLK is specified, MINBLK is set to the MAXBLK value actually used (whether specified or defaulted). If a value less than 1K is specified, MINBLK is set to 1K.

LIST={YES|NO}

specifies whether the names of copied members are to be listed in the SYSPRINT data set at the end of each input data set. When this parameter is omitted, the default from the EXEC PARM field applies.

INDD=Statement

In addition, when INDD, a COPY statement parameter, appears on a record other than the COPY statement, it is referred to as an INDD statement. It functions as a control statement in this context.

If one or more INDD statements are immediately followed by end of file or another COPY or COPYMOD or ALTERMOD statement, a full copy, unload, or load operation is completed using the most recent previously specified output data set.

EXCLUDE Statement

The EXCLUDE statement specifies members to be excluded from the copy, unload, or load step. All members in the input data set except those specified on each EXCLUDE statement are included in the operation. More than one EXCLUDE statement may be used in succession, in which case, the second and subsequent statements are treated as a continuation of the first.

The EXCLUDE statement must follow either a COPY statement, an ALTERMOD, a COPYMOD statement, or one or more INDD= statements. An EXCLUDE statement cannot appear with a SELECT statement in the same copy, unload, or load step. The EXCLUDE statement cannot be used with a compress-in-place or COPYGRP.

When using COPYGROUP and a SELECT MEMBER statement that uses pattern masking has been provided, an EXCLUDE statement with member name pattern masking may also be specified. When EXCLUDE MEMBER pattern masking is specified, the members excluded from the COPYGROUP operation will be a subset of those members that were selected for inclusion by the SELECT MEMBER pattern mask or masks.

If neither SELECT nor EXCLUDE is specified, the entire data set is copied (a "full copy").

The syntax of the EXCLUDE statement is:

Label	Statement	Parameters
[<i>label</i> !]	EXCLUDE	MEMBER=[(<i>name1</i> [, <i>name2</i>] [, . . .] [])]

where:

MEMBER=[(*name1* [, *name2*] [, . . .] [])]

specifies members on the input data set that are not to be copied, unloaded, or loaded to the output data set. When using COPYGROUP and a SELECT MEMBER statement that uses pattern masking has been provided, you can also provide a member pattern mask of members to be excluded from the input data set. The members are not deleted from the input data set.

The values that can be coded are:

name or filter pattern mask

identifies a specific member to be processed. All names and new names specified in one copy step must be unique. You cannot duplicate either old names, or new names, or both, under any circumstances. You cannot specify a name that is more than eight characters in length.

When using COPYGROUP and a SELECT statement that uses member name pattern masking, you can specify an EXCLUDE MEMBER= member pattern mask of members to be excluded from the input data set. The following rules and examples apply to member name filter pattern masking on the EXCLUDE statement:

- Pattern masking characters are asterisk (*) and percent sign (%).
- The asterisk character is used in the filter pattern mask to indicate that any characters within the member name that follow the pattern mask are acceptable.
- The percent character is used in the filter pattern mask to indicate that any character within the member name in a particular position is acceptable.
- The pattern must be part of a COPYGROUP operation where a SELECT statement that has specified MEMBER= with filter pattern masking is also specified.
- The pattern must be part of a EXCLUDE statement.
- Filtering will only be performed with primary member names.

See [Table 14 on page 69](#). for examples of member name filter patterns.

The control statement operation and keyword parameters can be abbreviated to their first letters; EXCLUDE can be abbreviated to E and MEMBER can be abbreviated to M.

Each member name on an EXCLUDE statement must be unique.

SELECT Statement

The SELECT statement specifies members to be selected from input data sets to be altered, copied, loaded, or unloaded to an output data set. This statement is also used to rename or replace selected members on the output data set. More than one SELECT statement may be used in succession, in which case the second and subsequent statements are treated as a continuation of the first.

The SELECT statement must follow either a COPY statement, a COPYGRP statement, a COPYMOD statement, or one or more INDD statements. A SELECT statement cannot appear with an EXCLUDE statement in the same copy, unload, or load step, and it cannot be used with a compress-in-place function. When both the INDD and OUTDD statements are pointing to the same data set, the members in the SELECT statement are replaced or copied to a new name within the same data set, based on the replace and newname options.

When a selected member is found on an input data set, it is not searched for again in this copy step.

Each member name on a SELECT statement must be unique.

A selected member will not replace an identically named member in the output data set unless the replace option is specified on either the data set or member level. In addition, unless the replace option

is specified, a renamed member will not replace a member in the output data set that has the same new name as the renamed member.

The syntax of the SELECT statement is:

Label	Statement	Parameters
[<i>label</i>]	SELECT	MEMBER=({ <i>name1</i> (<i>name1,newname1</i> [, <i>R</i>]) (<i>name1</i> , , <i>R</i>) } [, { <i>name2</i> (<i>name2,newname2</i> [, <i>R</i>]) (<i>name2</i> , , <i>R</i>) }] [, . . .])

where:

MEMBER=({*name*|(*name, newname*[,*R*])|(*name*,,*R*)} [,...])

specifies the members to be selected from the input data set.

To rename a member, specify the old name of the member, followed by the new name and, optionally, the R (replace) parameter. This group must then be enclosed in parentheses. If member name pattern masking is being used to select the members from the input data set, then the rename option is not supported.

To replace a member, specify the name of the member and the R parameter. Two commas must separate the R from the member name, and the group must be enclosed in parentheses.

When any option within parentheses is specified anywhere in the MEMBER field, the entire field, exclusive of the MEMBER keyword, must be enclosed in a second set of parentheses.

The values that can be coded are:

name or filter pattern mask

identifies a specific member to be processed. All names and new names specified in one copy step must be unique. You cannot duplicate either old names, or new names, or both, under any circumstances. You cannot rename A to B and B to C, because B will appear twice. You cannot specify a name that is more than eight characters in length.

When the COPYGROUP statement is used, the member name can either identify a specific member to be processed, or a member name pattern mask where multiple names may meet the name pattern criteria. Member name pattern masking cannot be used with *newname*. See [“SELECT Member name filter pattern masking” on page 69](#) for rules and examples of member name pattern masking.

newname

specifies a new name for a selected member. Member names can consist of A - Z, 0 - 9, or \$ # @ _ } \, or {, and cannot be more than eight characters in length. The member is copied, unloaded, or loaded to the output data set using its new name. If the name already appears on the output partitioned data set or PDSE, the member is not copied unless replacement is also specified. *Newname* cannot be specified with ALTERMOD.

R

specifies that the input member is to replace any identically named member that exists on the output data set. If the input member's name is identical to any output member's alias name, the name will refer to the new member and not the old member.

R may not be coded with ALTERMOD or COPYGRP.

The control statement operation and keyword parameters can be abbreviated to their first letters; for example, SELECT can be abbreviated to S and MEMBER can be abbreviated to M.

Related reading: For a description of replacing members, see [“Replacing Members in a Data Set” on page 46](#).

SELECT Member name filter pattern masking

The following rules and examples apply to member name filter pattern masking on the SELECT statement.

- The pattern masking characters are asterisk (*) and percent sign (%).
- The asterisk character is used in the filter pattern mask to indicate that any characters within the member name that follow the pattern mask are acceptable.
- The percent character is used in the filter pattern mask to indicate that any character within the member name in a particular position is acceptable.
- The pattern must be part of a COPYGROUP operation.
- The pattern must be part of a SELECT statement.
- Filtering will only be performed with primary member names.
- Alias names in a SELECT statement will be ignored and not copied.

Examples of member name filter pattern usage appear in [Table 14 on page 69](#).

Table 14. Member name filter pattern examples			
Member name pattern	Description	Examples of member names that would match	Examples of member names that would NOT match
*	Any member name	ABC, A12#\$F	None
CD*	Member names starting with CD	CDEFGH	ABCD
*CD	Member names ending with CD	ABCD	CDEFGH
CD%	Three-character member names starting with CD	CD\$	CD, CDEFG
%CD	Member names with second character C and third character D	BCDE, \$CD	CD, CDEFG
%CD*	Member names with second character C and third character D	BCD, BCDE, \$CD	BBCDE, CD
*CD%	Member names with second to last character D and third to last character C	CDE, BCDE, BBCDE	CD
AB%C	Member names that have the substring AB, followed by any one character, followed by the substring C	ABKC, KKABKCKK	ABC
%	Any one-character member name	A, B, C	ABC
%%	Any two-character member name	AB, BC, CD	ABC
**	Not a valid member name pattern	Not Applicable	Not Applicable

IEBCOPY Examples

The following examples illustrate some of the uses of IEBCOPY. [Table 15 on page 69](#) can be used as a quick-reference guide to IEBCOPY examples. The numbers in the "Example" column refer to examples that follow.

Table 15. IEBCOPY example directory			
Operation	Device	Comments	Example
Alter in Place	Disk	Selected members are altered in place.	“Example 10: Alter Load Modules in Place” on page 88
Convert to PDSE	Disk	Converts a partitioned data set to a PDSE.	“Example 13: Convert a Partitioned Data Set to a PDSE” on page 90

Table 15. IEBCOPY example directory (continued)

Operation	Device	Comments	Example
COPY	Disk	Copies a full data set from one disk volume to another.	“Example 1: Copy an entire data set” on page 71
COPY	Disk	Copies three input data sets to an existing output data set.	“Example 2: Merge four data sets” on page 72
COPY	Disk	Copy a PDSE to a PDSE.	“Example 14: Copy Groups from a PDSE to a PDSE” on page 91
COPY	Disk	Selects members from two input data sets and copies them to an existing output data set. One member replaces an identically named member that already exists on the output data set.	“Example 3: Copy and Replace Selected Members of a Data Set” on page 73
COPY	Disks	Selects, excludes, and copies members from input data sets to one output data set. Illustrates multiple copy operations.	“Example 6: Multiple Copy Operations with One Output Data Set” on page 78
COPY	Disks	Selects, excludes, and copies members from input data sets to different output data sets. Illustrates multiple copy operations.	“Example 7: Multiple Copy Operations with Different Output Data Sets” on page 81
COPY	Disks	Copy an entire PDSE to a PDSE with the replace (R) option.	“Example 15: Copy Groups from a PDSE to a PDSE with Replace” on page 91
COPY	Disks	Copy a selected group by specifying an alias.	“Example 16: Copy a Selected Group from a PDSE to a PDSE” on page 92
COPYGROUP	Disks	Copy a selected group of PDS members and their aliases to another PDS.	“Example 17: Copy Selected Members and their Aliases from a PDS to a PDS” on page 92
COPYGROUP	Disks	Copy selected members from a PDS to another PDS.	“Example 18: Copy Selected Members from a PDS to a PDS” on page 93
COPY and Compress-in-place	Disk	Copies two input data sets to an existing output data set, which is compressed in place. Copies and replaces all members of one data set. Members on the output data set have the same name as those replaced.	“Example 5: Merge Data Sets and Compress the Merged Data Set” on page 76
Copy and reblock	Disk (3380)	Copies a load library to devices with different optimal block sizes. Reblocking must take place before the member can be added to the load library.	“Example 11: Replace a Load Module Using COPYMOD” on page 89
Copy and reblock	Disk (3380)	Copies load library to devices having different block sizes. Reblocks the library to size compatible with each device to which the library will be copied, then copies to those devices.	“Example 12: Reblock Load Library and Distribute It to Different Device Types” on page 89
Load	Tape and Disk	Loads a sequential data set to disk.	“Example 8: Loading a Data Set” on page 86
Unload and Compress-in-place	Disk and Tape	Unloads a partitioned data set to a tape volume to create a compressed backup copy.	“Example 4: Unload and Compress a Data Set” on page 75
Unload, Load, and COPY	Disk and Tape	Excludes, unloads, loads, and copies selected members.	“Example 9: Unload Selected Members, Load, Copy and Merge” on page 87

Examples that use **disk** or **tape** in place of actual device numbers must be changed before use. The actual device numbers depend on how your installation has defined the devices to your system.

Example 1: Copy an entire data set

In this example, a partitioned data set (DATASET5) is copied from one disk volume to another. [Figure 3 on page 71](#) shows the input and output data sets before and after processing.

```
//COPY      JOB      ...
//JOBSTEP   EXEC     PGM=IEBCOPY
//SYSPRINT  DD      SYSOUT=A
//SYSUT1    DD      DSN=DATASET5,UNIT=DISK,VOL=SER=111113,
//          DISP=SHR
//SYSUT2    DD      DSN=DATASET4,UNIT=DISK,VOL=SER=111112,
//          DISP=(NEW,KEEP),SPACE=(TRK,(5,1,2))
```

The control statements are as follows:

- SYSUT1 DD defines a partitioned data set, DATASET5, that contains two members (A and C).
- SYSUT2 DD defines a new partitioned data set, DATASET4, that is to be kept after the copy operation. Five tracks are allocated for the data set; two blocks are allocated for directory entries.
- Because the partitioned data set has only two members, SYSUT3 and SYSUT4 DD are not needed.
- Because the input and output data sets are identified as SYSUT1 and SYSUT2, the SYSIN data set is not needed. The SYSUT1 data set will be copied in full to the SYSUT2 data set. After the copy operation is finished, DATASET4 will contain the same members that are in DATASET5. However, there will be no embedded, unused space in DATASET4. If you are copying a PDSE, the processing is the same, except that there is no embedded, unused space in a PDSE.

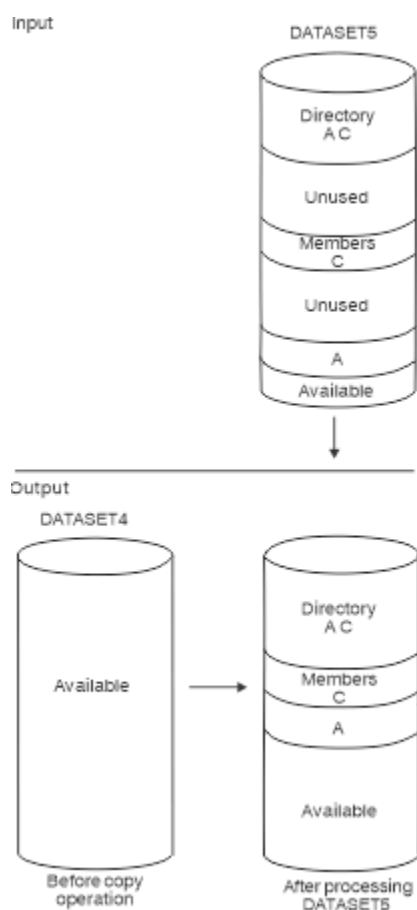


Figure 3. Copying a partitioned data set—full copy

Example 2: Merge four data sets

In this example, members are copied from three input partitioned data sets (DATASET1, DATASET5, and DATASET6) to an existing output partitioned data set (DATASET2). The sequence in which the control statements occur controls the manner and sequence in which partitioned data sets are processed. [Figure 4 on page 72](#) shows the input and output data sets before and after processing.

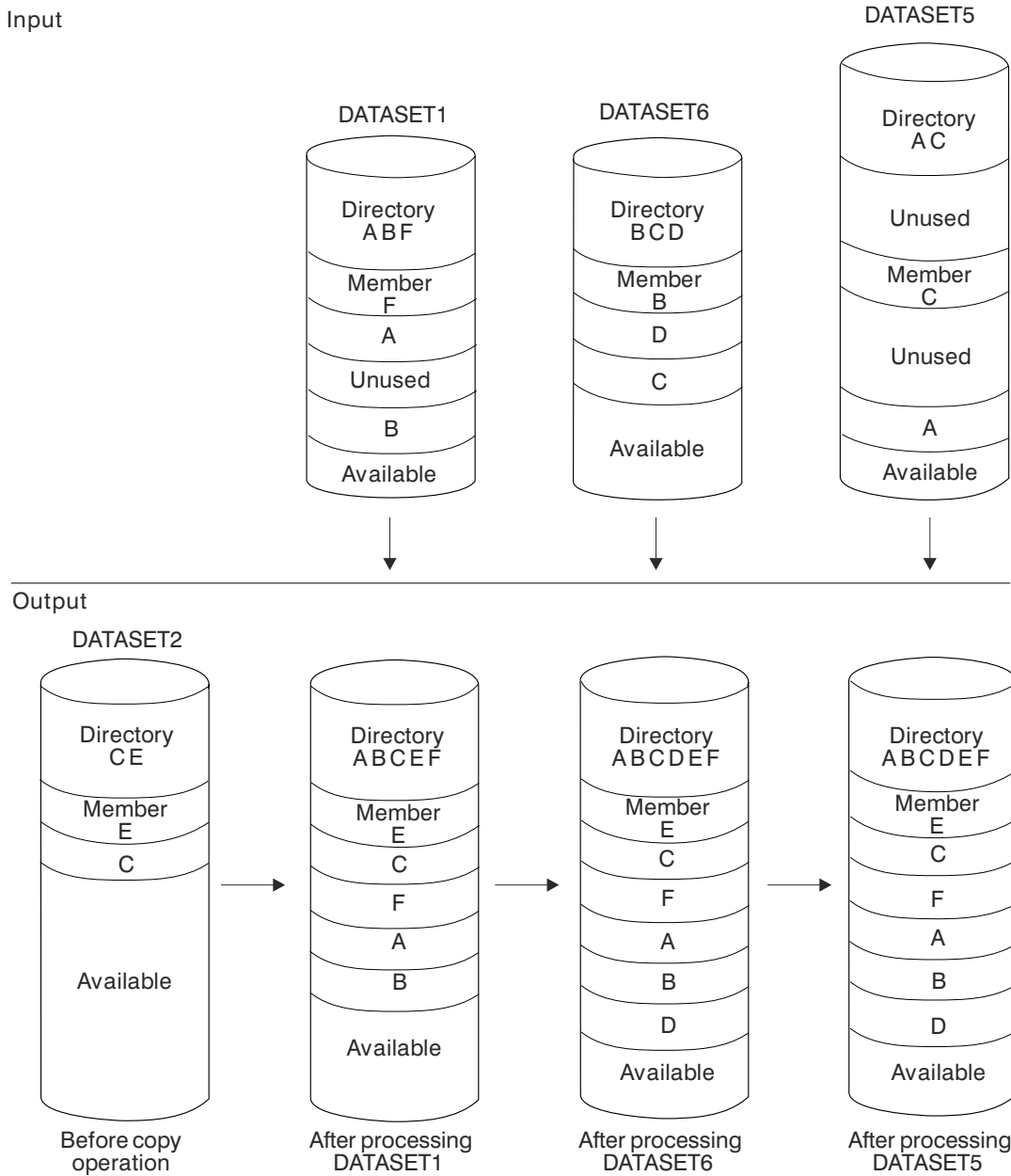


Figure 4. Copying from three input partitioned data sets

The example follows:

```
//COPY      JOB      ...
//JOBSTEP   EXEC     PGM=IEBCOPY
//SYSPRINT  DD      SYSOUT=A
//IN1       DD      DSNAME=DATASET1,UNIT=disk,VOL=SER=111112,
//          DD      DISP=SHR
//IN5       DD      DSNAME=DATASET5,UNIT=disk,VOL=SER=111114,
//          DD      DISP=OLD
//OUT2      DD      DSNAME=DATASET2,UNIT=disk,VOL=SER=111115,
```

```
//          DISP=(OLD,KEEP)
//IN6      DD  DSNAME=DATASET6,UNIT=disk,VOL=SER=111117,
//          DISP=(OLD,DELETE)
//SYSUT3   DD  UNIT=SYSDA,SPACE=(TRK,(1))
//SYSIN    DD  *
COPYOPER   COPY  OUTDD=OUT2
              INDD=IN1
              INDD=IN6
              INDD=IN5
/*
```

The control statements are as follows:

- IN1 DD defines a partitioned data set (DATASET1). This data set contains three members (A, B, and F) in fixed format with a logical record length of 80 bytes and a block size of 80 bytes.
- IN5 DD defines a partitioned data set (DATASET5). This data set contains two members (A and C) in fixed blocked format with a logical record length of 80 bytes and a block size of 160 bytes.
- OUT2 DD defines a partitioned data set (DATASET2). This data set contains two members (C and E) in fixed-block format. The members have a logical record length of 80 bytes and a block size of 240 bytes.
- IN6 DD defines a partitioned data set (DATASET6). This data set contains three members (B, C, and D) in fixed-block format with a logical record length of 80 bytes and a block size of 400 bytes. This data set is to be deleted when processing is completed.
- SYSUT3 defines a temporary spill data set.
- SYSIN DD defines the control data set, which follows in the input stream. The data set contains a COPY statement and three INDD statements.
- COPY indicates the start of the copy operation. The OUTDD parameter specifies DATASET2 as the output data set.
- The first INDD statement specifies DATASET1 as the first input data set to be processed. All members (A, B, and F) are copied to DATASET2.
- The second INDD statement specifies DATASET6 as the second input data set to be processed. Processing occurs as follows:
 1. Since replacement is not specified, members B and C, which already exist in DATASET2, are not copied to DATASET2.
 2. Member D is copied to DATASET2.
 3. All members in DATASET6 are lost when the data set is deleted.
- The third INDD statement specifies DATASET5 as the third input data set to be processed. No members are copied to DATASET2 because all exist in DATASET2.

Example 3: Copy and Replace Selected Members of a Data Set

In this example, two members (A and B) are selected from two input partitioned data sets (DATASET5 and DATASET6) and copied to an existing output partitioned data set (DATASET1). Member B replaces an identically named member that already exists on the output data set. [Figure 5 on page 74](#) shows the input and output data sets before and after processing.

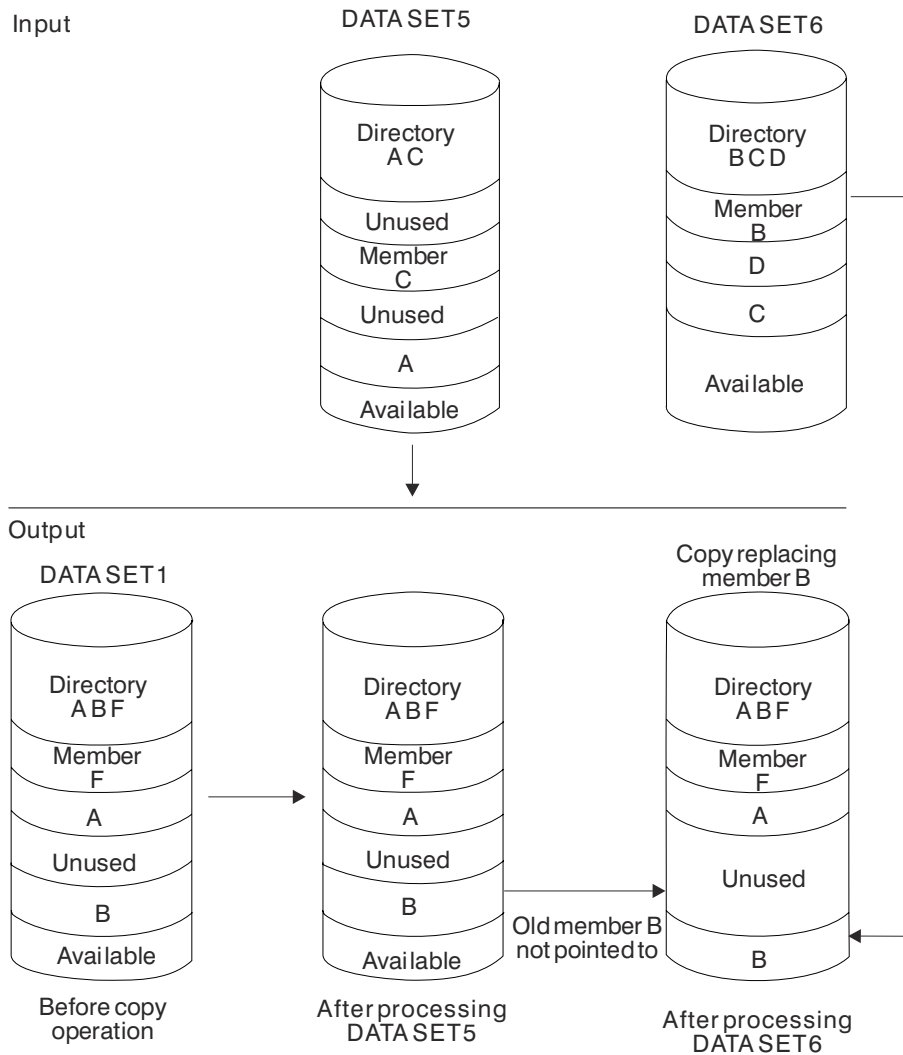


Figure 5. Selective copy with Replace specified on the member level

```
//COPY      JOB      ...
//JOBSTEP   EXEC     PGM=IEBCOPY
//SYSPRINT  DD      SYSOUT=A
//OUT1      DD      DSNAME=DATASET1,UNIT=disk,VOL=SER=111112,
//           DISP=(OLD,KEEP)
//IN6       DD      DSNAME=DATASET6,UNIT=disk,VOL=SER=111115,
//           DISP=OLD
//IN5       DD      DSNAME=DATASET5,UNIT=disk,VOL=SER=111116,
//           DISP=(OLD,KEEP)
//SYSUT3    DD      UNIT=SYSDA,SPACE=(TRK,(1))
//SYSUT4    DD      UNIT=SYSDA,SPACE=(TRK,(1))
//SYSIN     DD      *
COPYOPER    COPY     OUTDD=OUT1,
                     INDD=IN5,IN6
                     SELECT MEMBER=((B,,R),A)
/*
```

The control statements are as follows:

- OUT1 DD defines a partitioned data set (DATASET1), which contains three members (A, B and F).
- IN6 DD defines a partitioned data set (DATASET6), which contains three members (B, C and D).
- IN5 DD defines a partitioned data set (DATASET5), which contains two members (A and C).
- SYSUT3 and SYSUT4 DD define temporary spill data sets. One track is allocated for each on a disk volume.

- SYSIN DD defines the control data set, which follows in the input stream. The data set contains a COPY statement, an INDD statement, and a SELECT statement.
- COPY indicates the start of the copy operation. The use of a SELECT statement causes a selective copy. The OUTDD parameter specifies DATASET1 as the output data set.
- INDD specifies DATASET5 as the first input data set to be processed and DATASET6 as the second input data set to be processed. Processing occurs as follows:
 1. Selected members are searched for on DATASET5.
 2. Member A is found, but is not copied to DATASET1 because DATASET1 already has a member named "A", and the replace option is not specified for member A.
 3. Selected members not found on DATASET5 are searched for on DATASET6.
 4. Member B is found and copied to DATASET1, even though there is already a DATASET1 member "B" in DATASET1, because the replace option is specified for member B on the member level. The pointer in DATASET1's directory is changed to point to the new (copied) member B; thus, the space occupied by the old member B is unused.
- SELECT specifies the members to be selected from the input data sets (DATASET5 and DATASET6) to be copied to the output data set (DATASET1).

Example 4: Unload and Compress a Data Set

In this example, a partitioned data set is unloaded to a tape volume to create a backup copy of the data set. If this step is successful, the partitioned data set is to be compressed in place.

```
//SAVE      JOB      ...
//STEP1     EXEC    PGM=IEBCOPY
//SYSPRINT  DD      SYSOUT=A
//SYSUT1    DD      DSN=PARTPDS,UNIT=DISK,VOL=SER=PCP001,
//              DISP=OLD
//SYSUT2    DD      DSN=SAVDATA,UNIT=TAPE,VOL=SER=TAPE03,
//              DISP=(NEW,KEEP),LABEL=(,SL)
//SYSUT3    DD      DSN=TEMP1,UNIT=DISK,VOL=SER=111111,
//              DISP=(NEW,DELETE),SPACE=(80,(60,45))
//SYSIN     DD      DUMMY
//STEP2     EXEC    PGM=IEBCOPY,COND=(0,NE),PARM='SIZE=500K'
//SYSPRINT  DD      SYSOUT=A
//COMPDS    DD      DSN=PARTPDS,UNIT=DISK,DISP=OLD,
//              VOL=SER=PCP001
//SYSUT3    DD      DSN=TEMPA,UNIT=DISK,VOL=SER=111111,
//              DISP=(NEW,DELETE),SPACE=(80,(60,45))
//SYSIN     DD      *
              COPY  OUTDD=COMPDS,INDD=COMPDS
/*
```

The control statements are as follows:

- SYSUT1 DD defines a partitioned data set (PARTPDS) that resides on a disk volume and is assumed to have 700 members. The number of members is used to calculate the space allocation on SYSUT3.
- SYSUT2 DD defines a sequential data set to hold PARTPDS in unloaded form. Block size information can optionally be added; this data set must be NEW.
- SYSUT3 DD defines the temporary spill data set. The SYSUT4 data set is never used for an unload operation.
- SYSIN DD defines the control data set. Because SYSIN is dummied and SYSUT2 defines a sequential data set, all members of the SYSUT1 data set will be unloaded to the SYSUT2 data set.
- The second EXEC statement marks the beginning of the compress-in-place operation. The SIZE parameter indicates that the buffers are to be as large as possible. The COND parameter indicates that the compress-in-place is to be performed only if the unload operation was successful.
- COMPDS DD defines a partitioned data set (PARTPDS) that contains 700 members and resides on a disk volume.
- SYSUT3 DD defines the temporary spill data set to be used if there is not enough space in main storage for the input data set's directory entries. TEMPA contains one 80-character record for each member.

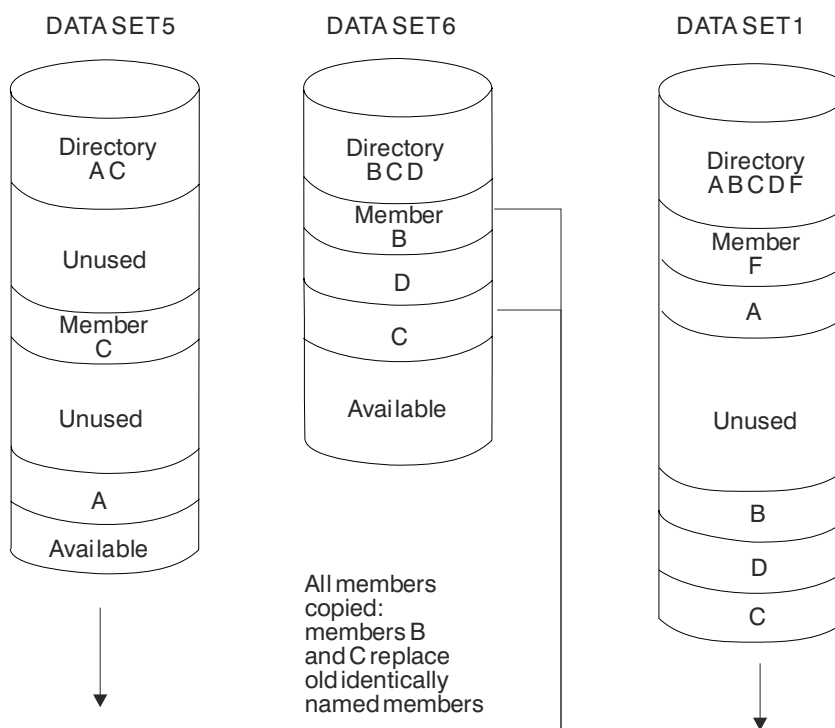
- SYSIN DD defines the control data set, which follows in the input stream. The data set contains a COPY statement.
- COPY marks the beginning of the copy operation. Because the same DD statement is specified for both the INDD and OUTDD operands, the data set is compressed in place. If a PDSE is being used, this step will not be processed.

If you want to unload more than one data set in a single use of IEBCOPY, you must use a separate COPY statement for each unload operation. Only one input data set may be specified in an unload operation.

Example 5: Merge Data Sets and Compress the Merged Data Set

In this example, two input partitioned data sets (DATASET5 and DATASET6) are copied to an existing output partitioned data set (DATASET1). In addition, all members on DATASET6 are copied; members on the output data set that have the same names as the copied members are replaced. After DATASET6 is processed, the output data set (DATASET1) is compressed in place. [Figure 6 on page 77](#) shows the input and output data sets before and after processing.

Input



Output

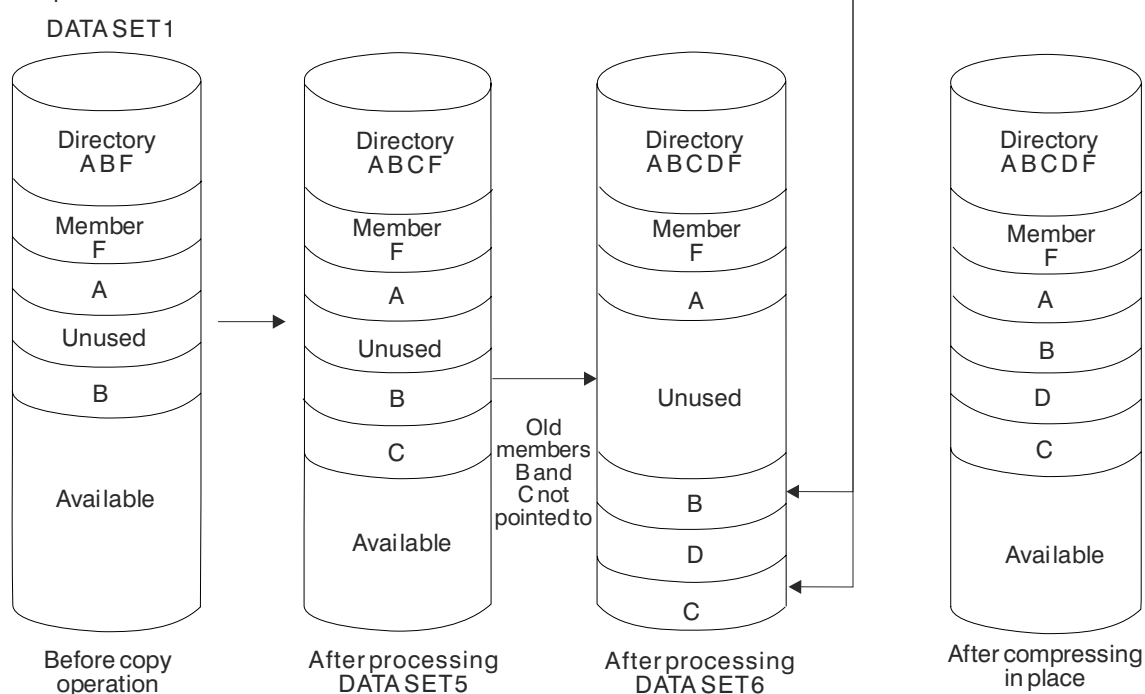


Figure 6. Compress-in-Place following full copy with “replace” specified

```
//COPY      JOB    ...
//JOBSTEP   EXEC   PGM=IEBCOPY
//SYSPRINT  DD     SYSOUT=A
//INOUT1    DD     DSN=DATASET1,UNIT=disk,VOL=SER=111112,
//           DISP=(OLD,KEEP)
//IN5       DD     DSN=DATASET5,UNIT=disk,VOL=SER=111114,
//           DISP=OLD
//IN6       DD     DSN=DATASET6,UNIT=disk,VOL=SER=111115,
//           DISP=(OLD,KEEP)
```

```
//SYSUT3 DD UNIT=SYSDA,SPACE=(TRK,(1))
//SYSUT4 DD UNIT=SYSDA,SPACE=(TRK,(1))
//SYSIN DD *
COPYOPER COPY OUTDD=INOUT1,INDD=(IN5,(IN6,R),INOUT1)
/*
```

The control statements are as follows:

- INOUT1 DD defines a partitioned data set (DATASET1), which contains three members (A, B and F).
- IN5 DD defines a partitioned data set (DATASET5), which contains two members (A and C).
- IN6 DD defines a partitioned data set (DATASET6), which contains three members (B, C and D).
- SYSUT3 and SYSUT4 DD define temporary spill data sets. One track is allocated for each on a disk volume.
- SYSIN DD defines the control data set, which follows in the input stream. The data set contains a COPY statement.
- COPY indicates the start of the copy operation. The OUTDD operand specifies DATASET1 as the output data set.

The INDD operand specifies DATASET5 as the first input data set to be processed. It then specifies DATASET6 as the second input data set to be processed. In addition, the replace option is specified for all members copied from DATASET6. Finally, it specifies DATASET1 as the last input data set to be processed. Since DATASET1 is also the output data set, DATASET1 is compressed in place. However, if DATASET1 is a PDSE, the compress-in-place operation will not be processed.

Processing occurs as follows:

1. Member A is not copied from DATASET5 into DATASET1 because it already exists on DATASET1 and the replace option was not specified for DATASET5.
2. Member C is copied from DATASET5 to DATASET1, occupying the first available space.
3. All members are copied from DATASET6 to DATASET1, immediately following the last member. Members B and C are copied even though the output data set already contains members with the same names because the replace option is specified on the data set level.

The pointers in DATASET1's directory are changed to point to the new members B and C. Thus, the space occupied by the old members B and C is unused. The members currently on DATASET1 are compressed in place, thereby eliminating embedded unused space.

Example 6: Multiple Copy Operations with One Output Data Set

In this example, members are selected, excluded, and copied from input partitioned data sets onto an output partitioned data set. This example is designed to illustrate multiple copy operations.

The example follows. [Figure 7 on page 80](#) and [Figure 8 on page 81](#) show the input and output data sets before and after processing.

```
//COPY JOB ...
//JOBSTEP EXEC PGM=IEBCOPY
//SYSPRINT DD SYSOUT=A
//INOUTA DD DSN=DATASETA,UNIT=disk,VOL=SER=111113,
// DISP=OLD
//INB DD DSN=DATASETB,UNIT=disk,VOL=SER=111115,
// DISP=(OLD,KEEP)
//INC DD DSN=DATASETC,UNIT=disk,VOL=SER=111114,
// DISP=(OLD,KEEP)
//IND DD DSN=DATASETD,UNIT=disk,VOL=SER=111116,
// DISP=OLD
//INE DD DSN=DATASETE,UNIT=disk,VOL=SER=111117,
// DISP=OLD
//OUTX DD DSN=DATASETX,UNIT=disk,VOL=SER=111112,
// DISP=(NEW,KEEP),SPACE=(TRK,(3,1,2))
//SYSUT3 DD UNIT=SYSDA,SPACE=(TRK,(1))
//SYSIN DD *
COPERST1 COPY O=OUTX,I=INOUTA
COPY OUTDD=INOUTA,INDD=INOUTA
INDD=INB
COPY OUTDD=INOUTA
```

```

        EXCLUDE  INDD=IND
                MEMBER=MM
                INDD=INC
        SELECT   MEMBER=( (ML,MD,R) )
                INDD=INE
/*

```

The control statements are as follows:

- INOUTA DD defines a partitioned data, DATASETA, which contains seven members (MA, MB, MC, MD, ME, MF and MG).
- INB DD defines a partitioned data set, DATASET B, which contains two members (MA and MJ).
- INC DD defines a partitioned data set, DATASET C, which contains four members (MF, ML, MM and MN).
- IND DD defines a partitioned data set, DATASET D, which contains two members (MM and MP).
- INE DD defines a partitioned data set, DATASET E, which contains four members (MD, ME, MF and MT).
- OUTX DD defines a partitioned data set (DATASET X). This data set is new and is to be kept after the copy operation. Three tracks are allocated for the data set on a disk volume. Two blocks are allocated for directory entries.
- SYSUT3 defines a temporary spill data set.
- SYSIN DD defines the control data set, which follows in the input stream. The data set contains two COPY statements, several INDD statements, a SELECT statement, and an EXCLUDE statement.
- The first COPY statement indicates the start of the first copy operation. This copy operation is done to create a backup copy of DATASETA.
- The second COPY statement indicates the start of another copy operation. Since DATASETA is specified in both the INDD and OUTDD parameters, DATASETA is compressed in place.

The output data set is compressed in place first to save space because it is known that it contains embedded, unused space.

The following INDD statement specifies DATASET B as the next input data set to be copied. Only member MJ is copied, because DATASETA already contains a member named "MA".

- The third COPY statement indicates the start of the third copy operation. The OUTDD parameter specifies DATASETA as the output data set. This copy operation contains more than one copy step.

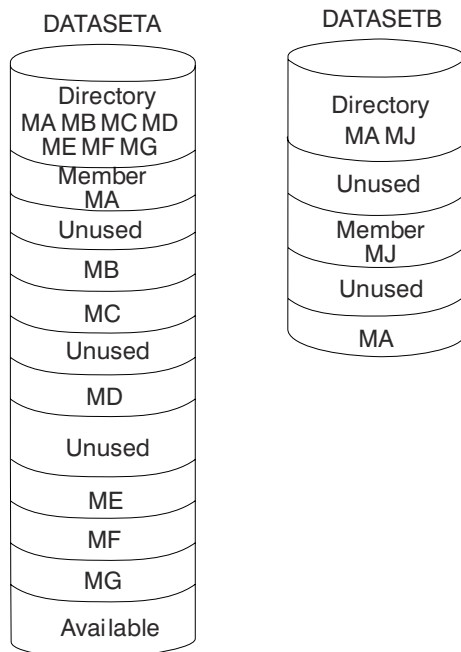
The first INDD statement specifies DATASET D as the first input data set to be processed. Only member MP is copied to DATASETA because the EXCLUDE statement specifies that member MM is to be excluded from the first copy step within this copy operation.

The second INDD statement marks the beginning of the second copy step for this copy operation and specifies DATASET C as the second input data set to be processed. The SELECT statement specifies that member ML of DATASET C is to be renamed "MD", and that the new member will replace any member in DATASETA that happens to be named "MD". Member ML is searched for, found, copied to DATASETA and renamed.

The third INDD statement marks the beginning of the third copy step for this copy operation and specifies DATASET E as the last data set to be copied. Only member MT is copied, because DATASETA already contains the other members. Because the INDD statement is not followed by an EXCLUDE or SELECT statement, a full copy is performed.

Compress-in-Place Operation

Input



Output

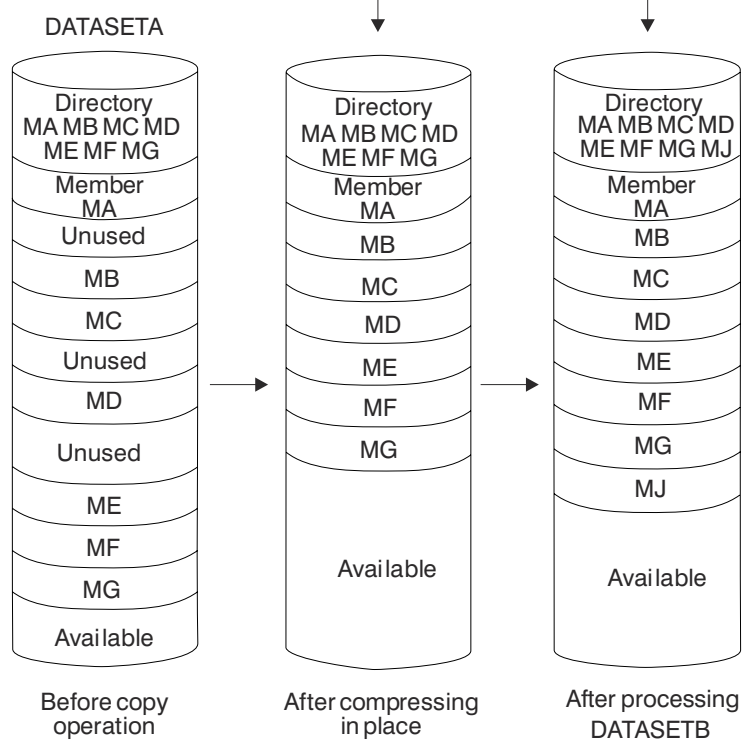
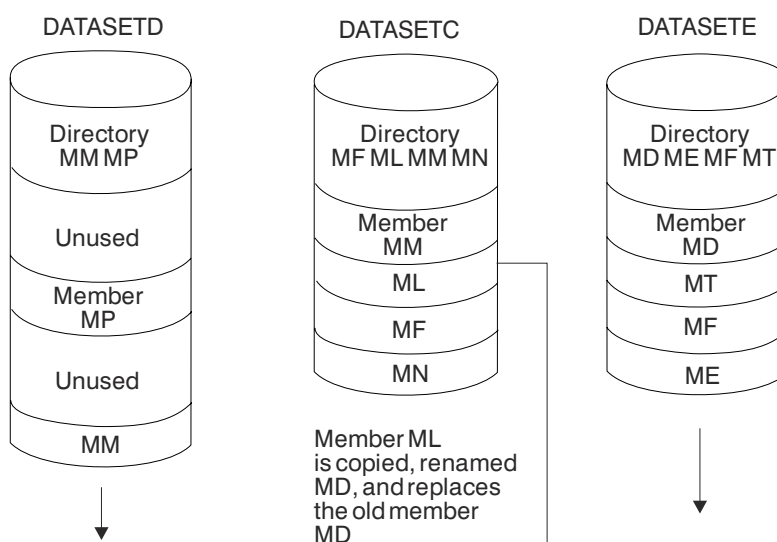


Figure 7. Multiple copy operations/copy steps (Part 1 of 2)

Multiple Copy Steps

Input



Output

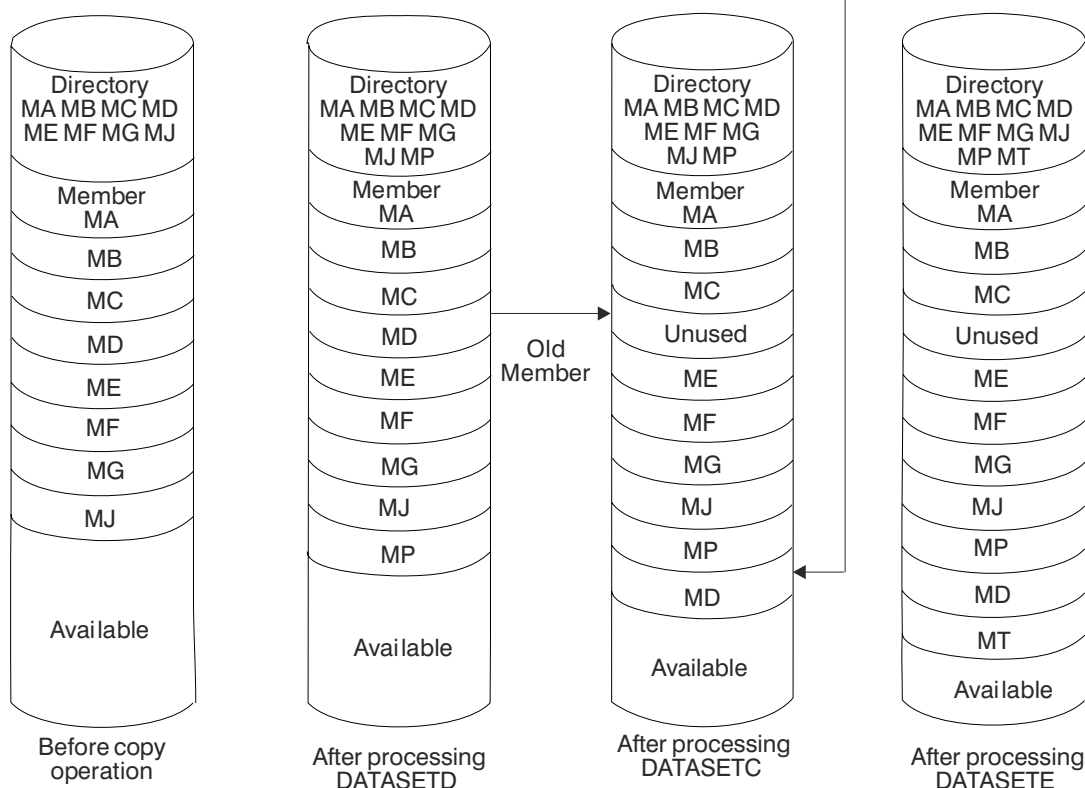


Figure 8. Multiple copy operations/copy steps (Part 2 of 2)

Example 7: Multiple Copy Operations with Different Output Data Sets

In this example, members are selected, excluded, and copied from input partitioned data sets to an output partitioned data set. This example is designed to illustrate multiple copy operations. [Figure 9 on page 84](#), [Figure 10 on page 85](#), and [Figure 11 on page 86](#) show the input and output data sets before and after processing.

The example follows:

```

//COPY      JOB      ...
//JOBSTEP   EXEC     PGM=IEBCOPY
//SYSPRINT  DD      SYSOUT=A
//OUTA      DD      DSN=DATASETA,UNIT=disk,VOL=SER=111113,
//           DISP=OLD
//INOUTB    DD      DSN=DATASETB,VOL=SER=111115,UNIT=disk,
//           DISP=(OLD,KEEP)
//INOUTC    DD      DSN=DATASETC,VOL=SER=111114,UNIT=disk,
//           DISP=(OLD,KEEP)
//INOUTD    DD      DSN=DATASETD,VOL=SER=111116,DISP=OLD,
//           UNIT=disk
//INE       DD      DSN=DATASETE,VOL=SER=111117,DISP=OLD,
//           UNIT=disk
//SYSUT3    DD      UNIT=SYSDA,SPACE=(TRK,(1))
//SYSUT4    DD      UNIT=SYSDA,SPACE=(TRK,(1))
//SYSIN     DD      *
           COPY      OUTDD=OUTA
           INDD=INE
           SELECT    MEMBER=(MA,MJ)
           INDD=INOUTC
           EXCLUDE    MEMBER=(MM,MN)
           COPY      OUTDD=INOUTB,INDD=INOUTD
           INDD=((INOUTC,R),INOUTB)
           COPY      OUTDD=INOUTD,INDD=((INOUTB,R))
           SELECT    MEMBER=MM
/*

```

The control statements are as follows:

- OUTA DD defines a partitioned data set, DATASETA, which contains three members (MA, MB and MD).
- INOUTB DD defines a partitioned data set, DATASETB, which contains two members (MA and MJ).
- INOUTC DD defines a partitioned data set, DATASETC, which contains four members (MF, ML, MM and MN).
- INOUTD DD defines a partitioned data set, DATASETD, which contains two members (MM and MP).
- INE DD defines a partitioned data set, DATASETE, which contains three members (MA, MJ and MK).
- SYSUT3 and SYSUT4 DD define temporary spill data sets. One track is allocated for each on a disk volume.
- SYSIN DD defines the control data set, which follows in the input stream. The data set contains three COPY statements, two SELECT statements, one EXCLUDE statement, and several INDD statements.
- The first COPY statement indicates the start of a copy operation. The OUTDD operand specifies DATASETA as the output data set.

The first INDD statement specifies DATASETE as the first input data set to be processed. The SELECT statement specifies that members MA and MJ are to be copied from DATASETE to DATASETA. Processing occurs as follows:

1. Member MA is searched for and found, but is not copied because the replace option is not specified.
2. Member MJ is searched for, found, and copied to DATASETA.

The second INDD statement marks the end of the first copy step and the beginning of the second copy step within the first copy operation. It specifies DATASETC as the second input data set to be processed. Members MF and ML, which are not named on the EXCLUDE statement, are copied because DATASETA contains neither one of them. EXCLUDE specifies that members MM and MN are not to be copied from DATASETC to DATASETA.

- The second COPY statement indicates the start of another copy operation. The OUTDD parameter specifies DATASETB as the output data set. The INDD parameter specifies DATASETD as the first input data set to be processed. Members MP and MM are copied to DATASETB.

The next INDD statement specifies DATASETC as the second and DATASETB as the third input data set to be processed. Members MF, ML, MM and MN are copied from DATASETC. Member MM is copied, although DATASETB already contains a member MM, because the replace option is specified.

The pointer in DATASETB's directory is changed to point to the new (copied) member MM. Thus, the space occupied by the replaced member MM is embedded, unused space. DATASETB is then

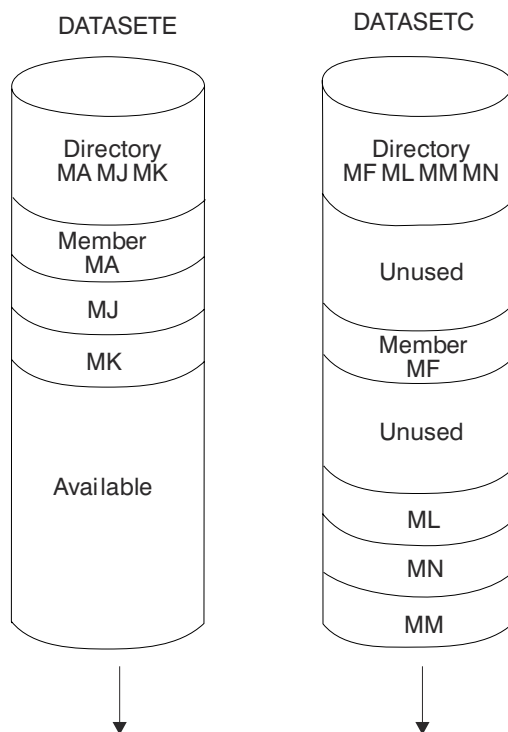
compressed in place to remove embedded, unused space. (DATASETB is specified as both the input and output data sets.)

- The third COPY statement indicates the start of the last copy operation. The OUTDD parameter specifies DATASET D as the output data set. The INDD parameter specifies DATASET B as the input data set.

SELECT specifies that member MM is to be copied from DATASET B to DATASET D. Since the replace option is specified on the data set level, member MM is copied and replaces DATASET D's member MM.

First copy operation

Input



Output

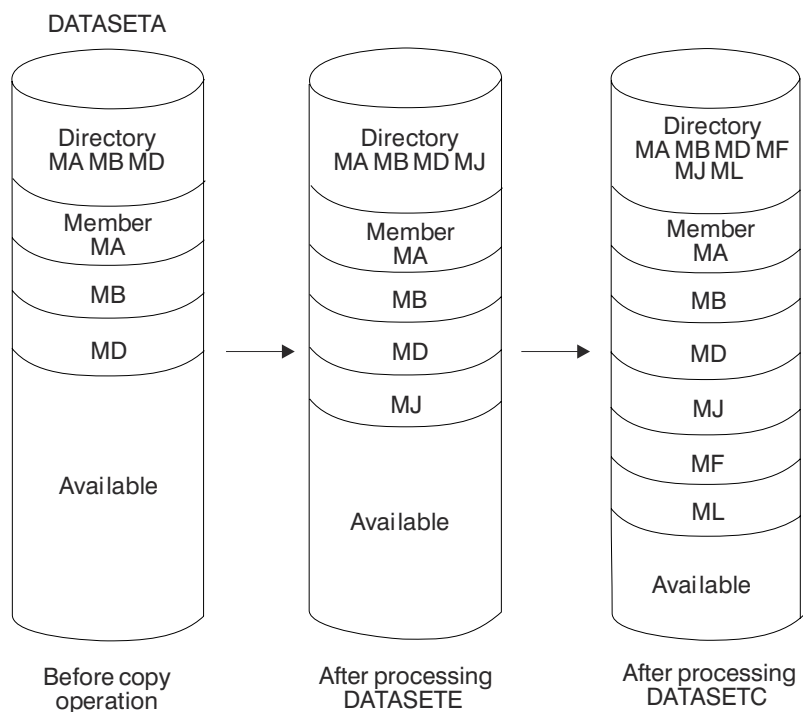
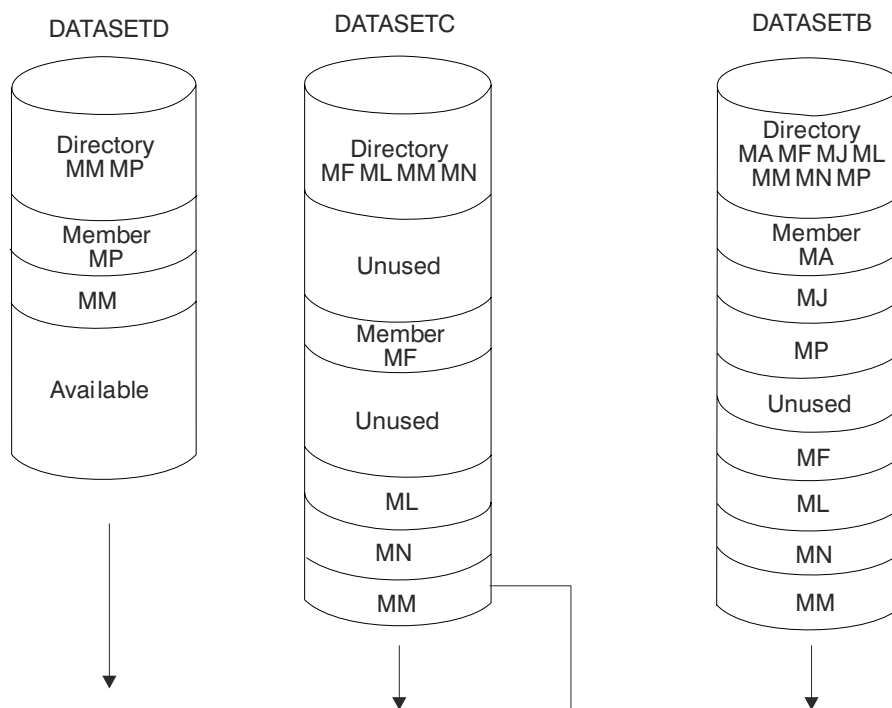


Figure 9. Multiple copy operations/copy steps within a job step (Part 1 of 3)

Second copy operation

Input



Output

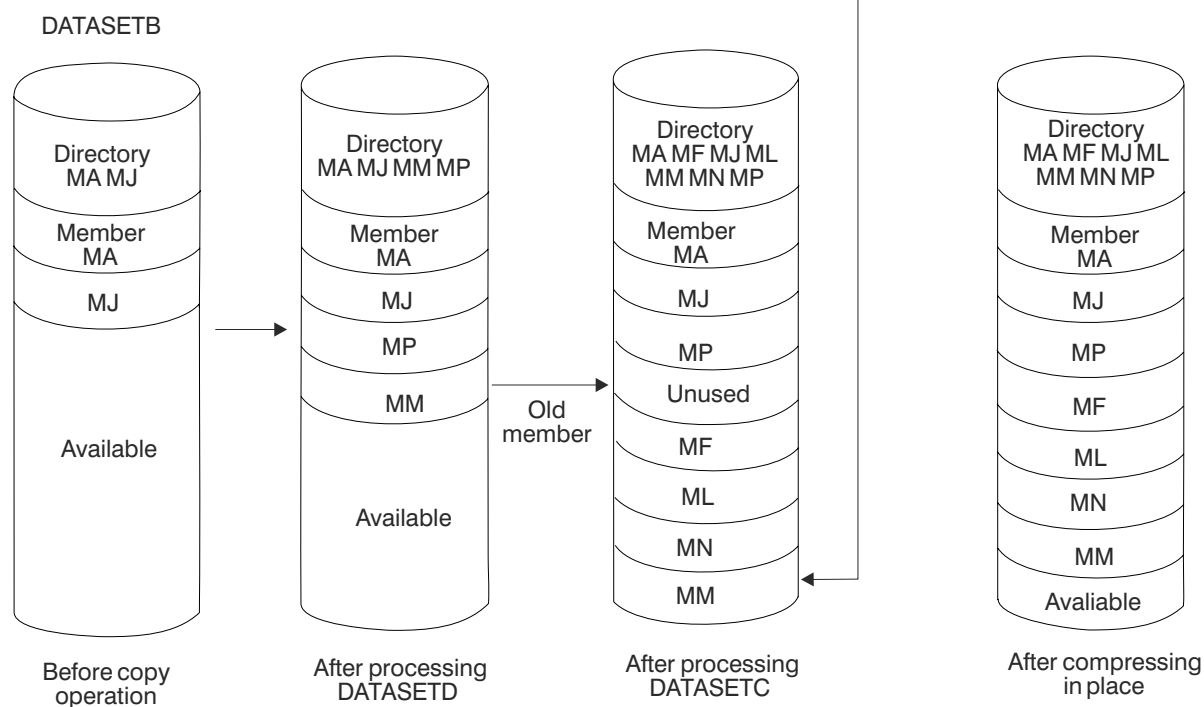


Figure 10. Multiple copy operations/copy steps within a job step (Part 2 of 3)

Third copy operation

Input

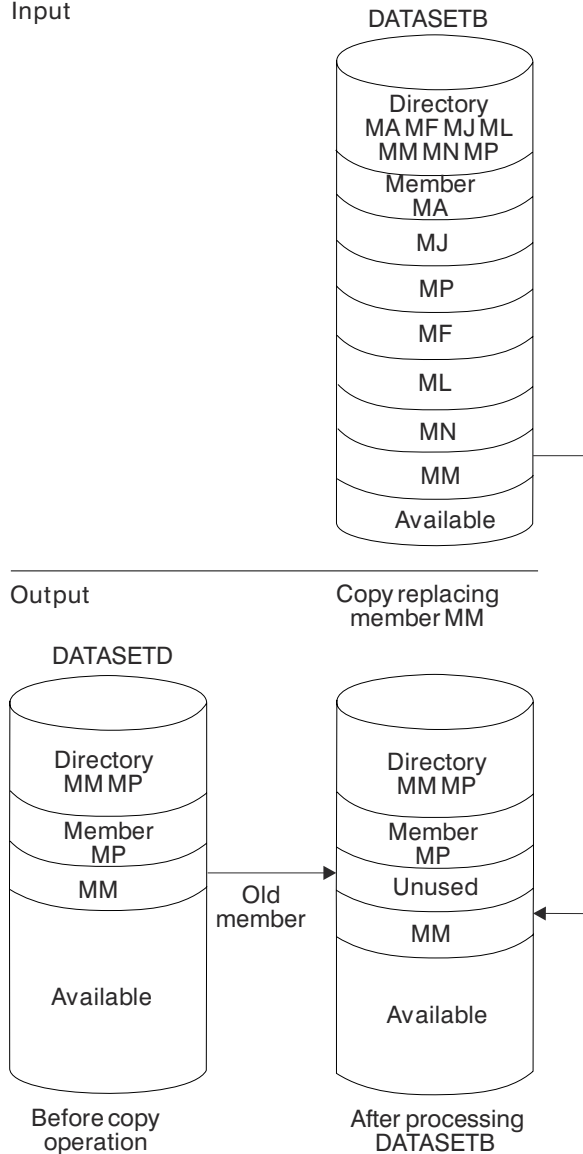


Figure 11. Multiple copy operations/copy steps within a job step (Part 3 of 3)

Example 8: Loading a Data Set

In this example, a sequential data set that was created by an IEBCOPY unload operation is loaded.

```
//LOAD      JOB      ...
//STEPS     EXEC     PGM=IEBCOPY
//SYSPRINT  DD      SYSOUT=A
//SYSUT1    DD      DSN=UNLOADSET,UNIT=tape,LABEL=(,SL),
//           VOL=SER=TAPE01,DISP=OLD
//SYSUT2    DD      DSN=DATASET4,UNIT=disk,VOL=SER=2222222,
//           DISP=(NEW,KEEP),SPACE=(CYL,(10,5,10))
//SYSUT3    DD      DSN=TEMP1,UNIT=disk,VOL=SER=111111,
//           DISP=(NEW,DELETE),SPACE=(80,(15,1))
//SYSIN     DD      DUMMY
/*
```

The control statements are as follows:

- SYSUT1 DD defines a sequential data set that was previously created by an IEBCOPY unload operation. The data set contains 28 members in sequential organization.
- SYSUT2 DD defines a new partitioned data set on a disk volume. This data set is to be kept after the load operation. Ten cylinders are allocated for the data set; ten blocks are allocated for directory entries.
- SYSUT3 DD defines a temporary spill data set on a disk volume.
- SYSIN DD defines the control data set. Because SYSIN is dummied, SYSUT1 defines a sequential data set, and SYSUT2 defines a partitioned data set, the entire SYSUT1 data set will be loaded into the SYSUT2 data set.

Example 9: Unload Selected Members, Load, Copy and Merge

In this example, members are selected, excluded, unloaded, loaded, and copied. Processing will occur as follows:

1. unload, excluding members
2. unload, selecting members
3. load and copy to merge members

```
//COPY      JOB      ...
//STEP      EXEC    PGM=IEBCOPY
//SYSPRINT  DD      SYSOUT=A
//PDS1      DD      DSN=ACCOUNTA,UNIT=DISK,VOL=SER=333333,
//              DISP=OLD
//PDS2      DD      DSN=ACCOUNTB,UNIT=DISK,VOL=SER=333333,
//              DISP=OLD
//SEQ1      DD      DSN=SAVAC,UNIT=DISK,VOL=SER=333333,
//              DISP=(NEW,KEEP),SPACE=(CYL,(5,2))
//SEQ2      DD      DSN=SAVACB,UNIT=TAPE,VOL=SER=T01911,
//              DISP=(NEW,KEEP),LABEL=(,SL)
//NEWUP     DD      DSN=NEWACC,UNIT=TAPE,VOL=SER=T01219,
//              DISP=OLD,LABEL=(,SL)
//MERGE     DD      DSN=ACCUPDAT,UNIT=DISK,VOL=SER=222222,
//              DISP=OLD
//SYSUT3     DD      DSN=TEMP1,VOL=SER=666666,UNIT=DISK,
//              DISP=(NEW,DELETE),SPACE=(80,(1,1))
//SYSUT4     DD      DSN=TEMP2,VOL=SER=666666,UNIT=DISK,
//              DISP=(NEW,DELETE),SPACE=(256,(1,1)),DCB=(KEYLEN=8)
//SYSIN      DD      *
//              COPY      OUTDD=SEQ1,INDD=PDS1
//              EXCLUDE   MEMBER=(D,C)
//              COPY      OUTDD=SEQ2,INDD=PDS2
//              SELECT    MEMBER=(A,K)
//              COPY      OUTDD=MERGE,INDD=((NEWUP,R),PDS1,PDS2)
//              EXCLUDE   MEMBER=A
/*
```

The control statements are as follows:

- PDS1 DD defines a partitioned data set called ACCOUNTA that contains six members (A, B, C, D, E, and F).
- PDS2 DD defines a partitioned data set called ACCOUNTB that contains three members (A, K, and L).
- SEQ1 DD defines a new sequential data set called SAVAC.
- SEQ2 DD defines a new sequential data set called SAVACB on a tape volume. The tape has IBM standard labels.
- NEWUP DD defines an old sequential data set called NEWACC that is the unloaded form of a partitioned data set that contains eight members (A, B, C, D, M, N, O, and P). It resides on a tape volume.
- MERGE DD defines a partitioned data set called ACCUPDAT that contains six members (A, B, C, D, Q, and R).
- SYSUT3 and SYSUT4 DD define temporary spill data sets.
- SYSIN DD defines the control data set, which follows in the input stream.

- The first COPY statement indicates the start of the first copy operation. The OUTDD parameter specifies that SAVAC is the output data set, and the INDD parameter specifies that ACCOUNTA is the input data set. Because SAVAC is a sequential data set, ACCOUNTA will be unloaded in this copy operation.

The EXCLUDE statement specifies that members D and C are not to be unloaded to SAVAC with the rest of ACCOUNTA.

- The second COPY statement indicates the start of the second copy operation. The OUTDD parameter specifies that SAVACB is the output data set, and the INDD parameter specifies that ACCOUNTB is the input data set. Because SAVACB is a sequential data set, ACCOUNTB will be unloaded in this copy operation.

The SELECT statement specifies that members A and K are the only members of ACCOUNTB that are to be unloaded to SAVACB.

- The third COPY statement indicates the start of the last copy operation. The OUTDD parameter specifies that ACCUPDAT is the output data set. The EXCLUDE statement specifies that member A is excluded from this copy operation. The three data sets specified in the INDD parameter will be processed as follows:
 1. The data set NEWACC is a sequential data set, so it is loaded into ACCUPDAT. Because the replace option is specified, members B, C, and D in NEWACC replace identically named members in ACCUPDAT. The remaining members of NEWACC are also copied to ACCUPDAT, except for A, which is excluded from the copy operation.
 2. The data set ACCOUNTA is a partitioned data set, so its members are copied to ACCUPDAT. Because replacement is not specified, only members E and F are copied.
 3. The data set ACCOUNTB is a partitioned data set, so its members are copied to ACCUPDAT. Only members K and L are copied.

Example 10: Alter Load Modules in Place

In this example, all members of data set MODLIBJ, members MODX, MODY, and MODZ of data set MODLIBK, and all members of data set MODLIBL, except MYMACRO and MYJCL, are altered in place.

```
//ALTERONE JOB ...
//STEP1 EXEC PGM=IEBCOPY
//SYSPRINT DD SYSOUT=A
//SYSUT3 DD UNIT=SYSDA,SPACE=(TRK,(5,1))
//LIBJ DD DSN=MODLIBJ,DISP=(OLD,KEEP)
//LIBK DD DSN=MODLIBK,DISP=(OLD,KEEP)
//LIBL DD DSN=MODLIBL,DISP=(OLD,KEEP)
//SYSIN DD *
      ALTERMOD OUTDD=LIBJ
      ALTERMOD OUTDD=LIBK,LIST=NO
      SELECT MEMBER=(MODX,MODY,MODZ)
      ALTERMOD OUTDD=LIBL
      EXCLUDE MEMBER=(MYMACRO,MYJCL)
/*
```

The control statements are as follows:

- LIBJ DD defines the partitioned data set MODLIBJ, which has been previously created and cataloged.
- LIBK DD defines the partitioned data set MODLIBK, which has been previously created and cataloged.
- LIBL DD defines the partitioned data set MODLIBL, which has been previously created and cataloged.
- SYSIN DD defines the control data set, which follows in the input stream.
- The first ALTERMOD statement specifies that the entire data set defined in LIBJ is to be altered in place.
- The second ALTERMOD statement plus the following SELECT statement indicates that members MODX, MODY, and MODZ are to be altered in place. The remainder of MODLIBK is unchanged.
- The third ALTERMOD statement plus the following EXCLUDE statement indicates that all of MODLIBL is to be altered in place except the members called MYMACRO and MYJCL. These members remain unchanged.

Example 11: Replace a Load Module Using COPYMOD

In this example, a load module in an existing load library is replaced by another module. The new module originally resides on a 3390 DASD device, whereas the load library to which it is copied resides on a 3380. Because the module has a block size larger than the block size assigned to the output data set, the module must be reblocked before it is added to the load library.

This example illustrates how you can transfer load modules between devices of different sizes. In this case, updated modules are created on a 3390 and tested before being added to the load library for general use.

```
//STEP1 EXEC PGM=IEBCOPY
//REPLACE JOB ...
//SYSPRINT DD SYSOUT=A
//TESTLIB DD DSN=JOHNDOE.COBOL.TESTLOAD,DISP=SHR,UNIT=3390,
//          VOL=SER=TEST01,DCB=(BLKSIZE=23470)
//PRODLIB DD DSN=PAYROLL.MASTER.LOADLIB,DISP=(OLD,KEEP)
//          UNIT=3380,VOL=SER=PROD01,DCB=(BLKSIZE=19069)
//SYSIN DD *
//          COPYMOD OUTDD=PRODLIB,INDD=TESTLIB
//          SELECT MEMBER=((WAGETAX,,R))
/*
```

The control statements are as follows:

- TESTLIB DD defines a load library on a 3390 direct access device. It has a block size of 23470.
- PRODLIB DD defines a load library on a 3380 direct access device. It has a block size of 19069.
- SYSIN DD defines the control data set, which follows in the input stream.
- The COPYMOD statement identifies PAYROLL.MASTER.LOADLIB as the output data set and JOHNDOE.COBOL.TESTLOAD as the input data set. The SELECT statement indicates that the load module WAGETAX is to be copied from the input data set and is to replace any member with that name that is in the output data set. The member is also reblocked to 19069.

Note that, in this case, COPYMOD has to be used in order to copy the member WAGETAX into the PAYROLL.MASTER.LOADLIB. Because the original block size of WAGETAX is larger than the largest block size that can reside in the output data set, attempting this operation with the COPY statement would be unsuccessful. The problem would be attributed to a DCB validation error because of incorrect block size.

Example 12: Reblock Load Library and Distribute It to Different Device Types

In this example, a load library is distributed (by copying it) to devices whose maximum block size differs from that on which the original load library resides. The library is first reblocked to a maximum block size that is compatible with each of the devices to which the library will be copied. Then, the library is copied to those devices.

This example illustrates how load libraries can be developed on one type of direct access device and then distributed to other types of direct access devices.

```
//RBLKCOPY JOB ...
//REBLOCK EXEC PGM=IEBCOPY
//SYSPRINT DD SYSOUT=A
//TESTED DD DSN=TESTED.MASTER.LOADLIB,DISP=SHR
//STDSIZE DD DSN=PROGRAM.MASTER.LOADLIB,DISP=(OLD,KEEP),
//          UNIT=3390,VOL=SER=PROG01,DCB=(BLKSIZE=23470)
//SYSIN DD *
//          COPYMOD OUTDD=STDSIZE,INDD=TESTED,MAXBLK=13030
/*
//DISTRIB EXEC PGM=IEBCOPY
//SYSPRINT DD SYSOUT=A
//STDSIZE DD DSN=PROGRAM.MASTER.LOADLIB,DISP=SHR
//LIB3350 DD DSN=PROGRAM.LIB3380.LOADLIB,DISP=(OLD,KEEP),
//          UNIT=3380,VOL=SER=PACK01,DCB=(BLKSIZE=19069)
//LIB3330 DD DSN=PROGRAM.LIB3380.LOADLIB,DISP=(OLD,KEEP),
//          UNIT=3380,VOL=SER=PACK02,DCB=(BLKSIZE=13030)
//SYSIN DD *
//          COPY OUTDD=LIB3380,INDD=STDSIZE
```

```

COPY OUTDD=LIB3380,INDD=STDSIZE
/*

```

The control statements are as follows:

- The REBLOCK EXEC statement begins the reblocking step.
- TESTED DD defines the cataloged load library TESTED.MASTER.LOADLIB.
- STDSIZE DD defines an existing data set, PROGRAM.MASTER.LOADLIB, which resides on a 3390 direct access device and has a block size of 23470.
- The COPYMOD statement in the SYSIN data set specifies that TESTED.MASTER.LOADLIB is to be merged into PROGRAM.MASTER.LOADLIB. It also specifies that PROGRAM.MASTER.LOADLIB is to be reblocked with a maximum block size of 13030. This block size is chosen because it is small enough to fit on both 3380 and 3390 direct access devices.
- The DISTRIB EXEC statement begins the distribution step, where the reblocked data set is copied to devices with different maximum block sizes.
- STDSIZE DD defines the same data set that was reblocked in the previous step.
- LIB3380 DD defines the data set PROGRAM.LIB3380.LOADLIB, which resides on a 3380 direct access device.
- The COPY statements in the SYSIN data set specify that the data set PROGRAM.MASTER.LOADLIB is to be copied to the output data sets without being reblocked. If PROGRAM.MASTER.LOADLIB had not been reblocked to the smaller block size, this step would end unsuccessfully.

Example 13: Convert a Partitioned Data Set to a PDSE

In this example, a partitioned data set is converted to a PDSE.

```

//CONVERT JOB ...
//STEP1 EXEC PGM=IEBCOPY
//SYSPRINT DD SYSOUT=A
//SYSUT1 DD DSN=PDSESET,DISP=SHR,DSNTYPE=PDS
//SYSUT2 DD DSN=PDSESET,LIKE=PDSESET,DSNTYPE=LIBRARY,
// DISP=(NEW,CATLG),STORCLAS=SCLASX,DATACLAS=DCLASY

```

The control statements are as follows:

- SYSUT1 DD defines the input data set, PDS, which is a partitioned data set. The DSNTYPE keyword has no effect because it is an existing data set.
- SYSUT2 DD defines the output data set, PDSE, which is a partitioned data set extended. This new data set will be SMS-managed because it has a storage class.

The LIKE parameter indicates that the DCB and SPACE attributes for PDSESET are to be copied from PDSESET. The DSNTYPE parameter defines the new data set as a PDSE rather than as a partitioned data set. DATACLAS=DCLASY identifies the PPDSE as a program object PDSE with undefined logical record length.

The Storage Management Subsystem chooses an appropriate volume for the allocation, based on how SCLASX was defined.

- Since the ddnames "SYSUT1" and "SYSUT2" are used to define the input and output data sets, no SYSIN data set is required.

Example 14: Copy Groups from a PDSE to a PDSE

In this example, members and their aliases (groups) are copied from a PDSE to a PDSE (full data set copy). See [“Copying Program Objects \(COPYGRP and COPYGROUP Statements\)”](#) on page 47 for information about copying groups.

```
//CPYGRP   JOB    ...
//STEP1    EXEC  PGM=IEBCOPY
//SYSPRINT DD    SYSOUT=A
//DDIN     DD    DSN=PDSESETA,DISP=SHR
//DDOUT    DD    DSN=PDSESETB,LIKE=PDSESETA,DSNTYPE=LIBRARY,
//          DISP=(NEW,CATLG)
//SYSUT3   DD    UNIT=SYSDA,SPACE=(TRK,(1,1))
//SYSIN    DD    *
GROUPCPY   COPYGRP INDD=DDIN,OUTDD=DDOUT
/*
```

The control statements are as follows:

- DDIN DD defines the input data set, PDSESETA, which is a partitioned data set extended.
DDOUT DD defines the output data set, PDSESETA, which is a partitioned data set extended.
The LIKE subparameter indicates that the DCB and SPACE attributes for PDSESETB are to be copied from PDSESETA. The DSNTYPE subparameter defines the new data set as a PDSE.
The Storage Management Subsystem chooses an appropriate volume for the allocation.
- SYSUT3 DD defines a temporary spill data set.
- SYSIN DD defines the control data set, which follows in the input stream. The data set contains a COPYGRP statement, an INDD statement, and an OUTDD statement.
- COPYGRP indicates the start of the copy operation.
The INDD parameter shows PDSESETA as the input data set.
The OUTDD parameter shows PDSESETB as the output data set.

Example 15: Copy Groups from a PDSE to a PDSE with Replace

In this example, members and their aliases are copied in groups from a PDSE to a PDSE with the replace (R) option. See [“Replacing Program Objects”](#) on page 48 for information about replacing groups with COPYGRP.

```
//CPYGRP   JOB    ...
//STEP1    EXEC  PGM=IEBCOPY
//SYSPRINT DD    SYSOUT=A
//DDIN     DD    DSN=PDSESETA,DISP=SHR
//DDOUT    DD    DSN=PDSESETB,LIKE=PDSESETA,DSNTYPE=LIBRARY,
//          DISP=(NEW,CATLG)
//SYSUT3   DD    UNIT=SYSDA,SPACE=(TRK,(1,1))
//SYSIN    DD    *
GROUPCPY   COPYGRP INDD=((DDIN,R)),OUTDD=DDOUT
/*
```

The control statements are as follows:

- DDIN DD defines the input data set, PDSE, which is a partitioned data set extended.
DDOUT DD defines the output data set, PDSE, which is a partitioned data set extended.
The LIKE parameter indicates that the DCB and SPACE attributes for PDSESETB are to be copied from PDSESETA.
The DSNTYPE parameter defines the new data set as a PDSE.
The Storage Management Subsystem chooses an appropriate volume for the allocation.
- SYSUT3 DD defines a temporary spill data set.

- SYSIN DD defines the control data set, which follows in the input stream. The data set contains a COPYGRP statement, an INDD statement, and an OUTDD statement.
- COPYGRP indicates the start of the copy operation.

The ((INDD,R)) parameter shows PDSESETA as the input data set containing members to replace members with the same name in PDSESETB.

The OUTDD parameter shows PDSESETB as the output data set.

Example 16: Copy a Selected Group from a PDSE to a PDSE

In this example, a selected member and its aliases are copied from a PDSE to a PDSE. Either the member's name or a maximum of eight characters can be given on the SELECT statement. See [“Copying Program Objects \(COPYGRP and COPYGROUP Statements\)” on page 47](#) for information about selecting groups on COPYGRP.

```
//COPYGRP      JOB      ...
//STEP1        EXEC    PGM=IEBCOPY
//SYSPRINT     DD      SYSOUT=A
//DDIN         DD      DSN=PDSESETA,DISP=SHR
//DDOUT        DD      DSN=PDSESETB,LIKE=PDSESETA,DSNTYPE=LIBRARY,
//              DISP=(NEW,CATLG)
//SYSUT3       DD      UNIT=SYSDA,SPACE=(TRK,(1,1))
//SYSIN        DD      *
GROUPCPY      COPYGRP  INDD=DDIN,OUTDD=DDOUT
              SELECT   MEMBER=(ALIAS001)
/*
```

The control statements are as follows:

- DDIN DD defines the input data set, PDSE, which is a partitioned data set extended.
- DDOUT DD defines the output data set, PDSE, which is a partitioned data set extended.
- All PDSEs must be managed by the Storage Management Subsystem.
- The LIKE parameter indicates that the DCB and SPACE attributes for PDSESETB are to be copied from PDSESETA.
- The DSNTYPE parameter defines the new data set as a PDSE.
- The Storage Management Subsystem chooses an appropriate volume for the allocation.
- SYSUT3 DD defines a temporary spill data set.
- SYSIN DD defines the control data set, which follows in the input stream. The data set contains a COPYGRP statement, an INDD statement, and an OUTDD statement.
- COPYGRP indicates the start of the copy operation.
- The INDD parameter shows PDSESETA as the input data set.
- The OUTDD parameter shows PDSESETB as the output data set.
- The SELECT statement indicates that a group that has the alias ALIAS001 is to be selected from the input data set (PDSESETA) and copied to the output data set (PDSESETB).

Example 17: Copy Selected Members and their Aliases from a PDS to a PDS

In this example, selected members and their aliases are copied from a PDS to a PDS, while an EXCLUDE statement is used to prevent other members and their aliases from being copied. The SELECT and EXCLUDE statements can be used only if the operation is a COPYGROUP and member name filter pattern masking is used for both. Refer to the descriptions of the SELECT and EXCLUDE statements for details in using member name filter pattern masking.


```

//CPYGROUP JOB ...
//STEP1     EXEC PGM=IEBCOPY
//SYSPRINT  DD SYSOUT=*
//DDIN      DD DISP=SHR,DSN=PDSA
//DDOUT     DD DISP=(NEW,CATLG),DSN=PDSB,LIKE=PDSA,DSNTYPE=PDS
//SYSIN     DD *
CPYLBL1     COPYGROUP INDD=DDIN,OUTDD=DDOUT
            SELECT MEMBER=(ABCDEFGH,BCD*,%D*,GHIJKLMN)
            EXCLUDE MEMBER=(DEF*,EFGHIJKL,%GHI*)
/*

```

The control statements are as follows:

- DDIN DD defines the input data set, PDSA, a partitioned data set with the following members:
 1. ABCDEFGH
 2. BCDEFGHI
 3. CDEFGHIJ
 4. DEFGHIJK
 5. EFGHIJKL
 6. FGHIJKLM
 7. GHIJKLMN
- DDOUT DD defines the output data set, PDSB, a partitioned data set.
- The LIKE parameter indicates that the DCB and SPACE attributes for PDSB are to be copied from PDSA.
- The DSNTYPE parameter defines the data set as a PDS.
- The Storage Management Subsystem chooses an appropriate volume for the newly allocated data set.
- SYSIN DD defines the control data set, which follows in the input stream. The data set contains a COPYGROUP statement, a SELECT statement and an EXCLUDE statement.
- COPYGROUP indicates the start of a group copy operation.
- The INDD parameter shows PDSA as the input data set.
- The OUTDD parameter shows PDSB as the output data set.
- The SELECT statement is using filter pattern masking and indicates that any member matching the masking pattern and all of its aliases will be selected from the input dataset and copied to the output dataset. In this example the members ABCDEFGH, BCDEFGHI, CDEFGHIJ, and GHIJKLMN and their aliases will be copied.
- The EXCLUDE statement can be used for filter pattern masking for a COPYGROUP operation only when a filter masking SELECT statement is present. This will exclude the matched members and all of their aliases from being copied to the output dataset. In this example the members DEFGHIJK, EFGHIJKL, and FGHIJKLM and their aliases will not be copied.

Example 18: Copy Selected Members from a PDS to a PDS

In this example, selected members are copied from a PDS to a PDS. This example differs from Example 16 in that COPYGROUP allows members to be copied along with their aliases from a PDS to a PDS, whereas COPYGRP only copies aliases when one or both of the input and output data sets is a PDSE. If both are PDSs it will result in a COPY operation. See [“Copying Program Objects \(COPYGRP and COPYGROUP Statements\)”](#) on page 47 for information about selecting groups with COPYGROUP and the differences between the COPYGROUP and COPYGRP statements.

```

//CPYGROUP JOB ...
//STEP1     EXEC PGM=IEBCOPY
//SYSPRINT  DD SYSOUT=*
//DDIN      DD DISP=SHR,DSN=PDSA,DSNTYPE=PDS
//DDOUT     DD DSN=PDSB,LIKE=PDSA,DSNTYPE=PDS,DISP=(NEW,CATLG)
//SYSIN     DD *
CPYLBL2     COPYGROUP INDD=DDIN,OUTDD=DDOUT
            SELECT MEMBER=(MEMBER02)

```

```
SELECT MEMBER=(MEMBER03)  
/*
```

The control statements are as follows:

- DDIN DD defines the input data set, PDSA, which is a partitioned data set.
- DDOUT DD defines the output data set, PDSB, which is a partitioned data set.
- The LIKE parameter indicates that the DCB and SPACE attributes for PDSB are to be copied from PDSA.
- The DSNTYPE parameter defines the new data set as a PDS.
- The Storage Management Subsystem chooses an appropriate volume for the allocation.
- SYSIN DD defines the control data set, which follows in the input stream. The data set contains a COPYGROUP statement, and two SELECT statements.
- COPYGROUP indicates the start of a group copy operation.
- The INDD parameter shows PDSA as the input data set.
- The OUTDD parameter shows PDSB as the output data set.
- The SELECT statement indicates that the member MEMBER02, MEMBER03, and all their aliases are to be selected from the input data set (PDSA) and copied to the output data set (PDSB).

Chapter 5. IEBDG (Test Data Generator) Program

IEBDG is a data set utility that is used to provide a *pattern* of test data to be used as a programming debugging aid. This pattern of data can then be analyzed quickly for predictable results.

You can create a data set without supplying any input data, or you can specify a data set from which input data is to be taken. The data set that you create may have records of any format. Sequential data sets, or members of partitioned data sets or PDSEs, can be used for input or output.

IEBDG also gives you the option of specifying your own exit routine for monitoring each output record before it is written.

When you define the contents of a field, you must decide:

- Which pattern is to be placed initially in the defined field. You can use your own pattern, or a pattern that is supplied by IBM.
- What action, if any, is to be performed to alter the contents of the field after it is selected for each output record.

If IEBDG is invoked from an application program, you can dynamically allocate the data sets by issuing SVC 99 before calling IEBDG.

Selecting a Pattern

You can either use one of the IBM-supplied patterns, which are described here, or you can specify your own pattern.

IBM-Supplied Patterns

IBM supplies seven patterns:

- Alphanumeric
- Alphabetic
- Zoned decimal
- Packed decimal
- Binary number
- Collating sequence
- Random number

You may choose a pattern when defining the contents of a field. All patterns, except the packed decimal, binary, zoned decimal, and random number patterns, repeat in a given field, provided that the defined field length is sufficient to permit repetition. For example, the alphabetic pattern is:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEF...
```

Table 16 on page 95 shows the IBM-supplied patterns.

Table 16. IBM-supplied test data patterns

Type	Expressed in Hexadecimal	Expressed in Printable Characters
Alphanumeric	C1C2...E9, F0...F9	AB...Z, 0...9
Alphabetic	C1C2...E9	AB...Z

Table 16. IBM-supplied test data patterns (continued)

Type	Expressed in Hexadecimal	Expressed in Printable Characters
Zoned Decimal	F0F0...F9F9	00...99
Packed Decimal	0000...001C (Positive pattern) 0000...001D (Negative pattern)	Not applicable
Binary Number	00000001 (Positive pattern) FFFFFFFF (Negative pattern)	Not applicable
Collating Sequence	40...F9	␣\$.<(+ &!\$*);~-, %_>?:'=" A...Z 0...9
Random Number	Random hexadecimal digits	Not applicable

A 4-byte packed decimal, binary number, zoned decimal, or random number is right-aligned at the end of the defined field. The remainder of the field will contain fill characters for random numbers, zeros (X'00') for packed decimal and binary, and zoned zeros (X'F0') for decimal patterns.

You can specify a starting character when defining an alphanumeric, alphabetic, or collating-sequence field. For example, a 10-byte alphabetic field for which "H" is specified as the starting character would appear as:

```
HIJKLMNOPQ
```

The same 10-byte alphabetic field with no specified starting character would appear as:

```
ABCDEFGHIJ
```

You can specify a mathematical sign when defining a packed decimal or binary field. If no sign is specified, the field is assumed to be positive.

User-Specified Patterns

Instead of selecting an IBM-supplied pattern, you may want to specify your own pattern to be placed in the defined field. You can provide:

- A character string
- A decimal number to be converted to packed decimal by IEBDG
- A decimal number to be converted to binary by IEBDG

When you supply a pattern, a pattern length must be specified that is equal to or less than the specified field length. A character pattern is left-aligned at the start of a defined field; a decimal number that is converted to packed decimal or to binary is right-aligned at the end of a defined field.

You can initially fill a defined field with either a character or a hexadecimal digit. For example, the 10-byte pattern "BADCFEHGJI" is to be placed in a 15-byte field. The character "2" is to be used to pad the field. The result is BADCFEHGJI22222. (If no fill character is provided, the remaining bytes contain binary zeros.) The fill character, if specified, is written in each byte of the defined field before the inclusion of any pattern.

Modifying Fields in a Record

You can use IEBDG to change the contents of a field. You can select one of these actions to change a field after it is included in each applicable output record:

- Ripple
- Shift left
- Shift right
- Truncate left
- Truncate right
- Fixed
- Roll
- Wave

Figure 12 on page 97 shows the effects of each of the actions on a 6-byte alphabetic field. Note that the roll and wave actions are applicable only when you use one of your own patterns. In addition, the result of a ripple action depends on which type of pattern (IBM-supplied or user-supplied) is used.

Ripple-user-supplied pattern	Ripple-IBM-supplied pattern	Shift left	Shift right		
ABCDEF	ABCDEF	ABCDEF	ABCDEF		
BCDEFA	BCDEFG	BCDEF	ABCDE		
CDEFAB	CDEFGH	CDEF	ABCD		
DEFABC	DEFGHI	DEF	ABC		
EFABCD	EFGHIJ	EF	AB		
FABCDE	FGHIJK	F	A		
ABCDEF	GHIJKL	ABCDEF	ABCDEF		
BCDEFA	HIJKLM	BCDEF	ABCDE		

Truncate left	Truncate right	Fixed	Roll-user-supplied pattern	Wave-user-supplied pattern
ABCDEF	ABCDEF	ABCDEF	AAA	AAA
BCDEF	ABCDE	ABCDEF	AAA	AAA
CDEF	ABCD	ADCDEF	AAA	AAA
DEF	ABC	ABCDEF	AAA	AAA
EF	AB	ABCDEF	AAA	AAA
F	A	ABCDEF	AAA	AAA
ABCDEF	ABCDEF	ABCDEF	AAA	AAA
BCDEF	ABCDE	ABCDEF	AAA	AAA

Figure 12. IEBDG actions

If no action is selected, or if the specified action is not compatible with the format, the fixed action is assumed by IEBDG.

Input and Output

IEBDG uses the following input:

- An input data set that contains records to be used in the construction of an output data set or member of a partitioned data set or PDSE. The input data sets are optional; that is, output records can be created entirely from utility control statements.
- A control data set that contains any number of sets of utility control statements.

IEBDG produces the following output:

- An output data set that is the result of the IEBDG operation. One output data set is created by each set of utility control statements included in the job step.
- A message data set that contains informational messages, the contents of utility control statements, and any error messages.

Input and output data sets may be sequential, or members of a partitioned data set or PDSE. BDAM and VSAM are not supported.

Related reading: For information about IEBDG return codes, see [Appendix A, “Invoking Utility Programs from an Application Program,”](#) on page 315.

Control

IEBDG is controlled by job and utility control statements. The job control statements process IEBDG and define the data sets used and produced by IEBDG. Utility control statements are used to control the functions of the program and to define the contents of the output records.

Job Control Statements

Table 17 on page 98 shows the job control statements for IEBDG.

Both input and output data sets can contain fixed, variable, or undefined records.

Table 17. Job control statements for IEBDG

Statement	Use
JOB	Starts the job.
EXEC	Specifies the program name (PGM=IEBDG) or, if the job control statements reside in a procedure library, the procedure name.
SYSPRINT DD	Defines a sequential data set for messages. The data set can be written on a system output device, a tape volume, or a DASD volume.
anyname1 DD	Defines an optional input data set or member of a data set. Any number of these statements (each having a ddname different from all other ddnames in the job step) can be included in the job step.
anyname2 DD	Defines an output data set or member of a data set. Any number of these DD statements can be included per job step; however, only one statement is applicable per set of utility control statements.
SYSIN DD	Defines the control data set, which contains the utility control statements and, optionally, input records. The data set normally resides in the input stream; however, it can be defined as a sequential data set or as a member of a partitioned data set or PDSE.

EXEC Statement

The EXEC statement can include an optional PARM parameter to specify the number of lines to be printed between headings in the message data set.

Table 18 on page 99 shows the syntax of the EXEC statement:

Table 18. Syntax of EXEC statement		
Label	Statement	Parameters
// [stepname]	EXEC	PGM=IEBDG[, PARM= ' LINECT=nnnn ']

where:

PGM=IEBDG

specifies that you want to run the IEBDG program.

PARM='LINECT=nnnn'

specifies the number of lines to be printed per page on the output listing. The number is a 4-digit decimal number from 0000 to 9999.

If PARM is omitted, 58 lines are printed between headings (unless a channel 12 punch is encountered in the carriage control tape, in which case a skip to channel 1 is performed and a heading is printed).

SYSPRINT DD Statement

If the SYSPRINT DD statement is omitted, no messages are written. The block size for the SYSPRINT data set must be a multiple of 121. Any blocking factor can be specified.

anyname1 DD Statement

You can use any number of data sets or members as input for IEBDG. You must code a separate DD statement for each data set or member.

You cannot, for instance, use two members of a partitioned data set by only coding one DD statement for the data set. If the input data set is a *member* of a partitioned data set or PDSE, you must code the DD statement so that it refers to that member. Do this by coding the data set name parameter as `DSNAME=datasetname(membername)`.

You can also specify a sequential data set as an input data set. If the data set is sequential, it can be basic format, large format, extended format, compressed format, a UNIX file, a TSO terminal or on tape. The data set maximum block size cannot exceed 32760 bytes.

anyname2 DD Statement

You can create any number of data sets in one use of IEBDG. You must code a separate DD statement for each data set or member you are creating. The data sets or members must be new.

If you are creating a new member of a partitioned data set or PDSE, the DD statement must refer to that specific member. Do this by coding the data set name parameter as `DSNAME=datasetname(membername)`.

You can also create sequential data sets. If the data set is sequential, it can be basic format, large format, extended format, compressed format, a UNIX file, a TSO terminal or on tape. If you identify the same sequential data set for input and output in one operation, the utility might destroy some input data before it is read. This might cause I/O errors. They can be the same member of a partitioned data set because the utility writes the member in a different area of the data set. The data set maximum block size cannot exceed 32760 bytes.

Refer to [z/OS DFSMS Using Data Sets](#) for information on estimating space allocations.

SYSIN DD Statement

The block size for the SYSIN data set must be a multiple of 80. Any blocking factor can be specified, but the block size cannot exceed 32760 bytes.

Utility Control Statements

IEBDG is controlled by the utility control statements that are shown in [Table 19 on page 100](#).

Table 19. IEBDG utility control statements

Statement	Use
DSD	Specifies the ddnames of the input and output data sets. One DSD statement must be included for each set of utility control statements.
FD	Defines the contents and lengths of fields to be used in creating output records.
REPEAT	Specifies the number of times a CREATE statement or a group of CREATE statements are to be used in generating output records.
CREATE	Defines the contents of output records.
END	Marks the end of a set of IEBDG utility control statements.

Any number of sets of control statements can appear in a single job step. Each set defines one data set. A set of control statements contains one DSD statement, any number of FD, CREATE and REPEAT statements, and one END statement when INPUT is omitted from the FD statements.

General continuation requirements for utility control statements are described in [“Continuing utility control statements” on page 8](#).

DSD Statement

The DSD statement marks the beginning of a set of utility control statements and specifies the data sets that IEBDG is to use as input. The DSD statement can be used to specify one output data set and any number of input data sets for each application of IEBDG.

The syntax of the DSD statement is:

Label	Statement	Parameters
[<i>label</i>]]	DSD	OUTPUT=(<i>ddname</i>) [, INPUT=(<i>ddname1</i> [, <i>ddname2</i>] [, . . .])]

where:

OUTPUT=(*ddname*)

specifies the ddname of the DD statement defining the output data set.

INPUT=(*ddname1* [, *ddname2*] [, . . .])

specifies the ddnames of a DD statements defining data sets used as input to the program.

Any number of data sets can be included as input—that is, any number of ddnames referring to corresponding DD statements can be coded. Whenever ddnames are included on a continuation record, they must begin in column 4.

The ddname SYSIN must not be coded in the INPUT parameter on the DSD control statement. Each ddname should not appear more than once on any control statement.

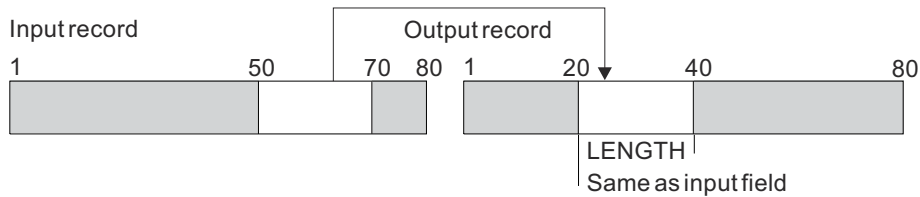
Each parameter should appear no more than once on any DSD statement.

FD Statement

The FD statement defines the contents and length of a field that will be used subsequently by a CREATE statement (or statements) to form output records. A defined field within the input logical record may be selected for use in the output records if it is referred to, by name, by a subsequent CREATE statement.

[Figure 13 on page 101](#) shows how the FD statement is used to specify a field in an input record to be used in output records. The first part of the figure shows that a field in the input record beginning at byte 50 is

selected for use in the output record. The other part of the figure shows that the field is to be placed at byte 20 in the output record.



FD NAME=FIELD1,LENGTH=20,STARTLOC=20,FROMLOC=50,INPUT=INSET

Figure 13. Field selected from the input record for use in the output record

When selecting fields from an input record (FD INPUT=ddname), the field must be defined by an FD statement within each set of utility control statements. In that case, defined fields for field selection are not usable across sets of utility control statements; such an FD record may be duplicated and used in more than one set of utility control statements within the job step.

You can also indicate that a numeric field is to be modified after it has been referred to *n* times by a CREATE statement or statements, that is, after *n* cycles, a modification is to be made. A modification will add a number you supply to a field.

The syntax of the FD statement is:

Label	Statement	Parameters
[label]	FD	NAME=name , LENGTH=length [, STARTLOC=starting-location] [, FILL={'character' X'nn'}] [, {FORMAT=pattern [, CHARACTER=character] PICTURE=length, {' character-string' P 'n' B }}] [, SIGN=sign] [, ACTION={ EX RO RP SL SR TL TR WV} [, INDEX=n [, CYCLE n] [, RANGE= n]] [, INPUT=ddname] [, FROMLOC=number]

where:

NAME=name

specifies the name of the field defined by this FD statement.

LENGTH=length

specifies the length, in bytes, of the defined field. For variable records, the sum of the field must be four less than the maximum record length. This is to account for the four-byte record descriptor word.

If the INPUT parameter is specified, the LENGTH parameter overrides the length of each input logical record. If the combination of FROMLOC and LENGTH values is longer than the input record, the result is unpredictable.

For ACTION=RP or WV, the length is limited to 16383 bytes. For ACTION=RO, the length is limited to 10922 bytes.

STARTLOC=starting-location

specifies a starting location (within all output records using this field) in which a field is to begin. For example, if the first byte of an output record is chosen as the starting location, the keyword is coded STARTLOC=1; if the tenth byte is chosen, STARTLOC=10 is coded.

Default: The field will begin in the first available byte of the output record (determined by the order of specified field names in the applicable CREATE statement). For variable records, the starting location is the first byte after the length descriptor.

FILL={'character'|X 'nn'}

specifies a value that is to be placed in each byte of the field in the output record before any other operation in the construction of the field. These values can be coded:

'character'

specifies a character that is to be converted to EBCDIC and placed in each byte of the output field.

X'nn'

specifies 2 hexadecimal digits (for example, FILL=X'40', or FILL=X'FF') to be placed in each byte of the output field.

Default: Binary zeros are placed in the output field.

FORMAT=pattern[,CHARACTER=character]

specifies an IBM-supplied pattern that is to be placed in the defined field. FORMAT must not be used when PICTURE is used. The values that can be coded are:

pattern

specifies the IBM-supplied patterns, as follows:

AL

specifies an alphabetic pattern.

AN

specifies an alphanumeric pattern.

BI

specifies a binary pattern.

CO

specifies a collating sequence pattern.

PD

specifies a packed decimal pattern.

RA

specifies a random binary pattern.

ZD

specifies a zoned decimal pattern.

CHARACTER=character

specifies the starting character of a field.

PICTURE=length,{ 'character-string'|P'n'|B'n'}

specifies the length and the contents of a user-supplied pattern. PICTURE must not be used when FORMAT is used. If both PICTURE and NAME are omitted, the fill character specified in the CREATE statement appears in each byte of applicable output records. These values can be coded:

length

specifies the number of bytes that the pattern will occupy. *Length* must be equal to or less than the LENGTH parameter value in the FD statement.

'character-string'

specifies a character string that is to be converted to EBCDIC and placed in the applicable records. The character string is left-aligned at the defined starting position. A character string may be broken in column 71, if you place a nonblank character in column 72 and continue the string in

column 4 of the next statement. The number of characters within the quotation marks must equal the number specified in the *length* subparameter.

P'n'

specifies a decimal number that is to be converted to packed decimal and right-aligned (within the boundaries of the defined length and starting byte) in the output records or defined field. The number of characters within the quotation marks must equal the number specified in the *length* subparameter.

B'n'

specifies a decimal number that is to be converted to binary and right-aligned (within the boundaries of the defined length and starting byte) in the output records or defined field. The number of characters within the quotation marks must equal the number specified in the *length* subparameter.

SIGN=sign

specifies a mathematical sign (+ or -), to be used when defining a packed decimal or binary field.

Default: Positive (+).

ACTION={FX|RO|RP|SL|SR|TL|TR|WV}

specifies how the contents of a defined field are to be altered (if at all) after the field's inclusion in an output record.

. These values can be coded:

FX

specifies that the contents of a defined field are to remain fixed after the field's inclusion in an output record.

FX is the default.

RO

specifies that the contents of a defined field are to be rolled after the field's inclusion in an output record. The pattern ("picture") is moved to the left by one byte for each output record, until the first nonblank character of the picture is in the first byte of the field. Then, the picture is moved to the right one byte for each output record, until it returns to its original position in the field.

RO can be used only for a user-defined pattern. For RO to be effective, the picture length must be less than the field length.

RP

specifies that the contents of a defined field are to be rippled after the field's inclusion in an output record.

SL

specifies that the contents of a defined field are to be shifted left after the field's inclusion in an output record.

SR

specifies that the contents of a defined field are to be shifted right after the field's inclusion in an output record.

TL

specifies that the contents of a defined field are to be truncated left after the field's inclusion in an output record.

TR

specifies that the contents of a defined field are to be truncated right after the field's inclusion in an output record.

WV

specifies that the contents of a defined field are to be waved after the field's inclusion in an output record. The pattern ("picture") is moved to the left by one byte for each output record, until the first nonblank character of the picture is in the first byte of the field. Then the character string is reset to its original position.

WV can be used only for a user-defined pattern. For WV to be effective, the picture length must be less than the field length.

INDEX=*n*[,CYCLE=*n*] [,RANGE=*n*]

specifies a decimal number to be added to this field whenever a specified number of records have been written. INDEX is valid only with FORMAT patterns ZD, PD and BI, or PICTURE patterns P'*n*' and *n* . Additional values can be coded:

CYCLE=*n*

specifies a number of output records (to be written as output or made available to an exit routine) that are treated as a group by the INDEX keyword. Whenever this field has been used in the construction of the specified number of records, it is modified as specified in the INDEX parameter. For example, if CYCLE=3 is coded, output records may appear as 111 222 333 444, and so forth. This parameter can be coded only when INDEX is coded.

RANGE=*n*

specifies an absolute value which the contents of this field can never exceed. If an index operation tries to exceed the specified absolute value, the contents of the field as of the previous index operation are used.

Default: No indexing is performed. If CYCLE is omitted and INDEX is coded, a CYCLE value of 1 is assumed; that is, the field is indexed after each inclusion in a potential output record.

INPUT=*ddname*

specifies the ddname of a DD statement defining a data set used as input for field selection. Only a portion of the record described by the FD statement will be placed in the output record. If the record format of the output data set indicates variable-length records, the position within the output record will depend upon where the last insertion into the output record was made unless STARTLOC is specified.

The ddname SYSIN must not be coded in the INPUT parameter on the FD control statement. Each ddname should not appear more than once on any control statement.

A corresponding ddname must also be specified in the associated CREATE statement in order to have the input records read.

FROMLOC=*number*

specifies the location of the selected field within the input logical record. The number represents the position in the input record. If, for example, FROMLOC=10 is coded, the specified field begins at the tenth byte; if FROMLOC=1 is coded, the specified field begins at the first byte. (For variable-length records, significant data begins in the first byte after the 4-byte length descriptor.)

When retrieving data sets with fixed or fixed-blocked record formats and RKP>0, the record consists of the key plus the data with embedded key. To copy the entire record, the output logical record length has to be the input logical record length plus the key length. If only the data (which includes the embedded key) is to be copied, set FROMLOC equal to the keylength.

Default: The start of the input record.

Related reading:

- For information about starting characters, see [“IBM-Supplied Patterns” on page 95](#) .
- For information about system actions compatible with FORMAT and PICTURE values, see [Table 20 on page 105](#).
- For examples of IEBDG ACTION patterns, see [Figure 12 on page 97](#).

Usage notes for FD

Some of the FD keywords do not apply when you select certain patterns or pictures; for example, the INDEX, CYCLE, RANGE, and SIGN parameters are used only with numeric fields. [Table 20 on page 105](#) shows which IEBDG keywords can be used with which patterns. Each keyword should appear no more than once on any FD statement.

Table 20. Compatible IEBDG Operations

FORMAT/PICTURE Value	Compatible Parameters
FORMAT=AL (alphabetic) FORMAT=AN (alphanumeric) FORMAT=CO (collating sequence)	ACTION=SL (shift left) ACTION=SR (shift right) ACTION=TL (truncate left) ACTION=TR (truncate right) ACTION=FX (fixed) ACTION=RP (ripple)
FORMAT=ZD (zoned decimal) FORMAT=PD (packed decimal) FORMAT=BI (binary)	INDEX= <i>n</i> CYCLE= <i>n</i> RANGE= <i>n</i> SIGN= <i>n</i> ¹ SIGN= <i>n</i> (1)
PICTURE=P' <i>n</i> ' (packed decimal) PICTURE= <i>n</i> (binary)	INDEX= <i>n</i> CYCLE= <i>n</i> RANGE= <i>n</i> SIGN= <i>n</i>
PICTURE='string' (EBCDIC)	ACTION=SL (shift left) ACTION=SR (shift right) ACTION=TL (truncate left) ACTION=TR (truncate right) ACTION=FX (fixed) ACTION=RP (ripple) ACTION=WV (wave) ACTION=RO (roll)

Note: (1) Zoned decimal numbers (ZD) do not include a sign.

REPEAT Statement

The REPEAT statement specifies the number of times a CREATE statement or group of CREATE statements is to be used repetitively in the generation of output records. The REPEAT statement precedes the CREATE statements to which it applies. The syntax of the REPEAT statement is:

Label	Statement	Parameters
[<i>label</i>]	REPEAT	QUANTITY= <i>number</i> [, CREATE= <i>number</i>]

where:

QUANTITY=number

specifies the number of times the defined group of CREATE statements is to be used repetitively. This number cannot exceed 65,535.

CREATE=number

specifies the number of CREATE statements to be included in the group.

Default: Only the first CREATE statement is repeated.

CREATE Statement

The CREATE statement defines the contents of a record to be written directly as an output record or to be made available to an exit routine you supply. An output record is constructed by referring to previously

defined fields by name or by providing a pattern ("picture") to be placed in the record. You can generate multiple records with a single CREATE statement.

An output record is constructed in the following order:

1. A fill character, specified or default (binary zero), is initially loaded into each byte of the output record.
2. If the INPUT operand is specified on the CREATE statement, and not on an FD statement, the input records are left-aligned in the corresponding output record.
3. If the INPUT operand specifies a *ddname* in any FD statement, only the fields described by the FD statements are placed in the output record.
4. FD fields, if any, are placed in the output record in the order of the appearance of their names in the CREATE statement. The location of the fields in the output record depends upon whether the field has a specified starting location (STARTLOC).

For instance, if you do not specify a starting location for any field, the fields will be placed in order in the output record, starting at the first position of the output record. [Figure 14 on page 106](#) shows the addition of field X to two different records. In record 1, field X is the first field referred to by the CREATE statement; therefore, field X begins in the first byte of the output record. In record 2, two fields, field A and field B, have already been referred to by a CREATE statement; field X, the next field referred to, begins immediately after field B. Field X does not have a special starting location in this example.

Record 1: **CREATE NAME=X**

1	21	80
Field X		

Record 2: **CREATE NAME=(A,B,X)**

1	21	41	61	80
Field A	Field B	Field X		

Figure 14. Default placement of fields within an output record using IEBDG

If FD fields have starting locations specified explicitly, each field is placed in the output record beginning at the location specified. These fields are written to the output record in the order they appear on the CREATE statement. Thus, if a field has the same starting location as another field, or has a starting location that overlaps another field, the field that appears later on the CREATE statement is the field whose contents occupy those positions. [Figure 15 on page 106](#) shows an example of two fields with specified starting locations that result in an overlap of the fields.

FD NAME=FIELD1,LENGTH=30,STARTLOC=20,...
FD NAME=FIELD2,LENGTH=40,STARTLOC=40,...
CREATE NAME=(FIELD1,FIELD2)

1	20	40	50	80
	FIELD1	FIELD2		

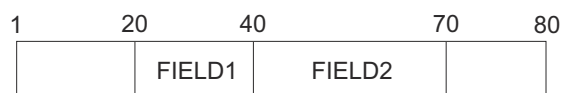
Figure 15. Placement of fields with specified output locations

If some fields specify a starting location and others do not, the location of the fields will depend on the order in which you specify them in the CREATE statement. A field with an unspecified starting location will begin immediately following the last record written.

For instance, [Figure 16 on page 107](#) shows how two fields, one with a starting location specified and one without, are written to an output record. Record 1 shows FIELD1, with a specified starting location, written first. FIELD1 starts at location 20, and occupies 20 bytes. FIELD2 is then written next, and so begins at position 40. Record 2 shows FIELD2, with an unspecified starting location, written first. FIELD2 is placed starting at the first position of the output record. FIELD1 is then placed at position 20, even though it overlaps FIELD2, which has a length of 30.

FD NAME=FIELD1,LENGTH=20,STARTLOC=20,...
FD NAME=FIELD2,LENGTH=30,...

Record 1: **CREATE NAME=(FIELD1,FIELD2)**



Record 2: **CREATE NAME=(FIELD2,FIELD1)**

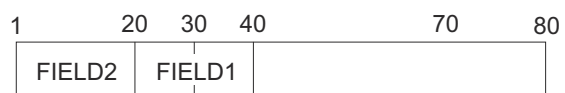


Figure 16. Placement of fields with only some output locations specified

5. A CREATE statement picture, if any, is placed in the output record.

[Figure 17 on page 107](#) shows three ways in which output records can be created from utility control statements.

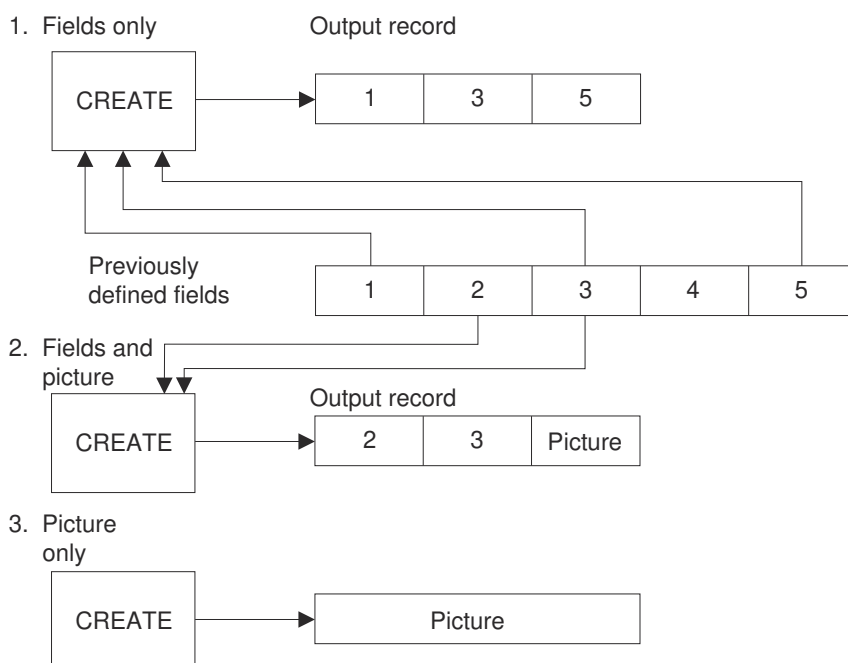


Figure 17. Creating output records with utility control statements

When defining a picture in a CREATE statement, you must specify its length and starting location in the output record. The specified length must be equal to the number of specified alphabetic or

numeric characters. (When a specified decimal number is converted to packed decimal or binary, it is automatically right-aligned.)

You can use another data set as a source for input records for creating output records, or you can include input records in the input stream or SYSIN data set. Only one input data set can be used for an individual CREATE statement.

The syntax of the CREATE statement is:

Label	Statement	Parameters
[<i>label</i>]	CREATE	[QUANTITY= <i>n</i>] [, FILL={ <i>character</i> ' X'nn' '}] [, INPUT={ <i>ddname</i> SYSIN [({ <i>cccc</i> \$\$\$ <i>E</i> '})] }] [, PICTURE= <i>length,startloc</i> , { ' <i>character-string</i> ' P ' <i>n</i> ' B'n' }] [, NAME={ (<i>namelist</i>) (<i>namelist-or-copygroup</i>)) } [, EXIT= <i>routinename</i>]

where:

Note: Each keyword should appear no more than once on any CREATE statement.

QUANTITY=*n*

specifies the number of records that this CREATE statement is to generate; the contents of each record are specified by the other parameters. If both QUANTITY and INPUT are coded, and the quantity specified is greater than the number of records in the input data set, the number of records created is equal to the number of input records to be processed plus the generated data up to the specified number.

Default: If QUANTITY is omitted and INPUT is not specified, only one output record is created. If QUANTITY is omitted and INPUT is specified, the number of records created is equal to the number of records in the input data set.

If both QUANTITY and INPUT are coded, but the QUANTITY is less than the number of records in the input data set, then only the number of records specified by QUANTITY are written to the output data set.

FILL={ '*character*' | X'nn' }

specifies a value that is to be placed in each byte of the output record before any other operation in the construction of a record. These values can be coded:

'*character*'

specifies a character that is to be translated to EBCDIC and placed in each byte of the output record.

X'nn'

specifies 2 hexadecimal digits (for example, FILL=X'40', or FILL=X'FF') to be placed in each byte of the output record.

Default: Binary zeros (X'00') are placed in the output record.

INPUT={ddname|SYSIN [(cccc|\$\$\$E)]}

defines an input data set whose records are to be used in the construction of output records. If INPUT is coded, QUANTITY should also be coded, unless the remainder of the input records are all to be processed by this CREATE statement. If INPUT is specified in an FD statement referenced by this CREATE statement, there must be a corresponding ddname specified in the CREATE statement in order to get the input records read. These values can be coded:

ddname

specifies the ddname of a DD statement defining an input data set.

SYSIN[(cccc|\$\$\$E)]

specifies that the SYSIN data set contains records (other than utility control statements) to be used in the construction of output records. If SYSIN is coded, the input records follow this CREATE statement (unless the CREATE statement is in a REPEAT group, in which case the input records follow the last CREATE statement of the group). The "cccc" value can be any combination of from 1 to 4 characters. If cccc is coded, the end of the input records is indicated by a record containing those characters beginning in column 1.

The default value for cccc is \$\$\$E. If you do not code a value for cccc when you use INPUT=SYSIN, you must use \$\$\$E to mark the end of the input records in SYSIN. The first three characters are dollar signs, which are X'5B'.

PICTURE=length,startloc, {'character-string' P'n'|B'n'}

specifies the length, starting position and the contents of a user-supplied pattern. If both PICTURE and NAME are omitted, the fill character specified in the CREATE statement appears in each byte of applicable output records. These values can be coded:

length

specifies the number of bytes that the pattern ("picture") will occupy. *Length* must be equal to or less than the LENGTH parameter value in the FD statement.

startloc

specifies a starting position (within any applicable output record) in which the picture is to begin.

'character-string'

specifies a character string that is to be placed in the applicable records. The character string is left-aligned at the defined starting position. A character string may be broken in column 71, if you place a nonblank character in column 72 and continue the string in column 4 of the next statement.

P'n'

specifies a decimal number that is to be converted to packed decimal and right-aligned (within the boundaries of the defined length and starting position) in the output records or defined field.

B'n'

specifies a decimal number that is to be converted to binary and right-aligned (within the boundaries of the defined length and starting position) in the output records or defined field.

NAME={(namelist)| (namelist-or-(copygroup))}

specifies the name or names of previously defined fields to be included in the applicable output records. If both NAME and PICTURE are omitted, the fill character specified in the CREATE statement appears in each byte of the applicable output record. These values can be coded:

(namelist)

specifies the name or names of a field or fields to be included in the applicable output records. Multiple field names must be separated with commas. Each field (previously defined in the named FD statement) is included in an output record in the order in which its name is encountered in the CREATE statement. If only one name is coded, the parentheses are optional.

(namelist-or-(copygroup))

specifies that some or all fields are to be copied in the output records; that is, selected fields are to appear in an output record more than once. The copied fields are specified as:

(COPY=*n*,name1[,name2] [...])

where *n* specifies that the fields indicated are to be treated as a group and copied *n* number of times in each output record produced by this CREATE statement. Any number of *copygroups* can be included with NAME. A maximum of 20 field names can be included in a *copygroup*.

The names of fields that are not to be copied can be specified with *copygroups* in the NAME parameter, either before, after, or between *copygroups*.

For example:

NAME=(NAME1,(COPY=2,NAME2),NAME3,(COPY=4,NAME4)).

EXIT=routinename

specifies the name of your routine that is to receive control from IEBDG before writing each output record.

Related reading: For information about specifying an exit routine with IEBDG, see [Appendix C, “Specifying User Exits with Utility Programs,”](#) on page 343.

END Statement

The END statement is used to mark the end of a set of utility control statements. Each set of control statements can pertain to any number of input data sets but only to a single output data set.

The syntax of the END statement is:

Label	Statement	Parameters
[<i>label</i>]	END	

IEBDG Examples

These examples illustrate some of the uses of IEBDG. You can use [Table 21 on page 110](#) as a quick-reference guide to IEBDG examples.

Table 21. IEBDG example directory

Operation	Data Set Organization	Device	Comments	Example
Create output records from utility control statements	Sequential	Disk	Blocked output.	“Example 3: Create Output Records from Utility Control Statements” on page 112
Create partitioned members from utility control statements	Partitioned	Disk	Blocked output. One set of utility control statements per member.	“Example 5: Create Records in Three Output Data Sets and Write them to Three Partitioned Data Set Members” on page 115
Modify records from partitioned members and input stream	Partitioned, Sequential	Disk	Reblocking is performed. Each block of output records contains ten modified partitioned input records and two input stream records.	“Example 4: Use Members and Input Records as Basis of Output Member” on page 113
Place binary zeros in selected fields.	Sequential	Tape	Blocked input and output.	“Example 1: Place Binary Zeros in Records Copied from Sequential Data Set” on page 111
Ripple alphabetic pattern	Sequential	Tape, Disk	Blocked input and output.	“Example 2: Ripple 10-byte Alphabetic Pattern” on page 111

Table 21. IEBDG example directory (continued)

Operation	Data Set Organization	Device	Comments	Example
Roll and wave user-supplied patterns	Sequential	Disk	Output records are created from utility control statements.	“Example 6: Construct Records with Your Own Patterns” on page 117

Examples that use **disk** or **tape** in place of actual device numbers must be changed before use. The actual device numbers depend on how your installation defines the devices to your system.

Example 1: Place Binary Zeros in Records Copied from Sequential Data Set

In this example, binary zeros are placed in two fields of 100 records copied from a sequential data set. After the operation, each record in the copied data set (OUTSET) contains binary zeros in locations 20 through 29 and 50 through 59.

```
//CLEAROUT JOB ...
//STEP1 EXEC PGM=IEBDG
//SYSPRINT DD SYSOUT=A
//SEQIN DD DSN=INSET,UNIT=tape,DISP=(OLD,KEEP),
//          DCB=(RECFM=FB,LRECL=80,BLKSIZE=800),
//          LABEL=(,NL),VOLUME=SER=222222
//SEQOUT DD DSN=OUTSET,UNIT=tape,DISP=(,KEEP),
//          DCB=(RECFM=FB,LRECL=80,BLKSIZE=800),
//          VOLUME=SER=222333,LABEL=(,NL)
//SYSIN DD *
DSD OUTPUT=(SEQOUT),INPUT=(SEQIN)
FD NAME=FIELD1,LENGTH=10,STARTLOC=20
FD NAME=FIELD2,LENGTH=10,STARTLOC=50
CREATE QUANTITY=100,INPUT=SEQIN,NAME=(FIELD1,FIELD2)
END
/*
```

The control statements are as follows:

- SEQIN DD defines a sequential input data set (INSET). The data set was originally written on a unlabeled tape volume.
- SEQOUT DD defines the test data set (OUTSET). The output records are identical to the input records, except for locations 20 through 29 and 50 through 59, which contain binary zeros at the completion of the operation.
- SYSIN DD defines the control data set, which follows in the input stream.
- DSD marks the beginning of a set of utility control statements and refers to the DD statements defining the input and output data sets.
- The first and second FD statements create two 10-byte fields (FIELD1 and FIELD2). Because no pattern is specified for these fields, each field contains the default fill of binary zeros. The fields are to begin in the 20th and 50th bytes of each output record.
- CREATE constructs 100 output records in which the contents of previously defined fields (FIELD1, FIELD2) are placed in their respective starting locations in each of the output records. Input records from data set INSET are used as the basis of the output records.
- END signals the end of a set of utility control statements.

Example 2: Ripple 10-byte Alphabetic Pattern

In this example, a 10-byte alphabetic pattern is rippled. At the end of the job step the first output record contains "ABCDEFGH IJ", followed by data in location 11 through 80 from the input record; the second record contains "BCDEFGHIJK" followed by data in locations 11 through 80, and so forth.

```
//RIPPLE JOB ...
//STEP1 EXEC PGM=IEBDG
//SYSPRINT DD SYSOUT=A
//SEQIN DD DSN=INSET,UNIT=tape,VOL=SER=222222,
//          DCB=(RECFM=FB,LRECL=80,BLKSIZE=800),DISP=(OLD,KEEP)
```

```
//SEQOUT DD DSNAME=OUTSET,UNIT=disk,VOLUME=SER=111111,
//          DCB=(RECFM=FB,LRECL=80,BLKSIZE=800),
//          DISP=(,KEEP),SPACE=(TRK,(10,10))
//SYSIN DD *
DSD OUTPUT=(SEQOUT),INPUT=(SEQIN)
FD NAME=FIELD1,LENGTH=10,FORMAT=AL,ACTION=RP,STARTLOC=1
CREATE QUANTITY=100,INPUT=SEQIN,NAME=FIELD1
END
/*
```

The control statements are as follows:

- SEQIN DD defines an input sequential data set (INSET). The data set was originally written on a standard labeled tape volume.
- SEQOUT DD defines the test output data set (OUTSET). Ten tracks of primary space and ten tracks of secondary space are allocated for the sequential data set on a disk volume.
- SYSIN DD defines the control data set, which follows in the input stream.
- DSD marks the beginning of a set of utility control statements and refers to the DD statements defining the input and output data sets.
- The FD statement creates a 10-byte field in which the pattern ABCDEFGHIJ is initially placed. The data is rippled after each output record is written.
- CREATE constructs 100 output records in which the contents of a previously defined field (FIELD1) are included. The CREATE statement uses input records from data set INSET as the basis of the output records.
- END signals the end of a set of utility control statements.

Example 3: Create Output Records from Utility Control Statements

In this example, output records are created entirely from utility control statements. Three fields are created and used in the construction of the output records. In two of the fields, alphabetic data is truncated; the other field is a numeric field that is incremented (indexed) by one after each output record is written. [Figure 18 on page 112](#) shows the contents of the output records at the end of the job step.

Field 1	Field 2	Fill	Field 3 (packed decimal)		
1	31	61	71	75	80
ABCDEFGHIJKLMNOPQRSTUVWXYZABCD	ABCDEFGHIJKLMNOPQRSTUVWXYZABCD	FF ... FF	00 ... 00	0123456789	0C
BCDEFGHIJKLMNOPQRSTUVWXYZABCD	ABCDEFGHIJKLMNOPQRSTUVWXYZABC	FF ... FF	00 ... 00	0123456789	1C
CDEFGHIJKLMNOPQRSTUVWXYZABCD	ABCDEFGHIJKLMNOPQRSTUVWXYZAB	FF ... FF	00 ... 00	0123456789	2C
DEFGHIJKLMNOPQRSTUVWXYZABCD	ABCDEFGHIJKLMNOPQRSTUVWXYZA	FF ... FF	00 ... 00	0123456789	3C
EFGHIJKLMNOPQRSTUVWXYZABCD	ABCDEFGHIJKLMNOPQRSTUVWXYZ	FF ... FF	00 ... 00	0123456789	4C

Figure 18. Output Records at job step completion

```
//UTLYONLY JOB ... 72
//STEP1 EXEC PGM=IEBDG
//SYSPRINT DD SYSOUT=A
//SEQOUT DD DSNAME=OUTSET,UNIT=disk,DISP=(,KEEP),
//          DCB=(RECFM=FB,LRECL=80,BLKSIZE=800),
//          SPACE=(TRK,(10,10)),VOLUME=SER=111111
//SYSIN DD DATA
DSD OUTPUT=(SEQOUT)
FD NAME=FIELD1,LENGTH=30,STARTLOC=1,FORMAT=AL,ACTION=TL
FD NAME=FIELD2,LENGTH=30,STARTLOC=31,FORMAT=AL,ACTION=TR
FD NAME=FIELD3,LENGTH=10,STARTLOC=71,PICTURE=10, X
P'1234567890',INDEX=1
CREATE QUANTITY=100,NAME=(FIELD1,FIELD2,FIELD3),FILL=X'FF'
END
/*
```

The control statements are as follows:

- SEQOUT DD defines the test output data set. Ten tracks of primary space and ten tracks of secondary space are allocated for the sequential data set on a disk volume.
- SYSIN DD defines the control data set, which follows in the input stream.
- DSD marks the beginning of a set of utility control statements and refers to the DD statement defining the output data set.
- FD defines the contents of three fields to be used in the construction of output records. The first field contains 30 bytes of alphabetic data to be truncated left after each output record is written. The second field contains 30 bytes of alphabetic data to be truncated right after each output record is written. The third field is a 10-byte field containing a packed decimal number (1234567890) to be increased by one after each record is written.
- CREATE constructs 100 output records in which the contents of previously defined fields (FIELD1, FIELD2, and FIELD3) are included. Note that after each record is written, FIELD1 and FIELD2 are restored to full width.
- END signals the end of a set of utility control statements.

Example 4: Use Members and Input Records as Basis of Output Member

In this example, two partitioned members and input records from the input stream are used as the basis of a partitioned output member. Each block of 12 output records contains 10 modified records from an input partitioned member and two records from the input stream. [Figure 19 on page 114](#) shows the contents of the output partitioned member at the end of the job step.

Input		Output Records
Department 21	(rightmost 67 bytes of INSET1(MEMBA) record 1)	1 1st block of 12
	•	•
	•	•
	•	•
	•	•
Department 21	(rightmost 67 bytes of INSET1(MEMBA) record 10)	10
Input record 1	from input stream	11
Input record 2	from input stream	12
Department 21	(rightmost 67 bytes of INSET1(MEMBA) record 11)	1 2nd block of 12
	•	•
	•	•
	•	•
	•	•
Department 21	(rightmost 67 bytes of INSET1(MEMBA) record 20)	10
Input record 3	from input stream	11
Input record 4	from input stream	12
	•	
	•	
	•	
Department 21	(rightmost 67 bytes of INSET1(MEMBA) record 91)	1 10th block of 12
	•	•
	•	•
	•	•
	•	•
Department 21	(rightmost 67 bytes of INSET1(MEMBA) record 100)	10
Input record 19	from input stream	11
Input record 20	from input stream	12
Department 21	(rightmost 67 bytes of INSET2(MEMBA) record 1)	1 11th block of 12
	•	•
	•	•
	•	•
	•	•
Department 21	(rightmost 67 bytes of INSET2(MEMBA) record 10)	10
Input record 21	from input stream	11
Input record 21	from input stream	12
	•	
	•	
	•	

Figure 19. Output partitioned member at job step completion

```

//MIX      JOB      ...
//STEP1    EXEC     PGM=IEBDG
//SYSPRINT DD      SYSOUT=A
//PARIN1   DD      DSNAME=INSET1(MEMBA),UNIT=disk,DISP=OLD,
//              DCB=(RECFM=FB,LRECL=80,BLKSIZE=800,DSORG=PS),
//              VOLUME=SER=111111
//PARIN2   DD      DSNAME=INSET2(MEMBA),UNIT=disk,DISP=OLD,
//              DCB=(RECFM=FB,LRECL=80,BLKSIZE=960,DSORG=PS),
//              VOLUME=SER=222222
//PAROUT    DD      DSNAME=PARSET(MEMBA),UNIT=disk,DISP=(,KEEP),
//              VOLUME=SER=333333,SPACE=(TRK,(10,10,5)),
//              DCB=(RECFM=FB,LRECL=80,BLKSIZE=960,DSORG=PO)
//SYSIN     DD      DATA
              DSD      OUTPUT=(PAROUT),INPUT=(PARIN1,PARIN2)
              FD        NAME=FIELD1,LENGTH=13,PICTURE=13,'DEPARTMENT 21'
              REPEAT    QUANTITY=10,CREATE=2
              CREATE    QUANTITY=10,INPUT=PARIN1,NAME=FIELD1
              CREATE    QUANTITY=2,INPUT=SYSIN

```

(input records 1 through 20)

```

$$$E
  REPEAT  QUANTITY=10,CREATE=2
  CREATE  QUANTITY=10,INPUT=PARIN2,NAME=FIELD1
  CREATE  QUANTITY=2,INPUT=SYSIN

(input records 21 through 40)

$$$E
  END
/*

```

The control statements are as follows:

- PARIN1 DD defines one of the input partitioned members.
- PARIN 2 DD defines the second of the input partitioned members. (Note that the members are from different partitioned data sets.)
- PAROUT DD defines the output partitioned member. This example assumes that the partitioned data set does not exist before the job step; that is, this DD statement allocates space for the partitioned data set.
- SYSIN DD defines the control data set, which follows in the input stream.
- DSD marks the beginning of a set of utility control statements and refers to the DD statements defining the input and output data sets.
- FD creates a 13-byte field in which the picture "DEPARTMENT 21" is placed.
- The first REPEAT statement indicates that the following group of two CREATE statements is to be repeated 10 times.
- The first CREATE statement creates 10 output records. Each output record is constructed from an input record (from partitioned data set INSET1) and from previously defined FIELD1.
- The second CREATE statement indicates that two records are to be constructed from input records included next in the input stream.
- The \$\$\$E record separates the input records from the REPEAT statement. The next REPEAT statement group is identical to the preceding group, except that records from a different partitioned member are used as input.
- END signals the end of a set of utility control statements.

Example 5: Create Records in Three Output Data Sets and Write them to Three Partitioned Data Set Members

Restriction: This example will not work if the data sets are system-managed or SMS data sets.

In this example, output records are created from three sets of utility control statements and written in three members in one partitioned data set. Four fields are created and used in the construction of the output records. In two of the fields (FIELD1 and FIELD3), alphabetic data is shifted. FIELD2 is fixed zoned decimal and FIELD4 is fixed alphanumeric. [Figure 20 on page 116](#) shows the partitioned data set members at the end of the job step.

MEMBA			
Field 1	Field 3	Field 2	Binary Zeros
1	31	51	71 80
ABCDEFGHIJKLMNQRSTUUVWZYABCD	ABCDEFGHIJKLMNQRST	00000000000000000001	f i l l
BCDEFGHIJKLMNQRSTUUVWZYABCD	ABCDEFGHIJKLMNQRST	00000000000000000001	f i l l
CDEFGHIJKLMNQRSTUUVWZYABCD	ABCDEFGHIJKLMNQRST	00000000000000000001	f i l l
DEFGHIJKLMNQRSTUUVWZYABCD	ABCDEFGHIJKLMNOPQ	00000000000000000001	f i l l

MEMBB			
Field 2	Field 3	Field 3	Field 3
1	21	41	61 80
00000000000000000001	ABCDEFGHIJKLMN	ABCDEFGHIJKLMN	ABCDEFGHIJKLMN
00000000000000000001	ABCDEFGHIJKLMNO	ABCDEFGHIJKLMNO	ABCDEFGHIJKLMNO
00000000000000000001	ABCDEFGHIJKLMN	ABCDEFGHIJKLMN	ABCDEFGHIJKLMN
00000000000000000001	ABCDEFGHIJKLM	ABCDEFGHIJKLM	ABCDEFGHIJKLM

MEMBC		
Field 4	Field 1	Binary Zeros
1	31	61 80
ABCDEFGHIJKLMNQRSTUUVWXYZ0123	EFGHIJKLMNQRSTUUVWZYABCD	f i l l
BCDEFGHIJKLMNQRSTUUVWXYZ0123	FGHIJKLMNQRSTUUVWZYABCD	f i l l
ABCDEFHIJKLMNQRSTUUVWXYZ0123	GHIJKLMNQRSTUUVWZYABCD	f i l l
ABCD EFGHIJKLMNQRSTUUVWXYZ0123	HIJKLMNQRSTUUVWZYABCD	f i l l

Figure 20. Partitioned data set members at job step completion

```

//UTSTS   JOB    ...
//STEP1   EXEC   PGM=IEBDG
//SYSPRINT DD   SYSOUT=A
//PAROUT1 DD   DSNAME=PARSET(MEMBA),UNIT=disk,DISP=(,KEEP),
//           VOLUME=SER=111111,SPACE=(TRK,(10,10,5)),
//           DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)
//PAROUT2 DD   DSNAME=PARSET(MEMBB),UNIT=AFF=PAROUT1,
//           DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)
//           DISP=OLD,VOLUME=SER=111111
//PAROUT3 DD   DSNAME=PARSET(MEMBC),UNIT=AFF=PAROUT1,
//           DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)
//           DISP=OLD,VOLUME=SER=111111
//SYSIN   DD   DATA
//           DSD   OUTPUT=(PAROUT1)
//           FD    NAME=FIELD1,LENGTH=30,FORMAT=AL,ACTION=SL
//           FD    NAME=FIELD2,LENGTH=20,FORMAT=ZD
//           FD    NAME=FIELD3,LENGTH=20,FORMAT=AL,ACTION=SR
//           FD    NAME=FIELD4,LENGTH=30,FORMAT=AN
//           CREATE QUANTITY=4,NAME=(FIELD1,FIELD3,FIELD2)
//           END
//           DSD   OUTPUT=(PAROUT2)
//           CREATE QUANTITY=4,NAME=(FIELD2,(COPY=3,FIELD3))
//           END
//           DSD   OUTPUT=(PAROUT3)
//           CREATE QUANTITY=4,NAME=(FIELD4,FIELD1)
//           END
/*

```

The control statements are as follows:

- PAROUT1 DD defines the first member (MEMBA) of the partitioned output data set. This example assumes that the partitioned data set does not exist before this job step; that is, this DD statement allocates space for the data set.
- PAROUT2 and PAROUT3 DD define the second and third members, respectively, of the output partitioned data set. Note that each DD statement specifies DISP=OLD and UNIT=AFF=PAROUT1.
- SYSIN DD defines the control data set that follows in the input stream.

- DSD marks the beginning of a set of utility control statements and refers to the DD statement defining the member applicable to that set of utility control statements.
- FD defines the contents of a field that is used in the subsequent construction of output records.
- CREATE constructs four records from combinations of previously defined fields.
- END signals the end of a set of utility control statements.

Example 6: Construct Records with Your Own Patterns

In this example, 10 fields containing user-supplied character patterns are used in the construction of output records. After a record is written, each field is rolled or waved, as specified in the applicable FD statement. [Figure 21 on page 117](#) shows the contents of the output records at the end of the job step.

FIELD1	FIELD2	FIELD3	FIELD4	FIELD5	FIELD6	FIELD7	FIELD8	FIELD9	FIELD10
AAAAA	BBBBB	A AA	BB B	AAA	CCCCC	DDDD	C CC	DD D	CCC
AAAAA	BBBBB	A AA	BB B	AAA	CCCCC	DDDD	C CC	DD D	CCC
AAAAA	BBBBB	A AA	BB B	AAA	CCCCC	DDDD	C CC	DD D	CCC
AAAAA	BBBBB	A AA	BB B	AAA	CCCCC	DDDD	C CC	DD D	CCC
AAAAA	BBBBB	A AA	BB B	AAA	CCCCC	DDDD	C CC	DD D	CCC
AAAAA	BBBBB	A AA	BB B	AAA	CCCCC	DDDD	C CC	DD D	CCC
AAAAA	BBBBB	A AA	BB B	AAA	CCCCC	DDDD	C CC	DD D	CCC
AAAAA	BBBBB	A AA	BB B	AAA	CCCCC	DDDD	C CC	DD D	CCC
AAAAA	BBBBB	A AA	BB B	AAA	CCCCC	DDDD	C CC	DD D	CCC
AAAAA	BBBBB	A AA	BB B	AAA	CCCCC	DDDD	C CC	DD D	CCC

Figure 21. Contents of output records at job step completion

```
//ROLLWAVE JOB ...
//STEP1 EXEC PGM=IEBDG
//SYSPRINT DD SYSOUT=A
//OUTSET DD DSNAME=SEQSET,UNIT=disk,DISP=(,KEEP),
//          VOLUME=SER=SAMP,SPACE=(TRK,(10,10)),
//          DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)
//SYSIN DD *
DSD OUTPUT=(OUTSET)
FD NAME=FIELD1,LENGTH=8,PICTURE=8,'AAAAA',ACTION=RO
FD NAME=FIELD2,LENGTH=8,PICTURE=8,'BBBBB',ACTION=RO
FD NAME=FIELD3,LENGTH=8,PICTURE=8,'A AA',ACTION=RO
FD NAME=FIELD4,LENGTH=8,PICTURE=8,'BB B',ACTION=RO
FD NAME=FIELD5,LENGTH=8,PICTURE=8,'AAA',ACTION=RO
FD NAME=FIELD6,LENGTH=8,PICTURE=8,'CCCCC',ACTION=WV
FD NAME=FIELD7,LENGTH=8,PICTURE=8,'DDDD',ACTION=WV
FD NAME=FIELD8,LENGTH=8,PICTURE=8,'C CC',ACTION=WV
FD NAME=FIELD9,LENGTH=8,PICTURE=8,'DD D',ACTION=WV
FD NAME=FIELD10,LENGTH=8,PICTURE=8,'CCC',ACTION=WV
CREATE QUANTITY=300,NAME=(FIELD1,FIELD2,FIELD3,
FIELD4,FIELD5,FIELD6,FIELD7,FIELD8,FIELD9,FIELD10)
END
/*
```

72

X

The control statements are as follows:

- OUTSET DD defines the output sequential data set on a disk volume. Ten tracks of primary space and 10 tracks of secondary space are allocated to the data set.
- SYSIN DD defines the control data set that follows in the input stream.

- DSD marks the beginning of a set of utility control statements and refers to the DD statement defining the output data set.
- FD defines a field to be used in the subsequent construction of output records. The direction and frequency of the initial roll or wave depend on the location of data in the field.
- CREATE constructs 300 records from the contents of the previously defined fields.
- END signals the end of a set of utility control statements.

Chapter 6. IEBEDIT (Edit Job Stream) Program

You can use IEBEDIT to create a data set containing a selection of jobs or job steps. These jobs or job steps can be entered into the job stream at a later time for processing.

You can edit and selectively copy an input job stream to an output data set by using IEBEDIT. The program can copy:

- An entire job or jobs, including JOB statements and any associated JOBLIB statements, and JES2 or JES3 control statements.
- Selected job steps, including the JOB statement, JES2 or JES3 control statements following the JOB statement, and any associated JOBLIB statements.

All selected JOB statements, JES2 or JES3 control statements, JOBLIB statements, jobs, or job steps are placed in the output data set in the same order in which they appear in the input data set. A JES2 or JES3 control statement or a JOBLIB statement will be copied only if it follows a selected JOB statement.

When IEBEDIT encounters a selected job step containing an input record having the characters ".*" (period, period, asterisk) in columns 1 through 3, the program automatically converts that record to a termination statement (/ * statement) and places it in the output data set.

A "/ *nonblank" indicates a JES2 or JES3 control statement.

Input and Output

IEBEDIT uses the following input:

- An input data set, which is a sequential data set consisting of a job stream. The input data set is used as source data in creating an output sequential data set.
- A control data set, which contains utility control statements that are used to specify the organization of jobs and job steps in the output data set.

IEBEDIT produces the following output:

- An output data set, which is a sequential data set consisting of a resultant job stream.
- A message data set, which is a sequential data set that contains applicable control statements, error messages, if applicable, and, optionally, the output data set.

Related reading: For information about the IEBEDIT return codes, see [Appendix A, “Invoking Utility Programs from an Application Program,”](#) on page 315.

Control

IEBEDIT is controlled by job control statements and utility control statements. The job control statements are required to run or load the program and to define the data sets used and produced by the program. The utility control statements are used to control the functions of the program.

Job Control Statements

Table 22 on page 119 shows the job control statements for IEBEDIT.

Table 22. Job control statements for IEBEDIT

Statement	Use
JOB	Starts the job.
EXEC	Specifies the program name (PGM=IEBEDIT) or, if the job control statements reside in a procedure library, the procedure name.

Table 22. Job control statements for IEBEDIT (continued)

Statement	Use
SYSPRINT DD	Defines a sequential data set for messages. The data set can be written to a system output device, a tape volume, or a direct access volume.
SYSUT1 DD	Defines a sequential input data set on a card reader, tape volume, or direct access device.
SYSUT2 DD	Defines a sequential output data set on a card punch, printer, tape volume, or direct access device.
SYSIN DD	Defines the control data set. The data set normally is included in the input stream; however, it can be defined as a member of a procedure library or as a sequential data set existing somewhere other than in the input stream.

SYSPRINT DD Statement

The block size for the SYSPRINT data set must be a multiple of 121. If not, the job step will be ended with a return code of 8.

SYSUT1, SYSUT2, and SYSIN DD Statements

The block size for the SYSIN, SYSUT1, and SYSUT2 data sets must be a multiple of 80. Any blocking factor can be specified for these record sizes.

Any JES2 or JES3 control statement or JOBLIB DD statement that follows a selected JOB statement in the SYSUT1 data set will be automatically copied to the output data set. JES2 or JES3 control statements preceding the JOB statement are assumed to belong to the previous job. JES2 or JES3 control statements preceding the first JOB statement are included only if you request a total copy.

However, if the SYSUT1 data set is included in the input stream (SYSUT1 DD DATA) JES2 or JES3 control statements are included only if a delimiter other than "/" is coded in the SYSUT1 DD DATA statement. If another delimiter is not coded, the first two characters of the JES2 or JES3 control statement will act as a delimiter and end the SYSUT1 data set.

Related reading: For more information about coding another delimiter, see [z/OS MVS JCL User's Guide](#).

Utility Control Statement

IEBEDIT uses only one utility control statement, EDIT. Continuation requirements for the statement are described in [“Continuing utility control statements”](#) on page 8.

You can use the EDIT statement to indicate which step or steps of a specified job in the input data set you want included in the output data set. Any number of EDIT statements can be included in an operation. Thus, you can create a data set with any number of job steps in one operation.

EDIT statements must be included in the same order as the input jobs that they represent. You can copy the entire input data set by omitting the EDIT statement.

The syntax of the EDIT statement is:

Label	Statement	Parameters
[label]	EDIT	[START=jobname] [, TYPE={POSITION INCLUDE EXCLUDE}] [, STEPNAME=(namelist)] [, NOPRINT]

where:

START=jobname

specifies the name of the input job to which the EDIT statement applies. (Each EDIT statement must apply to a separate job.) If START is specified without TYPE and STEPNAME, the JOB statement and all job steps for the specified job are included in the output.

Default: If START is omitted and only one EDIT statement is provided, the first job encountered in the input data set is processed. If START is omitted from an EDIT statement other than the first statement, processing continues with the next JOB statement found in the input data set.

TYPE={POSITION|INCLUDE|EXCLUDE}

specifies the contents of the output data set. These values can be coded:

POSITION

specifies that the output is to consist of a JOB statement, the job step specified in the STEPNAME parameter, and all steps that follow that job step. All job steps preceding the specified step are omitted from the operation. POSITION is the default.

INCLUDE

specifies that the output data set is to contain a JOB statement and all job steps specified in the STEPNAME parameter.

EXCLUDE

specifies that the output data set is to contain a JOB statement and all job steps belonging to the job except those steps specified in the STEPNAME parameter.

STEPNAME=(namelist)

specifies the names of the job steps that you want to process.

Namelist can be a single job step name, a list of step names separated by commas, or a sequential range of steps separated by a hyphen (for example, STEPA-STEPE). Any combination of these may be used in one *namelist*. If more than one step name is specified, the entire *namelist* must be enclosed in parentheses.

When coded with TYPE=POSITION, STEPNAME specifies the first job step to be placed in the output data set. Job steps preceding this step are not copied to the output data set.

When coded with TYPE=INCLUDE or TYPE=EXCLUDE, STEPNAME specifies the names of job steps that are to be included in or excluded from the operation. For example, STEPNAME=(STEPA,STEPF-STEPL,STEPZ) indicates that job steps STEPA, STEPF through STEPL, and STEPZ are to be included in or excluded from the operation.

If STEPNAME is omitted, the entire input job whose name is specified on the EDIT statement is copied. If no job name is specified, the first job encountered is processed.

NOPRINT

specifies that the message data set is not to include a listing of the output data set.

Default: The resultant output is listed in the message data set.

IEBEDIT Examples

These examples show some of the uses of IEBEDIT. You can use [Table 23 on page 121](#) as a quick-reference guide.

Table 23. IEBEDIT example directory

Operation	Devices	Comments	Example
EDIT	Disk	Copies JOB statement for JOBA, the job step STEPF, and all steps that follow.	“Example 4: Copy Statement for JOBA and JOB STEPF” on page 123

Table 23. IEBEDIT example directory (continued)

Operation	Devices	Comments	Example
EDIT	Disk and Tape	Includes a job step from one job and excludes a job step from another job.	“Example 3: Include Step from One Job, Exclude Step from Another” on page 123
EDIT	Tape	Copies one job into output data set.	“Example 1: Copy One Job” on page 122
EDIT	Tape	Selectively copies job steps from each of three jobs.	“Example 2: Copy Steps from Three Jobs” on page 122
EDIT	Tape	Copies entire input data set. The “.” record is converted to a “/” statement in the output data set.	“Example 5: Copy Entire Input Data Set” on page 124
EDIT	Tape	Copies entire input data set, including JES2 control statements.	“Example 6: Copy Entire Data Set to Include New Delimiter” on page 124

Examples that use **disk** or **tape** in place of actual device numbers must be changed before use. The actual device numbers depend on how your installation has defined the devices to your system.

Example 1: Copy One Job

In this example, one job (JOBA), including all of its job steps, is copied into the output data set. The input data set contains three jobs: JOBA, JOBB and JOBC.

```
//EDIT1    JOB    ...
//STEP1    EXEC   PGM=IEBEDIT
//SYSPRINT DD    SYSOUT=A
//SYSUT1    DD    DSN=INJOBS,UNIT=tape,
//           DISP=(OLD,KEEP),VOL=SER=001234
//SYSUT2    DD    DSN=OUTTAPE,UNIT=tape,DISP=(NEW,KEEP),
//           VOL=SER=001235,DCB=(RECFM=FB,LRECL=80,BLKSIZE=80),
//SYSIN     DD    *
//           EDIT   START=JOBA
/*
```

The control statements are as follows:

- SYSUT1 DD defines the input data set, INJOBS. The data set resides on a standard labeled tape volume (001234).
- SYSUT2 DD defines the output data set, called OUTTAPE. The data set is to reside as the first data set on a standard labeled tape volume (001235). The system will select an optimal block size.
- SYSIN DD defines the control data set, which follows in the input stream.
- EDIT indicates that JOBA is to be copied in its entirety.

Example 2: Copy Steps from Three Jobs

This example copies job steps from each of three jobs. The input data set contains three jobs: JOBA, which includes STEPA, STEPB, STEPC, and STEPD; JOBB, which includes STEPE, STEPF, and STEPG; and JOBC, which includes STEPH and STEPJ.

```
//EDIT2    JOB    ...
//STEP1    EXEC   PGM=IEBEDIT
//SYSPRINT DD    SYSOUT=A
//SYSUT1    DD    DSN=INJOBS,DISP=(OLD,KEEP),VOLUME=SER=001234,
//           UNIT=tape
//SYSUT2    DD    DSN=OUTSTRM,UNIT=tape,DISP=(NEW,KEEP),
//           DCB=(RECFM=F,LRECL=80,BLKSIZE=80),LABEL=(2,SL)
//SYSIN     DD    *
//           EDIT   START=JOBA,TYPE=INCLUDE,STEPNAME=(STEPD,STEPJ)
//           EDIT   START=JOBB,TYPE=INCLUDE,STEPNAME=STEPE
```

```

EDIT    START=JOB,TYPE=INCLUDE,STEPNAME=STEPJ
/*

```

The control statements are as follows:

- SYSUT1 DD defines the input data set, INJOBS. The data set resides on a standard labeled tape volume (001234).
- SYSUT2 DD defines the output data set, OUTSTRM. The data set is to reside as the second data set on a standard labeled tape volume (001235). The short block size is very inefficient.
- SYSIN DD defines the control data set, which follows in the input stream.
- The EDIT statements copy the JOB statements and job steps described as follows:
 1. The JOB statement and steps STEPC and STEPD for JOBA.
 2. The JOB statement and STEPE for JOBB.
 3. The JOB statement and STEPJ for JOBC.

Example 3: Include Step from One Job, Exclude Step from Another

This example includes a job step from one job and excludes a job step from another job. The input data set contains three jobs: JOBA, which includes STEPA, STEPB, STEPC, and STEPD; JOBB, which includes STEPE, STEPF, and STEPG; and JOBC, which includes STEPH and STEPJ.

```

//EDIT3    JOB    ...
//STEP1    EXEC   PGM=IEBEDIT
//SYSPRINT DD    SYSOUT=A
//SYSUT1   DD     DSN=INSET,UNIT=disk,DISP=(OLD,KEEP),
//          VOLUME=SER=111111
//SYSUT2   DD     DSN=OUTTAPE,UNIT=tape,LABEL=(,NL),
//          DCB=(DEN=2,RECFM=FB,LRECL=80,BLKSIZE=8160),
//          DISP=(,KEEP)
//SYSIN    DD     *
            EDIT   START=JOB,TYPE=INCLUDE,STEPNAME=(STEPF-STEPG)
            EDIT   START=JOB,TYPE=EXCLUDE,STEPNAME=STEPJ
/*

```

The control statements are as follows:

- SYSUT1 DD defines the input data set, INSET. The data set resides on a disk volume (111111).
- SYSUT2 DD defines the output data set, OUTTAPE. The data set is to reside as the first or only data set on an unlabeled (800 bits per inch) tape volume.
- SYSIN DD defines the control data set, which follows in the input stream.
- The EDIT statements copy JOB statements and job steps as follows:
 1. The JOB statement and steps STEPF and STEPG for JOBB.
 2. The JOB statement and STEPH, excluding STEPJ, for JOBC.

Example 4: Copy Statement for JOBA and JOB STEPF

This example copies the JOB statement for JOBA, the job step STEPF, and all the steps that follow it. The input data set contains one job (JOBA), which includes STEPA through STEPL. Job steps STEPA through STEPE are not included in the output data set.

```

//EDIT4    JOB    ...
//STEP1    EXEC   PGM=IEBEDIT
//SYSPRINT DD    SYSOUT=A
//SYSUT1   DD     DSN=INSTREAM,UNIT=disk,
//          DISP=(OLD,KEEP),VOLUME=SER=111111
//SYSUT2   DD     DSN=OUTSTREM,UNIT=disk,
//          DISP=(,KEEP),VOLUME=SER=222222,SPACE=(TRK,2)
//          DCB=(RECFM=FB,LRECL=80,BLKSIZE=24000),
//SYSIN    DD     *
            EDIT   START=JOBA,TYPE=POSITION,STEPNAME=STEPF
/*

```

The control statements are as follows:

- SYSUT1 DD defines the input data set, INSTREAM. The data set resides on a disk volume (111111).
- SYSUT2 DD defines the output data set, OUTSTREAM. The data set is to reside on a disk volume (222222). Two tracks are allocated for the output data set.
- SYSIN DD defines the control data set, which follows in the input stream.
- EDIT copies the JOB statement for JOBA and job steps STEPF through STEPL.

Example 5: Copy Entire Input Data Set

This example copies the entire input data set. The record containing the characters "."* in columns 1 through 3 is converted to a "/"* statement in the output data set.

```
//EDIT5      JOB    ...
//STEP1      EXEC   PGM=IEBEDIT
//SYSPRINT   DD     SYSOUT=A
//SYSUT2     DD     DSN=OUTTAPE,UNIT=tape,
//            VOLUME=SER=001234,DISP=(NEW,KEEP),
//            DCB=(RECFM=F,LRECL=80,BLKSIZE=80)
//SYSIN      DD     DUMMY
//SYSUT1     DD     DATA
//BLDGDGIX   JOB    ...
//            EXEC   PGM=IEHPROGM
//SYSPRINT   DD     SYSOUT=A
//DD1        DD     UNIT=disk,VOLUME=SER=111111,DISP=OLD
//SYSIN      DD     *
//            BLDG    SCRATCH VTOC,VOL=3390=111111
//            ..*
//            /*
```

The control statements are as follows:

- SYSUT2 DD defines the output data set, OUTTAPE. The data set will be the first data set on a tape volume (001234).
- SYSIN DD defines a dummy control data set.
- SYSUT1 DD defines the input data set, which follows in the input stream. The job is stopped when the statement "/"* is encountered. (SYSUT1 therefore includes the DELETE JOB statement, EXEC statement, SYSPRINT, DD1, and SYSIN DD statements.)

Example 6: Copy Entire Data Set to Include New Delimiter

This example copies the entire input data set, including the JES2 control statement, because a new delimiter (JP) has been coded. Otherwise, the "/"* in the JES2 control statement would have stopped the input.

```
//EDIT6      JOB    ...
//STEP1      EXEC   PGM=IEBEDIT
//SYSPRINT   DD     SYSOUT=A
//SYSUT2     DD     DSN=TAPEOUT,UNIT=tape,
//            VOL=SER=001234,LABEL=(,SL),DISP=(NEW,KEEP)
//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=800),
//SYSIN      DD     DUMMY
//SYSUT1     DD     DATA,DLM=JP
//LISTVTOC   JOB    ...
//*MESSAGE    JOB    NEEDS VOLUME 338000
//FSTEP      EXEC   PGM=IEHLIST
//SYSPRINT   DD     SYSOUT=A
//DD2        DD     UNIT=disk,VOL=SER=338000,DISP=OLD
//SYSIN      DD     *
//            LISTVTOC FORMAT,VOL=disk=338000
//            /*
```

The control statements are as follows:

- SYSUT2 DD defines the output data set, TAPEOUT. The data set will be the first data set on a standard label tape volume (001234).
- SYSIN DD defines a dummy control data set.

- SYSUT1 DD defines the input data set, which follows in the input stream. The DLM parameter defines characters JP to act as a delimiter for the input data.
- IEBEDIT copies the JOB statement through the "/*" statement (including the LISTVTOC and MESSAGE job statements, FSTEP EXEC statement, and SYSPRINT, DD2 and SYSIN DD statements).

Chapter 7. IEBGENER (Sequential Copy/Generate Data Set) Program

You can use IEBGENER to perform these tasks:

- Create a backup copy of a sequential data set, a member of a partitioned data set or PDSE or a z/OS UNIX System Services (z/OS UNIX) file such as a HFS file.
- Produce a partitioned data set or PDSE, or a member of a partitioned data set or PDSE, from a sequential data set or a z/OS UNIX file.
- Expand an existing partitioned data set or PDSE by creating partitioned members and merging them into the existing data set.
- Produce an edited sequential or partitioned data set or PDSE.
- Manipulate data sets containing double-byte character set data.
- Print sequential data sets, members of partitioned data sets or PDSEs or z/OS UNIX files.
- Reblock or change the logical record length of a data set.
- Copy user labels on sequential output data sets.
- Supply editing facilities and exits for your routines that process labels, manipulate input data, create keys, and handle permanent input/output errors.

Recommendation: If you have the DFSORT product installed, you should be using ICEGENER as an alternative to IEBGENER when making an unedited copy of a data set or member. It may already be installed in your system under the name IEBGENER. It generally gives better performance.

Related reading :

- For information about the linkage conventions for user exit routines, see [Appendix C, “Specifying User Exits with Utility Programs,”](#) on page 343.
- For information about user labels on sequential output data sets, see [“Processing User Labels”](#) on page 347.

Creating a Backup Copy

You can produce a backup copy of a sequential data set or member of a partitioned data set or PDSE by copying the data set or member to any IBM-supported output device. For example, a copy can be made from tape to tape, from DASD to tape, and so forth.

A data set that resides on a direct access volume can be copied to its own volume, provided that you change the name of the data set.

Restriction:

When you use IEBGENER to process partitioned data sets as sequential data sets, IEBGENER will not perform any directory entry processing (for instance, copying attributes of members of load module libraries). IEBCOPY does perform directory entry processing. IEBGENER does not support copying data sets that have been PACKED using the ISPF (PACK ON) command.

Producing a Partitioned Data Set or PDSE from Sequential Input

Through the use of utility control statements, you can logically divide a sequential data set into *record groups* and assign member names to the record groups. IEBGENER places the newly created members in an output partitioned data set or PDSE.

You cannot produce a partitioned data set or PDSE if an input or output data set contains spanned records.

Figure 22 on page 128 shows how a partitioned data set or PDSE is produced from a sequential data set used as input. One side of the figure shows the sequential data set. Utility control statements are used to divide the sequential data set into record groups and to provide a member name for each record group. The other side of the figure shows the partitioned data set or PDSE produced from the sequential input.

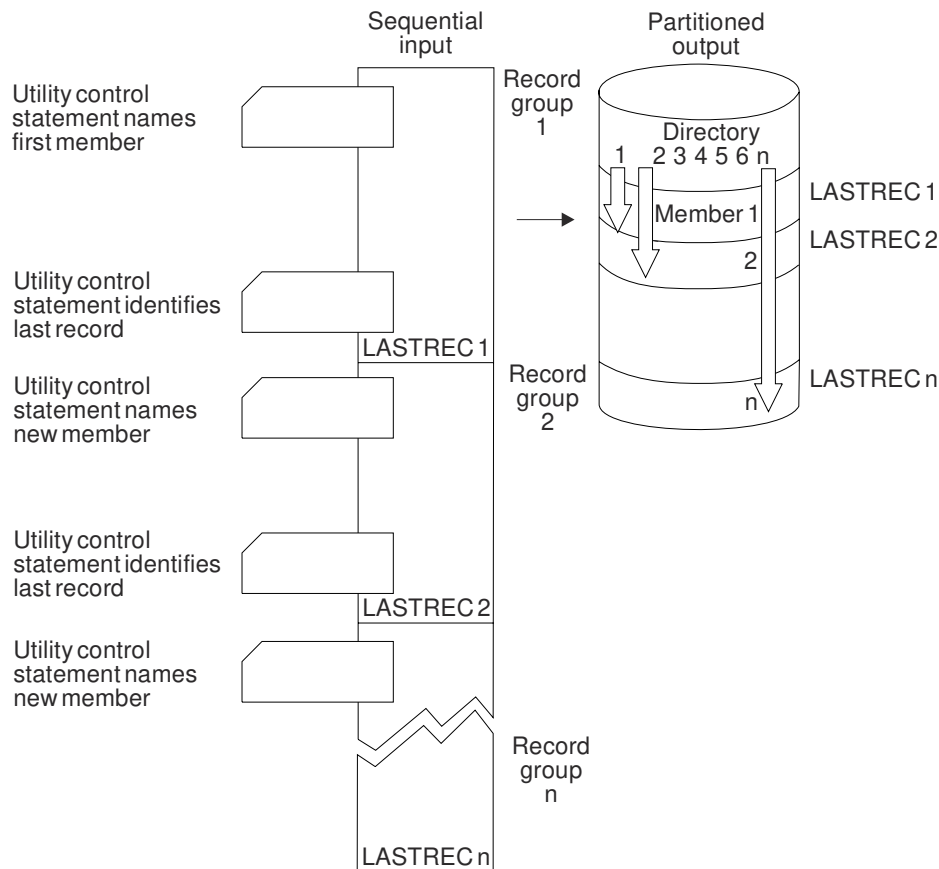


Figure 22. Using IEBGENER to create a partitioned data set or PDSE from sequential input

Adding Members to a Partitioned Data Set or PDSE

You can use IEBGENER to add members to a partitioned data set or PDSE. IEBGENER creates the members from sequential input and adds them to the data set. The merge operation—the ordering of the partitioned directory—is automatically performed by the program.

Figure 23 on page 129 shows how sequential input is converted into members that are merged into an existing partitioned data set or PDSE. The figure shows the sequential input that is to be merged with the existing partitioned data set or PDSE. Utility control statements are used to divide the sequential data set into record groups and to provide a member name for each record group. The figure also shows the expanded partitioned data set or PDSE. Members B, D, and F from the sequential data set were placed in *available space* and are sequentially ordered in the partitioned directory.

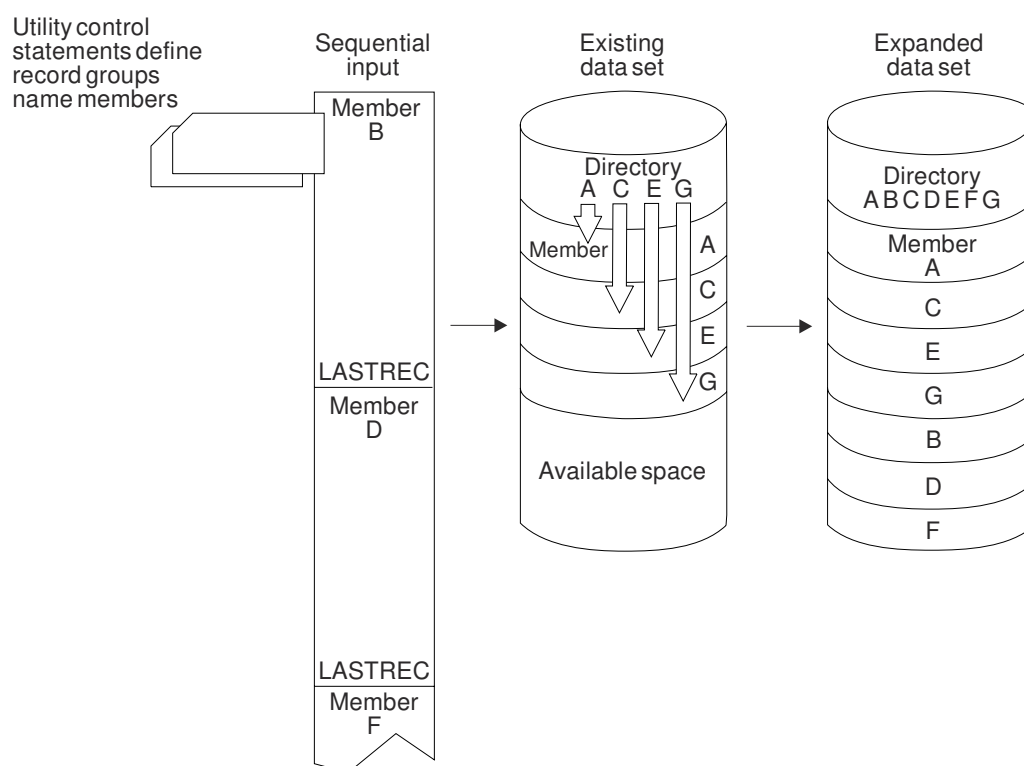


Figure 23. Using IEBGENER to add members to a partitioned data set or PDSE

Producing an Edited Data Set

You can use IEBGENER to produce an edited sequential or partitioned data set, or PDSE. Through the use of utility control statements, you can specify editing information that applies to a record, a group of records, selected groups of records, or an entire data set.

An edited data set can be produced by:

- Rearranging or omitting defined data fields within a record.
- Supplying literal information as replacement data.
- Converting data from packed decimal to unpacked decimal mode, unpacked decimal to packed decimal mode, or BCDIC (used here to mean the standard H character set of binary coded decimal interchange code) to EBCDIC mode.
- Adding or deleting shift-out/shift-in characters X'0E' and X'0F' when double-byte character set data is contained in the data set.

Figure 24 on page 130 shows part of an edited sequential data set. The figure first shows the data set before editing is performed. Utility control statements are used to identify the record groups to be edited and to supply editing information. In this figure, literal replacement information is supplied for information within a defined field. (Data is rearranged, omitted, or converted in the same manner.) The BBBB field in each record in the record group is to be replaced by CCCC. The figure then shows the data set after editing.

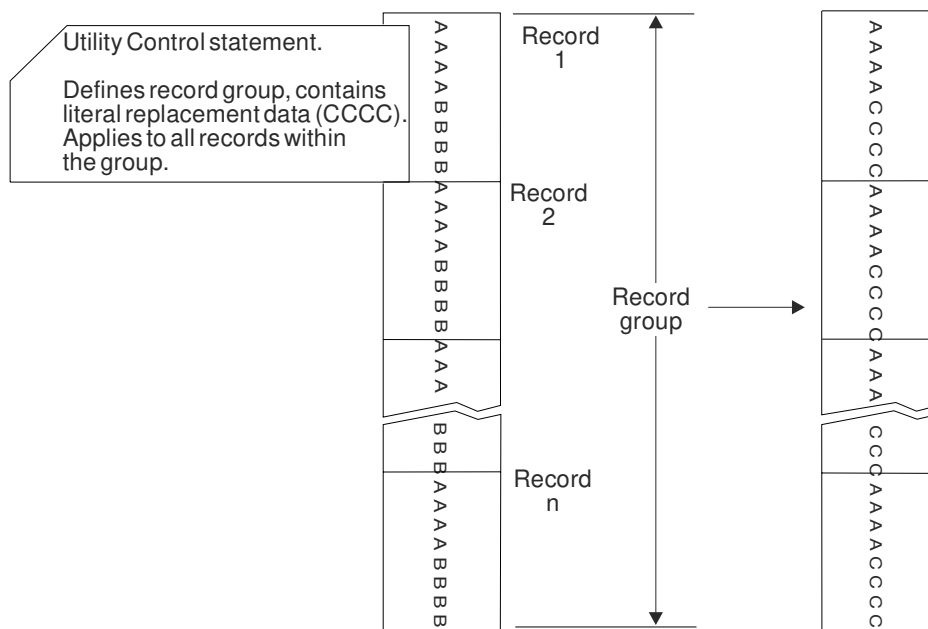


Figure 24. Editing a sequential data set using IEBGENER

IEBGNER cannot be used to edit a data set if the input and output data sets consist of variable spanned (VS) or variable blocked spanned (VBS) records and have equal block sizes and logical record lengths. In these cases, any utility control statements that specify editing are ignored. That is, for each physical record read from the input data set, the utility writes an unedited physical record on the output data set.

Related reading: For more information on converting from BCDIC to EBCDIC, see [z/OS DFSMS Using Data Sets](#).

Changing Logical Record Length

You can use IEBGENER to produce a reblocked output data set containing either fixed-length or variable-length records having a logical record length that differs from the input logical record length.

When you create a data set with a logical record length that differs from that of the input data set, you must specify where each byte in the output records is to come (either from literals or from input records). Any unspecified fields will contain unpredictable data. You cannot alter the logical record length if both the input and output data sets have variable or variable blocked record formats and the output logical record length is smaller than the input logical record length. In this case, IEBGENER fails with return code 12 and issues message IEB311I.

Using IEBGENER with Double-Byte Character Set Data

You can use IEBGENER to copy, edit, reblock, or print data sets that contain double-byte character set (DBCS) data. You can also convert sequential data sets containing DBCS data to partitioned data sets. Double-byte character sets are used to represent languages too complex to be represented with a single-byte character set. Japanese, for example, requires a double-byte character set.

DBCS data is typically enclosed in shift-out/shift-in X'0E' and X'0F' single-byte characters to indicate that the data must be handled as pairs of bytes and not single bytes. *Shift-out* indicates that the data is now shifting out of a single-byte character set string and *into* a double-byte character set string, and *shift-in* indicates that a double-byte character set string is shifting into a single-byte character set string. You can add or delete the shift-out/shift-in characters using IEBGENER. If you add them to a data set, however, you must account for the additional bytes they will require when you determine the logical record length of the output data set. You can also validate DBCS data using IEBGENER. DBCS data is not considered "valid" unless each byte of every character has a value between X'41' and X'FE', inclusive, or unless

the DBCS character is a DBCS space (X'4040'). This checking will ensure that each DBCS character is printable.

In order to print a data set containing DBCS data, the DBCS strings must be enclosed in shift-out/shift-in characters.

Input and Output

IEBGENER uses the following input:

- An input data set, which contains the data that is to be copied, edited, converted into a partitioned data set or PDSE, or converted into members to be merged into an existing data set. The input is either a sequential data set or a member of a partitioned data set or PDSE.
- A control data set, which contains utility control statements. The control data set is required if editing is to be performed or if the output data set is to be a partitioned data set or PDSE.
- The PARM parameter on the JCL EXEC statement or the dynamic invocation equivalent.

IEBGENER produces the following output:

- An output data set, which can be either sequential, including large format sequential, or partitioned. The output data set can be either a new data set (created during the current job step) or an existing data set. It can be a sequential data set or a PDS or PDSE that will be expanded. If a partitioned data set or PDSE is created, it is a new member with a new directory entry. None of the information is copied from the previous directory entry. The same sequential data set cannot be used as both the input and output data set. The output sequential data set cannot be any of the input data sets in a sequential concatenation.
- A message data set, which contains informational messages (for example, the contents of utility control statements) and any error messages.

Message IEC507D will be issued twice when adding data or members to an existing data set which has an unexpired expiration date. This occurs because the input and output data sets are opened twice.

If IEBGENER is invoked from an application program, you can dynamically allocate the data sets by issuing SVC 99 before calling IEBGENER.

Related reading :

- For information about the messages issued by the IEBGENER utility, see [z/OS MVS System Messages, Vol 7 \(IEB-IEE\)](#).
- For information about IEBGENER return codes, see [Appendix A, “Invoking Utility Programs from an Application Program,”](#) on page 315.
- For information about the JCL EXEC statement, see [“EXEC Statement”](#) on page 132.

Control

You control IEBGENER with job and utility control statements. The job control statements run IEBGENER and define the data sets that are used and produced by the program. The utility control statements control the functions of IEBGENER.

Job Control Statements

Table 24 on page 131 shows the job control statements for IEBGENER.

Table 24. Job control statements for IEBGENER

Statement	Use
JOB	Starts the job.
EXEC	Specifies the program name (PGM=IEBGENER) or, if the job control statements reside in a procedure library, the procedure name.

Table 24. Job control statements for IEBGENER (continued)

Statement	Use
SYSPRINT DD	Defines a sequential data set for messages. The data set can be written to a system output device, a tape volume, or a DASD volume.
SYSUT1 DD	Defines the input data set. It can define a sequential data set, a member of a partitioned data set or PDSE or a z/OS UNIX file. A sequential data set can be basic format, large format, extended format, compressed format, spooled input, tape, card reader, TSO terminal or dummy.
SYSUT2 DD	Defines the output data set. It can define a sequential data set, a member of a partitioned data set or PDSE, a partitioned data set or PDSE or a z/OS UNIX file. A sequential data set can be basic format, large format, extended format, compressed format, spooled output, tape, card punch, printer, TSO terminal or dummy.
SYSIN DD	Defines the control data set, or specifies DUMMY when the output is sequential and no editing is specified. The control data set normally resides in the input stream; however, it can be defined as a member in a partitioned data set or PDSE.

EXEC Statement

The EXEC statement is required for each use of IEBGENER.

You can code a value like: PARM='SDB=xxxxx' on the EXEC statement. This value is effective only if the output block size is not supplied by any other source. The system calculates an optimal value through a function called "system-determined block size". This PARM value controls whether the block size can exceed 32,760 bytes. SDB is the only valid parameter for the EXEC statement. You can code one of these values for xxxxx.

INPUT or YES

If the input block size is greater than 32,760 bytes, the default output block size greater than 32,760 bytes is permitted. Otherwise, it is not permitted.

SMALL

The default output block size is 32,760 bytes or less.

LARGE

The default output block size can be greater than 32,760 bytes. Older programs cannot read data sets that have such large blocks.

NO

The default output block size is 32,760 bytes or less, but the block size might be less efficient than if you coded SMALL.

Your system programmer sets the default value for SDB by setting the COPYSDB in the DEVSUPxx member in SYS1.PARMLIB. The IBM-supplied default for COPYSDB is INPUT. Prior to OS/390 Version 2 Release 10, IEBGENER ignored a PARM value and operated as if SDB=NO had been coded. Currently, only magnetic tape devices and dummy data sets allow a block size that exceeds 32,760 bytes. Do not depend on that restriction. Future levels of operating systems might allow larger blocks in more kinds of data sets.

The following conditions can cause your output data set to have a block size that exceeds 32,760 bytes:

- You coded the BLKSIZE keyword on the DD statement and the device will support the value.
- You coded DISP=MOD on the SYSUT2 DD statement if the output data set exists and has a large BLKSIZE.
- The output device type supports a large block size, but you did not code one, and SDB=LARGE is active.
- The output device type supports a large block size, but you did not code one. The input data set has a large block size and SDB=INPUT is active.

Note: In the last two cases, the system limits the block size to a block size limit. IEBGENER will never calculate a block size value that is too large for the device. With fixed-blocked records, the block size limit does not have to be a multiple of LRECL. The block size limit is the first one of these that is available:

- BLKSZLIM keyword on the output DD statement.
- Block size limit in the data class, even if the data set is not SMS-managed. You can code the DATACLAS keyword on the DD statement, or the system's ACS routines can assign a data class.
- TAPEBLKSZLM keyword in the DEVSUPxx member of SYS1.PARMLIB. The system programmer sets this as a system default.
- 32,760 bytes.

Using multiple buffers for IEBGENER increases the amount of virtual storage that is needed to run the program. You may need to change, or add, the REGION parameter for the additional storage to avoid 80A abends.

The default for the number of buffers is five. You can override this by specifying BUFNO or NCP on the SYSUT1 or SYSUT2 DD statement.

Before you run IEBGENER, you may need to calculate the region size (in virtual storage) that is needed to run the program. Specify this value in the REGION parameter.

You can calculate the region size by using the formula, as follows:

```
region size = 50K + (2 + SYSUT1 BUFNO)*(SYSUT1 BLKSIZE) +
              (2 + SYSUT2 BUFNO)*(SYSUT2 BLKSIZE)
```

If you do not use BUFNO in your JCL, use the default value of 5.

Related reading: For information on how to calculate the region size, see [“Example 5: Produce Blocked Copy on Tape from Unblocked Disk File”](#) on page 144.

SYSPRINT DD Statement

The SYSPRINT DD statement is required for each use of IEBGENER. The block size for the SYSPRINT data set must be a multiple of 121. Any blocking factor can be specified for this record size, but the maximum value for the block size is 32670 bytes.

SYSUT1 DD Statement

The input data set for IEBGENER, as specified in SYSUT1, can contain fixed, variable, undefined, or variable spanned records. You cannot use concatenated data sets with *unlike* attributes except for block size or device type as input to IEBGENER. The rules for concatenating data sets on SYSUT1 are the same as those for BSAM unless you are concatenating data sets on DASD and tapes. Then the concatenating rules are the same as for QSAM.

If the SYSUT1 data set is in the system input stream and contains JCL statements, code SYSUT1 DD DATA and not SYSUT1 DD *.

Block size for the input data set must be available in the data set label (DSCB), or tape label, or the DD statement. For a tape data set the block size can exceed 32,760 bytes.

The default record format is undefined (U) for the input data set. Record format must be specified if the data set is new, undefined, or a dummy data set.

The input logical record length must be specified when the record format is fixed blocked, variable spanned, or variable blocked spanned, or when the data set is new, or a dummy data set. If the input and output logical record length are not available and the record format is V or VB, IEBGENER sets the maximum record length to the block size -4, with a maximum of 32,760. In all other cases, a default logical record length of 80 is used.

A partitioned data set or PDSE cannot be produced if an input data set contains spanned records.

If both the SYSUT1 and the SYSUT2 DD statements specify standard user labels (SUL), IEBGENER copies user labels from SYSUT1 to SYSUT2.

Related reading:

- For information on concatenated data sets for BSAM and QSAM, see [z/OS DFSMS Using Data Sets](#).
- For information about available options for user label processing, see [“Processing User Labels” on page 347](#).

SYSUT2 DD Statement

The output data set for IEBGENER, as specified in SYSUT2, can contain fixed, variable, undefined, or variable spanned records (except partitioned data sets or PDSEs, which cannot contain variable spanned records). IEBGENER can reblock records if you specify a new maximum block length on the SYSUT2 DD statement. If you are reblocking fixed-length or variable-length records, keys can be retained only if you supply an exit routine to retain them. You cannot retain keys when you reblock variable spanned records.

If the output data set is on a card punch or a printer (not a spooled data set), you must specify DCB information on the SYSUT2 DD statement.

When the data set is on DASD or tape and record format and logical record length are not specified in the JCL for the output data set, and the data class does not supply them, values for each are copied from the input data set. Logical record length is not copied if the RECFM is undefined.

Note: If you are using IEBGENER with an SMF dump data set, IEBGENER can truncate your data. The SMF dump program, IFASMFDP, creates a data set with a logical record length larger than IEBGENER allows. Thus, IEBGENER will truncate the data set to a logical record length of 32⁺760, resulting in a loss of 7 bytes. If you do not have actual data in those 7 bytes, you will lose nothing in the truncation. However, care should be taken in using IEBGENER with the SMF dump program.

The output block size need not be specified, if logical record length and record format are specified or available for the input data set. The EXEC statement input data set and output device type affect the output block size (when not specified). If the output device type is tape or dummy, the block size can exceed 32⁺760 bytes. If you specify BLKSIZE, then SDB in the PARM value on the EXEC statement will have no effect.

The output logical record length must be specified when editing is to be performed and the record format is fixed blocked, variable spanned or variable blocked spanned.

A partitioned data set or PDSE cannot be produced if an input or output data set contains spanned records.

Table 25 on page 134 shows the effect of the availability of RECFM, LRECL, and BLKSIZE in the SYSUT2 DD statement.

<i>Table 25. Effect of output DD statements. "Available" means that you coded it or that the value is in the data set label or data class.</i>			
RECFM	LRECL	BLKSIZE	Result
Available	Available	Available	All are used.
Available	Available	Omitted	Block size is determined by the system.
Available	Omitted	Available	LRECL is copied from the input unless RECFM is undefined. The specified RECFM and BLKSIZE are used.
Available	Omitted	Omitted	LRECL is copied from the input unless RECFM is undefined. The BLKSIZE is copied from the input unless it is too large for the device; in that case, the system determines an optimal BLKSIZE.
Omitted	Available	Available	RECFM is copied from the input. The specified LRECL and BLKSIZE are used.

Table 25. Effect of output DD statements. "Available" means that you coded it or that the value is in the data set label or data class. (continued)

RECFM	LRECL	BLKSIZE	Result
Omitted	Available	Omitted	RECFM is copied from the input. The specified LRECL is used. BLKSIZE is determined by system.
Omitted	Omitted	Available	RECFM and LRECL are copied from the input, but LRECL is not copied if RECFM is undefined.
Omitted	Omitted	Omitted	All three are copied unless BLKSIZE is too large for the device. In that case, the system determines an optimal BLKSIZE.

Related reading: For a description of SDB, see [“EXEC Statement” on page 132](#).

When using IEBGENER, it is possible to override the existing RECFM, LRECL, and BLKSIZE attributes. This can occur when you are adding data to a sequential data set, or when adding or replacing a member to a PDS or PDSE. If the data set is a PDS or PDSE, any changes to these fields will affect all members in the data set. The new values will permanently override the existing values that stored in the DSCB until another override occurs.

If you override the attributes when adding to a non-empty data set, you probably have damaged the data set. Here are examples where you have not damaged the data set:

- The record format is variable and you set a larger record length that is not too large for the block size.
- The record format is undefined and you set a larger block size.
- The record format is fixed unblocked and you changed it to fixed blocked (FB) with a BLKSIZE value that is a multiple of LRECL.

In all of these cases, you have set attributes that are consistent with the previous records.

SYSIN DD Statement

The SYSIN DD statement is required for each use of IEBGENER. The block size for the SYSIN data set must be a multiple of 80. Any blocking factor can be specified for this block size. If SYSIN is dummy, IEBGENER copies the input data set sequentially.

Utility Control Statements

IEBGENER is controlled by utility control statements. The statements and the order in which they must appear are listed in [Table 26 on page 136](#).

The control statements are included in the control data set as required. If no utility control statements are included in the control data set, the entire input data set is copied sequentially.

When the output is to be sequential and editing is to be performed, one GENERATE statement and as many RECORD statements as required are used. If you are providing exit routines, an EXITS statement is required.

When the output is to be partitioned, one GENERATE statement, one MEMBER statement per output member, and RECORD statements, as required, are used. If you are providing exit routines, an EXITS statement is required.

A continuation line must start in columns 4 to 16, and a nonblank continuation line in column 72 is optional.

Table 26. IEBGENER utility control statements

Statement	Use
GENERATE	Indicates the number of member names and alias names, record identifiers, literals, and editing information contained in the control data set.
EXITS	Indicates that user routines are provided.
LABELS	Specifies user-label processing.
MEMBER	Specifies the member name and alias of a member of a partitioned data set or PDSE to be created.
RECORD	Defines a record group to be processed and supplies editing information.

GENERATE Statement

The GENERATE statement is required when: output is to be partitioned; editing is to be performed; or user routines are provided or label processing is specified. The GENERATE statement must appear before any other IEBGENER utility statements. If it contains errors or is inconsistent with other statements, IEBGENER is ended.

The syntax of the GENERATE statement is:

Label	Statement	Parameters
[<i>label</i>]	GENERATE	[, MAXNAME= <i>n</i>] [, MAXFLDS= <i>n</i>] [, MAXGPS= <i>n</i>] [, MAXLITS= <i>n</i>] [, DBCS={YES NO}]

where:

MAXNAME=*n*

specifies a number, from 1 to 3276, that is greater than or equal to the total number of member names and aliases appearing in subsequent MEMBER statements. MAXNAME is required if there are one or more MEMBER statements.

MAXFLDS=*n*

specifies a number, from 1 to 4095, that is greater than or equal to the total number of FIELD parameters appearing in subsequent RECORD statements. MAXFLDS is required if there are any FIELD parameters in subsequent RECORD statements.

MAXGPS=*n*

specifies a number, from 1 to 2520, that is greater than or equal to the total number of IDENT or IDENTG parameters appearing in subsequent RECORD statements. MAXGPS is required if there are any IDENT or IDENTG parameters in subsequent RECORD statements.

MAXLITS=*n*

specifies a number, from 1 to 2730, that is greater than or equal to the total number of characters contained in the FIELD literals of subsequent RECORD statements. Any DBCS characters used as literals on FIELD parameters count as two characters each.

MAXLITS is required if the FIELD parameters of subsequent RECORD statements contain literals. MAXLITS does not apply to literals used in IDENT or IDENTG parameters.

DBCS={YES|NO}

specifies if the input data set contains double-byte character set data.

EXITS Statement

The EXITS statement is used to identify exit routines you want IEBGENER to use. The syntax of the EXITS statement is:

Label	Statement	Parameters
[<i>label</i>]	EXITS	[INHDR= <i>routinename</i>] [, OUTHDR= <i>routinename</i>] [, INTLR= <i>routinename</i>] [, OUTTLR= <i>routinename</i>] [, KEY= <i>routinename</i>] [, DATA= <i>routinename</i>] [, IOERROR= <i>routinename</i>] [, TOTAL=(<i>routinename,size</i>)]

where:

INHDR=*routinename*

specifies the name of the routine that processes user input header labels.

OUTHDR=*routinename*

specifies the name of the routine that creates user output header labels. OUTHDR is ignored if the output data set is partitioned.

INTLR=*routinename*

specifies the name of the routine that processes user input trailer labels.

OUTTLR=*routinename*

specifies the name of the routine that processes user output trailer labels. OUTTLR is ignored if the output data set is partitioned.

KEY=*routinename*

specifies the name of the routine that creates the output record key. This routine does not receive control when a data set consisting of variable spanned (VS) or variable blocked spanned (VBS) type records is processed because no processing of keys is permitted for this type of data.

DATA=*routinename*

specifies the name of the routine that modifies the physical record (logical record for variable blocked type records) before it is processed by IEBGENER.

IOERROR=*routinename*

specifies the name of the routine that handles permanent input/output error conditions.

TOTAL=(*routinename,size*)

specifies that a user exit routine is to be provided before writing each record. The keyword OPTCD=T must be specified for the SYSUT2 DD statement. TOTAL is valid only when IEBGENER is used to process sequential data sets. These values must be coded:

routinename

specifies the name of your totaling routine.

size

specifies the number of bytes needed to contain totals, counters, pointers, and so forth. *Size* should be coded as a whole decimal number.

Related reading :

- For information about exit routines, see [Appendix C, “Specifying User Exits with Utility Programs,”](#) on page 343.
- For information about the processing of user labels as data set descriptors and user label totaling, see [“Processing User Labels”](#) on page 347.

LABELS Statement

The LABELS statement specifies if user labels are to be treated as data by IEBGENER. For a detailed discussion of this option, refer to [“Processing User Labels”](#) on page 347.

The LABELS statement is used when you want to specify that: no user labels are to be copied to the output data set; user labels are to be copied to the output data set from records in the data portion of the SYSIN data set; or user labels are to be copied to the output data set after they are modified by the user's label processing routines. If more than one valid LABELS statement is included, all but the last LABELS statement are ignored.

The syntax of the LABELS statement is:

Label	Statement	Parameters
[<i>label</i>]	LABELS	[DATA={ <u>YES</u> NO ALL ONLY INPUT}]

where:

DATA={YES|NO|ALL|ONLY|INPUT}

specifies if user labels are to be treated as data by IEBGENER. These values can be coded:

YES

specifies that any user labels that are not rejected by a label processing routine you have specified on the EXITS statement are to be treated as data. Processing of labels as data ends in compliance with standard return codes. YES is the default.

NO

specifies that user labels are not to be treated as data. In order to make standard user label (SUL) exits inactive, NO must be specified when processing input/output data sets with nonstandard labels (NSL).

ALL

specifies that all user labels in the group currently being processed are to be treated as data. A return code of 16 causes IEBGENER to complete processing the remainder of the group of user labels and to stop the job step.

ONLY

specifies that only user header labels are to be treated as data. User header labels are processed as data regardless of any return code. The job ends upon return from the OPEN routine.

INPUT

specifies that user labels for the output data set are supplied as 80-byte input records in the data portion of SYSIN. The number of input records that should be treated as user labels must be identified by a RECORD statement.

LABELS DATA=NO must be specified to make standard user labels (SUL) exits inactive when input/output data sets with nonstandard labels (NSL) are to be processed.

MEMBER Statement

The MEMBER statement is used when the output data set is to be partitioned. One MEMBER statement must be included for each member to be created by IEBGENER. The MEMBER statement provides the name and alias names of a new member.

All RECORD statements following a MEMBER statement refer to the one named in that MEMBER statement. If no MEMBER statements are included, the output data set is organized sequentially.

The syntax of the MEMBER statement is:

Label	Statement	Parameters
[<i>label</i>]	MEMBER	NAME=(<i>name</i> [, <i>alias 1</i>] [, <i>alias 2</i>] [, ...])

where:

NAME=(name[,alias][,...])

specifies a member name followed by a list of its aliases. If no aliases are specified in the statement, the member name need not be enclosed in parentheses.

RECORD Statement

The RECORD statement is used to define a record group and to supply editing information. A record group consists of records that are to be processed identically.

The RECORD statement is used when: the output is to be partitioned; editing is to be performed; or user labels for the output data set are to be created from records in the data portion of the SYSIN data set. The RECORD statement defines a record group by identifying the last record of the group with a literal name.

If no RECORD statement is used, the entire input data set or member is processed without editing. More than one RECORD statement might appear in the control statement stream for IEBGENER.

Within a RECORD statement, one IDENT or IDENTG parameter can be used to define the record group; one or more FIELD parameters can be used to supply the editing information applicable to the record group; and one LABELS parameter can be used to indicate that this statement is followed immediately by output label records.

If both output header labels and output trailer labels are to be contained in the SYSIN data set, you must include one RECORD statement (including the LABELS parameter), indicating the number of input records to be treated as user header labels and another RECORD statement (also including the LABELS parameter) for user trailer labels. The first such RECORD statement indicates the number of user header labels; the second indicates the number of user trailer labels. If only output trailer labels are included in the SYSIN data set, a RECORD statement must be included to indicate that there are no output header labels in the SYSIN data set (LABELS=0). This statement must precede the RECORD LABELS=n statement which signals the start of trailer label input records.

For a further discussion of the LABELS option, refer to [“Processing User Labels” on page 347](#).

The syntax of the RECORD statement is:

Label	Statement	Parameters
[label]	RECORD	[{ IDENT IDENTG } = (length , ' name ' , input-location)] [, FIELD = ([length] , [{ input-location ' literal ' }] , [conversion] , [output-location])] [, FIELD = . . .] [, LABELS = n]

where:

{IDENT|IDENTG}= (length,'name',input-location)

identifies the last record of a collection of records in the input data set. You use this parameter to identify the last record to be edited according to the FIELD parameters on the same RECORD statement. If you are creating a partitioned data set or PDSE, this parameter identifies the last record to be included in the partitioned data set or PDSE member named in the previous MEMBER statement. If the RECORD statement is not followed by additional RECORD or MEMBER statements, IDENT or IDENTG also defines the last record to be processed.

IDENT is used to identify a standard, single-byte character string. IDENTG is used to identify a double-byte character string.

The values for IDENT or IDENTG can be coded:

length

specifies the length (in bytes) of the identifying name. The length of your identifier cannot be greater than eight.

For IDENTG, the length must be an even number.

'name'

specifies the literal that identifies the last input record of a group of records. 'Name' must be coded within single apostrophes.

If you are using IDENTG, 'name' must be a double-byte character string. The DBCS string must be enclosed in shift-out/shift-in (SO/SI) characters. The SO/SI characters are not considered part of the literal specified by 'name', and they should not be included in the count for *length*. IEBGENER disregards the SO/SI characters when it looks for a match for 'name'.

'Name' can be specified in hexadecimal. To do so, code 'name' as *name* . Thus, if you do not have a keyboard that can produce certain characters, you can specify them in their hexadecimal versions. The values of the SO/SI characters are X'0E' and X'0F', respectively.

If no match for 'name' is found, the remainder of the input data is considered to be in one record group; subsequent RECORD and MEMBER statements are ignored.

input-location

specifies the starting position of the field that contains the identifying name in the input records. *Input-location* should be coded as a whole decimal number.

If you do not specify IDENT or IDENTG, all of the input data is considered to be in one record group. Only the first RECORD and MEMBER statements are used by IEBGENER.

FIELD=([length], [{input-location} 'literal'], [conversion],[output-location])

specifies field-processing and editing information. Only the contents of specified fields in the input record are copied to the output record; that is, any field in the output record that is not specified contains meaningless data.

Note that the variables on the FIELD parameter are positional; if any of the options are not coded, the associated comma preceding that variable must be coded.

The values that can be coded are:

length

specifies the length (in bytes) of the input field or literal to be processed. If *length* is not specified, a length of 80 is assumed. If a literal is to be processed, a length of 40 or less must be specified. If the data set is variable blocked, the length must be 4 bytes less than the LRECL of the output data set.

input-location

specifies the starting position of the field to be processed. *Input-location* should be coded as a whole decimal number. If *input-location* is not specified, it defaults to 1.

'literal'

specifies a literal (maximum length of 40 bytes) to be placed in the specified output location. If a literal contains apostrophes, each apostrophe must be written as two consecutive apostrophes.

You can specify a literal in hexadecimal by coding X'literal' You can also specify a double-byte character set string as the literal.

conversion

specifies a code that indicates the type of conversion to be performed on this field. If no conversion is specified, the field is moved to the output area without change. The values that can be coded are:

CG

specifies that shift-out/shift-in characters are to be removed, but that DBCS data is not to be validated. DBCS=YES must be specified on the GENERATE statement.

CV

specifies that DBCS data is to be validated, and that the input records contain single-byte character set data and double-byte. DBCS=YES must be specified on the GENERATE statement.

GC

specifies that shift-out/shift-in characters are to be inserted to enclose the DBCS data. DBCS=YES must be specified on the GENERATE statement.

GV

specifies that DBCS data is to be validated, and that the DBCS data is not enclosed by shift-out/shift-in characters. DBCS=YES must be specified on the GENERATE statement.

HE

specifies that H-set BCDIC data is to be converted to EBCDIC.

PZ

specifies that packed decimal data is to be converted to unpacked decimal data. Unpacking of the low-order digit and sign might result in an alphabetic character. This maximum length of an input packed decimal field is 16380 bytes.

VC

specifies that DBCS data is to be validated, and that shift-out/shift-in characters are to be inserted to enclose the DBCS data. DBCS=YES must be specified on the GENERATE statement.

VG

specifies that DBCS data is to be validated, and that shift-out/shift-in characters are to be eliminated from the records. DBCS=YES must be specified on the GENERATE statement.

ZP

specifies that unpacked decimal data is to be converted to packed decimal data.

When the ZP parameter is specified, the conversion is performed in place. The original unpacked field is replaced by the new packed field; therefore, the ZP parameter must be omitted from subsequent references to that field. If the field is needed in its original unpacked form, it must be referenced before the use of the ZP parameter.

If *conversion* is specified in the FIELD parameter, the length of the output record can be calculated for each conversion specification. When L is equal to the length of the input record, the calculation is made as follows:

- For a PZ (packed-to-unpacked) specification, $2L-1$.
- For a ZP (unpacked-to-packed) specification, $(L/2) + C$. If L is an odd number, C is 1/2; if L is an even number, C is 1.
- For an HE (H-set BCDIC to EBCDIC) specification, L.
- For the DBCS conversion codes, the shift-out/shift-in characters account for one byte each. If you add or delete them, account for the additional bytes.

output-location

specifies the starting location of this field in the output records. *Output-location* should be coded as a whole decimal number.

If *output-location* is not specified, the location defaults to 1.

LABELS=*n*

is an optional parameter that indicates the number of records in the SYSIN data set to be treated as user labels. The number *n*, which is a number from 0 to 8, must specify the exact number of label records that follow the RECORD statement. If this parameter is included, DATA=INPUT must be coded on a LABELS statement before it in the input stream.

IEBGENER Examples

The examples that follow illustrate some of the uses of IEBGENER. You can use Table 27 on page 142 as a quick-reference guide to IEBGENER examples. The numbers in the "Example" column refer to the examples that follow.

Table 27. IEBGENER example directory

Operation	Data Set Organization	Device	Comments	Example
PRINT	Sequential	Disk and Printer	Data set is listed on a printer.	“Example 1: Print a Sequential Data Set” on page 142
CONVERT	Sequential to Partitioned	Tape and Disk	Blocked output. Three members are to be created.	“Example 2: Create a Partitioned Data Set from Sequential Input” on page 142
MERGE	Sequential into Partitioned	Disk	Blocked output. Two members are to be merged into existing data set.	“Example 3: Convert Sequential Input into Partitioned Members” on page 143
COPY	Sequential	In-stream and Tape	Blocked output.	“Example 4: In-stream Input, Sequential Data Set to Tape Volume” on page 144
Copy and reblock	Sequential	Disk and Tape	Makes blocked tape copy from disk; explicit buffer request.	“Example 5: Produce Blocked Copy on Tape from Unblocked Disk File” on page 144
COPY—with editing	Sequential	Tape	Blocked output. Data set edited as one record group.	“Example 6: Edit and Copy a Sequential Input Data Set with Labels” on page 145
COPY—with editing	Sequential	z/OS UNIX file to Disk	Blocked output. New record length specified for output data set. Two record groups specified.	“Example 7: Edit and Copy a Sequential z/OS UNIX File to a Sequential Data Set” on page 146
COPY—with DBCS validation	Sequential	Disk	DBCS data is validated and edited before copying.	“Example 8: Edit Double-Byte Character Set Data” on page 147

Examples that use **disk** or **tape** in place of actual device numbers must be changed before use. The actual device numbers depend on how your installation has defined the devices to your system.

Example 1: Print a Sequential Data Set

In this example, a sequential data set is printed. The printed output is left-aligned, with one 80-byte record appearing on each line of printed output.

```
//PRINT JOB ...
//STEP1 EXEC PGM=IEBGNER
//SYSPRINT DD SYSOUT=A
//SYSIN DD DUMMY
//SYSUT1 DD DSN=DS001.DAT,DISP=SHR
//SYSUT2 DD SYSOUT=A
```

The job control statements are as follows:

- SYSIN DD defines a dummy data set. Since no editing is performed, no utility control statements are required.
- SYSUT1 DD defines the input sequential data set.
- SYSUT2 DD indicates that the output is to be written on the system output device (printer). IEBGENER copies LRECL and RECFM from the SYSUT1 data set and the system determines a BLKSIZE.

Example 2: Create a Partitioned Data Set from Sequential Input

In this example, a partitioned data set (consisting of three members) is created from sequential input.

```
//TAPEDISK JOB ...
//STEP1 EXEC PGM=IEBGNER
//SYSPRINT DD SYSOUT=A
//SYSUT1 DD DSN=INSET,UNIT=tape,LABEL=(,SL),
// DISP=(OLD,KEEP),VOLUME=SER=001234
```

```
//SYSUT2 DD DSN=NEWSET,UNIT=disk,DISP=(,KEEP),
//          VOLUME=SER=111112,SPACE=(TRK,(10,5,5)),
//          DCB=(RECFM=FB,LRECL=80,BLKSIZE=2000)
//SYSIN DD *
GENERATE MAXNAME=3,MAXGPS=2
MEMBER NAME=MEMBER1
GROUP1 RECORD IDENT=(8,'FIRSTMEM',1)
MEMBER NAME=MEMBER2
GROUP2 RECORD IDENT=(8,'SECNDMEM',1)
MEMBER NAME=MEMBER3
/*
```

The control statements are as follows:

- SYSUT1 DD defines the input data set (INSET). The data set is the first data set on a tape volume.
- SYSUT2 DD defines the output partitioned data set (NEWSET). The data set is to be placed on a disk volume. Ten tracks of primary space, five tracks of secondary space, and five blocks (256 bytes each) of directory space are allocated to allow for future expansion of the data set. The output records are blocked to reduce the space required by the data set.
- SYSIN DD defines the control data set, which follows in the input stream. The utility control statements are used to create members from sequential input data; the statements do not specify any editing.
- GENERATE indicates that three member names are included in subsequent MEMBER statements and that the IDENT parameter appears twice in subsequent RECORD statements.
- The first MEMBER statement assigns a member name (MEMBER1) to the first member.
- The first RECORD statement (GROUP1) identifies the last record to be placed in the first member. The name of this record (FIRSTMEM) appears in the first eight positions of the input record.
- The remaining MEMBER and RECORD statements define the second and third members. Note that, as there is no RECORD statement associated with the third MEMBER statement, the remainder of the input file will be loaded as the third member.

Example 3: Convert Sequential Input into Partitioned Members

In this example, sequential input is converted into two partitioned members. The newly created members are merged into an existing partitioned data set. User labels on the input data set are passed to the user exit routine.

```
//DISKTODK JOB ...
//STEP1 EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=A
//SYSUT1 DD DSN=INSET,UNIT=disk,DISP=(OLD,KEEP),
//          VOLUME=SER=111112,LABEL=(,SUL)
//SYSUT2 DD DSN=EXISTSET,UNIT=disk,DISP=(MOD,KEEP),
//          VOLUME=SER=111113
GENERATE MAXNAME=3,MAXGPS=1
EXITS INHDR=ROUT1,INTLR=ROUT2
MEMBER NAME=(MEMX,ALIASX)
GROUP1 RECORD IDENT=(8,'FIRSTMEM',1)
MEMBER NAME=MEMY
```

The control statements are as follows:

- SYSUT1 DD defines the input data set (INSET). The input data set, which resides on a disk volume, has standard user labels.
- SYSUT2 DD defines the output partitioned data set (EXISTSET). The members created during this job step are merged into the partitioned data set.
- The SYSIN DD statement is omitted. Because the GENERATE line does not begin with //, the system assumes it is preceded by a //SYSIN DD * line. SYSIN DD defines the control data set, which follows in the input stream. The utility control statements are used to create members from sequential input data; the statements do not specify any editing. A /* at the end of any DD * data set is unnecessary because a JCL statement or end of the job stream marks the end of the input stream data set.
- GENERATE indicates that a maximum of three names and aliases are included in subsequent MEMBER statements and that one IDENT parameter appears in a subsequent RECORD statement.

- EXITS defines the user routines that are to process user labels.
- The first MEMBER statement assigns a member name (MEMX) and an alias (ALIASX) to the first member.
- The RECORD statement identifies the last record to be placed in the first member. The name of this record (FIRSTMEM) appears in the first eight positions of the input record.
- The second MEMBER statement assigns a member name (MEMY) to the second member. The remainder of the input data set is included in this member.

Example 4: In-stream Input, Sequential Data Set to Tape Volume

In this example, an in-stream input, sequential data set is copied to a tape volume.

```
//CDTOTAPE JOB ...
//STEP1 EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=A
//SYSIN DD DUMMY
//SYSUT2 DD DSN=OUTSET,UNIT=tape,LABEL=(,SL),
// DISP=(,KEEP),VOLUME=SER=001234,
// DCB=(RECFM=FB,LRECL=80,BLKSIZE=2000)
//SYSUT1 DD *
// (in-stream data)
/*
```

The job control statements are as follows:

- SYSIN DD defines a dummy data set. No editing is performed; therefore, no utility control statements are needed.
- SYSUT2 DD defines the output data set, OUTSET. The data set is written to a tape volume with IBM standard labels. The data set is to reside as the first (or only) data set on the volume.
- SYSUT1 DD defines the in-stream data which is actually a JES SYSIN data set. The data set contains no statements.

Example 5: Produce Blocked Copy on Tape from Unblocked Disk File

In this example, a blocked copy on tape is made from an unblocked sequential disk file. Because the disk data set has a relatively small block size, the number of buffers explicitly requested is larger than the default of five. This improves performance by permitting more overlap of reading the SYSUT1 data set with writing the SYSUT2 data set.

```
//COPYJOB JOB
//STEP1 EXEC PGM=IEBGENER,REGION=318K
//SYSPRINT DD SYSOUT=A
//SYSIN DD DUMMY
//SYSUT1 DD DSN=INPUT,UNIT=disk,
// DISP=OLD,VOL=SER=X13380,
// DCB=(BUFNO=20,RECFM=F,LRECL=2000,BLKSIZE=2000)
//SYSUT2 DD DSN=OUTPUT,UNIT=tape,DISP=(NEW,KEEP),
// DCB=(RECFM=FB,LRECL=2000,BLKSIZE=32000)
```

The job control statements are as follows:

- The EXEC statement names the IEBGENER program and specifies the virtual storage region size required. (Calculation of region size is described in [Table 18 on page 99](#).)
- The SYSIN DD statement is a dummy, since no editing is to be performed.
- The SYSUT1 DD statement identifies an input disk file. Normally, the DCB RECFM, LRECL, and BLKSIZE information should not be specified in the DD statement for an existing disk file because the information exists in the data set label in the VTOC; it is specified in this example to illustrate the contrast with the output data set. The unit and volume serial information could be omitted if the data set were cataloged. The DCB information specifies BUFNO=20 to allow up to twenty blocks to be read with each rotation of the disk, assuming the disk track will hold that many blocks.

- The SYSUT2 DD statement identifies the output tape data set and specifies a block size of 32,000 bytes. The default of five buffers should be enough to keep pace with the input.

Example 6: Edit and Copy a Sequential Input Data Set with Labels

In this example, a sequential input data set is edited and copied.

```
//TAPETAPE JOB ...
//STEP1 EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=A
//SYSUT1 DD DSNAME=OLDSET,UNIT=tape,DISP=(OLD,KEEP),
//          VOLUME=SER=001234,LABEL=(3,SUL)
//SYSUT2 DD DSNAME=NEWSET,UNIT=tape,DISP=(NEW,PASS),
//          DCB=(RECFM=FB,LRECL=80),
//          VOLUME=SER=001235,LABEL=(,SUL)
//SYSIN DD *
GENERATE MAXFLDS=3,MAXLITS=11
RECORD FIELD=(10,'*****',,1),
        FIELD=(5,1,HE,11),FIELD=(1,'=',,16)
        EXITS INHDR=ROUT1,OUTTLR=ROUT2
        LABELS DATA=INPUT
        RECORD LABELS=2

(first header label record)
(second header label record)

RECORD LABELS=2

(first trailer label record)
(second trailer label record)

/*
```

The control statements are as follows:

- SYSUT1 DD defines the sequential input data set (OLDSET). The data set was originally written as the third data set on a tape volume.
- SYSUT2 DD defines the sequential output data set (NEWSET). The data set is written as the first data set on a tape volume. The output records are blocked to reduce the space required by the data set and to reduce the access time required when the data set is subsequently referred to. The BLKSIZE parameter is omitted so that the system will calculate an optimal value that is less than or equal to 32,760 bytes unless the system programmer sets the default differently. The data set is passed to a subsequent job step. The LABEL=(,SUL) is required because of the user labels created.
- SYSIN DD defines the control data set, which follows in the input stream.
- GENERATE indicates that a maximum of three FIELD parameters is included in subsequent RECORD statements and that a maximum of 11 literal characters are included in subsequent FIELD parameters.
- The first RECORD statement controls the editing, as follows: asterisks are placed in positions 1 through 10; positions 1 through 5 of the input record are converted from H-set BCDIC to EBCDIC mode and moved to positions 11 through 15; and an equal sign is placed in position 16.
- EXITS indicates that the specified user routines require control when SYSUT1 is opened and when SYSUT2 is closed.
- LABELS indicates that labels are included in the input stream.
- The second RECORD statement indicates that the next two records from SYSIN should be written out as user header labels on SYSUT2.
- The third RECORD statement indicates that the next two records from SYSIN should be written as user trailer labels on SYSUT2.

This example shows the relationship between the RECORD LABELS statement, the LABELS statement, and the EXITS statement. IEBGENER tries to write a first and second label trailer as user labels at close time of SYSUT2 before returning control to the system; the user routine, ROUT2, can review these records and change them, if necessary.

Related reading: For more information, see [“EXEC Statement”](#) on page 132.

Example 7: Edit and Copy a Sequential z/OS UNIX File to a Sequential Data Set

In this example, a z/OS UNIX System Services (z/OS UNIX) file is edited and copied. The logical record length of the output data set is less than that of the input data set.

```
//DISKDISK JOB ...
//STEP1 EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=A
//SYSUT1 DD PATH='/dist3/stor44/sales.mon',FILEDATA=TEXT,PATHOPTS=ORDONLY,
//          LRECL=100,BLKSIZE=1000,RECFM=FB
//SYSUT2 DD DSN=NEWSET,UNIT=disk,DISP=(NEW,KEEP),
//          VOLUME=SER=111113,DCB=(RECFM=FB,LRECL=80,
//          BLKSIZE=640),SPACE=(TRK,(20,10))
//SYSIN DD *
          GENERATE MAXFLDS=4,MAXGPS=1
          EXITS IOERROR=ERRORRT
GRP1 RECORD IDENT=(8,'FIRSTGRP',1),FIELD=(21,80,,60),FIELD=(59,1,,1)
GRP2 RECORD FIELD=(11,90,,70),FIELD=(69,1,,1)
/*
```

The control statements are as follows:

- SYSUT1 DD defines the input file. Its name is /dist3/stor44/sales.mon. It contains text in 100-byte records. The record delimiter is not stated here. The file might be on a non-System/390 system that is available via Network File System (NFS).
- SYSUT2 DD defines the output data set (NEWSET). Twenty tracks of primary storage space and ten tracks of secondary storage space are allocated for the data set on a disk volume. The logical record length of the output records is 80 bytes, and the output is blocked.
- SYSIN DD defines the control data set, which follows in the input stream.
- GENERATE indicates that a maximum of four FIELD parameters are included in subsequent RECORD statements and that one IDENT parameter appears in a subsequent RECORD statement.
- EXITS identifies the user routine that handles input/output errors.

Figure 25 on page 146 shows how a sequential input data set is edited and copied.

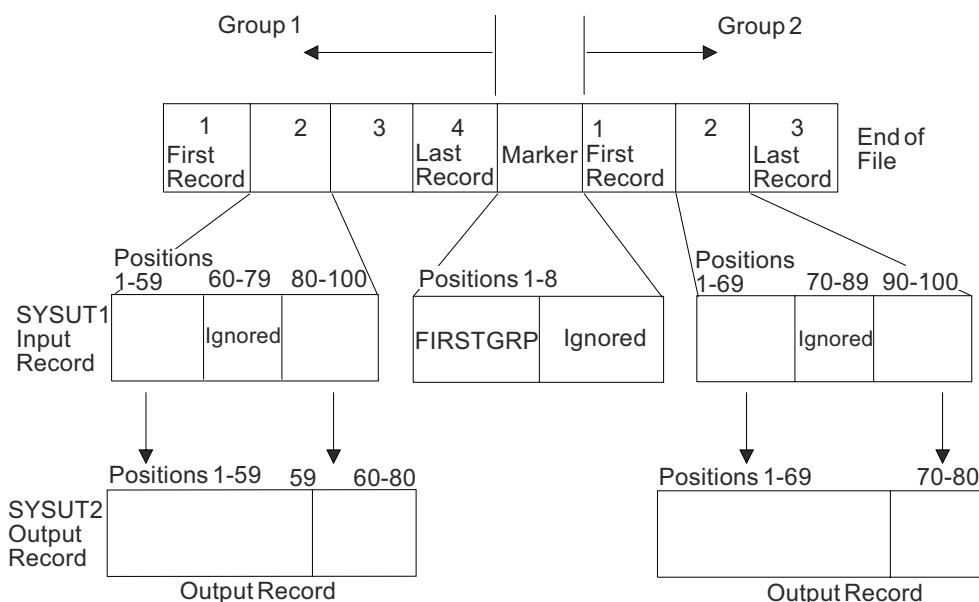


Figure 25. How a sequential data set is edited and copied

- The first RECORD statement (GRP1) controls the editing of the first record group. FIRSTGRP, which appears in the first eight positions of an input record, is defined as being the last record in the first group of records. The data in positions 80 through 100 of each input record are moved into positions 60 through 80 of each corresponding output record. (This example implies that the data in positions

60 through 79 of the input records in the first record group are no longer required; thus, the logical record length is shortened by 20 bytes.) The data in the remaining positions within each input record are transferred directly to the output records, as specified in the second FIELD parameter.

- The second RECORD statement (GRP2) indicates that the remainder of the input records are to be processed as the second record group. The data in positions 90 through 100 of each input record are moved into positions 70 through 80 of the output records. (This example implies that the data in positions 70 through 89 of the input records from group 2 are no longer required; thus, the logical record length is shortened by 20 bytes.) The data in the remaining positions within each input record are transferred directly to the output records, as specified in the second FIELD parameter.

Example 8: Edit Double-Byte Character Set Data

In this example, an edited data set containing double-byte character set data is created. Shift-out/shift-in characters (< and >) are inserted to enclose the DBCS strings.

```
//DBLBYTE JOB ...
//STEP1 EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=A
//SYSUT1 DD DSN=INPUT,DISP=(OLD,KEEP),UNIT=disk
//SYSUT2 DD DSN=OUTPUT,UNIT=disk,DISP=(NEW,CATLG),
// DCB=(LRECL=80,BLKSIZE=3200,RECFM=FB),SPACE=(TRK,(1,1))
//SYSIN DD *
GENERATE MAXFLDS=4,MAXLITS=9,DBCS=YES
RECORD FIELD=(20,1,1),FIELD=(16,33,VC,21),
FIELD=(30,50,VC,39),FIELD=(9,'*****',72)
/*
```

The control statements are as follows.

- SYSUT1 DD defines the input data set, INPUT, which resides on a disk volume.
- SYSUT2 DD defines the output data set, OUTPUT, which will reside on a disk volume.
- SYSIN DD defines the control data set, which follows in the input stream.
- GENERATE indicates that a maximum of four FIELD parameters and nine literal characters will appear on subsequent RECORD statements, and that the input data set contains DBCS data.
- RECORD specifies how input records will be edited before being placed in the output data set. The first FIELD parameter indicates that the first 20 positions (bytes) of the input records are to be placed in the first 20 positions of the output records.
- The second FIELD parameter indicates that data in positions 33 through 48 are to be checked to ensure that they are valid DBCS data, and that shift-out/shift-in characters are to be inserted around this field. For DBCS data to be valid, each byte of the 2-byte characters must have a hexadecimal value between X'41' and X'FE', or the 2-byte character must be a DBCS space (X'4040'). Once the checking and inserting are completed, this field is to be copied to the output records beginning at position 21.
- The third FIELD parameter operates on the 30-byte field beginning at position 50 in the input records. This field is checked for valid DBCS data, and shift-out/shift-in characters are inserted around the field. The resulting field is copied to the output records beginning at position 39.

Notice that in specifying the output locations in the FIELD parameter, you have to account for the additional positions that the SO/SI characters will use. For instance, the eight-character (16-byte) DBCS string beginning at position 21 does not end at position 36, but at 38. The SO/SI characters are single-byte characters, so the pair will take up two positions.

The final FIELD parameter clears out the final positions of the output records with asterisks.

Chapter 8. IEBIMAGE (Create Printer Image) Program

IEBIMAGE is a data set utility that creates and maintains the following types of IBM 3800 Printing Subsystem and IBM 4248 printer modules and stores them in a library:

- Forms control buffer modules for the 3800 and 4248 that specify controls for the vertical line spacing and any one of 12 channel codes per line.
- Copy modification modules for the 3800 that specify data that is to be printed on every page for specified copies of the output data set.
- Character arrangement table modules for the 3800 that translate the input data into printable characters and identify the associated character sets and graphic character modification modules.
- Graphic character modification modules for the 3800 that contain the scan patterns of characters that you design or characters from IBM-supplied modules.
- Library character set modules for the 3800 that contain the scan patterns of character sets that you define or IBM-supplied character sets.

The IEBIMAGE program creates and maintains all modules that are required for use on the 3800 Model 1 and Model 3 printers. The program default is to build these modules in the 3800 Model 1 format. However, 3800 Model 3 compatibility can be specified with IEBIMAGE utility control statements.

You can also use IEBIMAGE to create and maintain FCB modules for the 4248 printer. These modules are compatible with the 3262 Model 5 printer. However, the 3262 Model 5 does not support variable printer speeds or the horizontal copy feature of the 4248. Unless otherwise stated, where a reference to the 4248 printer is used in this topic, the 3262 Model 5 can be substituted.

Related reading: For information about creating images for printers, see [z/OS DFSMSdfp Advanced Services](#).

Storage Requirements for SYS1.IMAGELIB Data Set

The auxiliary storage requirement in tracks for SYS1.IMAGELIB is:

$$\text{Number of tracks} = (A + B) / T$$

where:

A

is the number of 1403 UCS images, 3211 UCS images, 3211 FCB images, 3525 data protection images, 3890 SCI programs, 3800 FCB modules, 4248 FCB images, 3262 Model 5 FCB images, 3800 character arrangement tables and 3800 library character sets (including images or modules supplied by you or IBM).

IBM supplies twelve 1403 UCS images, five 3211 UCS images, four 3211 FCB images, one 3800 FCB image, one 4245 UCS image table, one 4248 UCS image table, or eighty-four 3800 character arrangement tables, twenty 3800 Model 1 library character data sets, twenty 3800 Model 3, 6, and 8 library character sets, and graphic character modification modules.

Restriction: IBM supplies no 4245 or 4248 UCS images in SYS1.IMAGELIB. The 4245 and 4248 printers load their own UCS images into the UCS buffer at power-on time. IBM does supply 4245 and 4248 FCB images, which may be used.

B

is **(V + 600) / 1500** for each 3800 graphic character modification module and library character set module, each 3800 copy modification module, 4245 UCS image table, 4248 UCS image table, and each 3890 SCI program that is more than approximately 600 bytes.

V

is the virtual storage requirement in bytes for each module.

The virtual storage requirements for the IBM-supplied 3800 graphic character modification module containing the World Trade National Use Graphics are 32420 bytes for Model 1 and 55952 bytes for Model 3, 6, and 8. The virtual storage requirements for the IBM-supplied 3800 library character sets for the Model 1 are 4680 bytes and 8064 bytes for the Model 3, 6, and 8.

T

is the approximate number of members per track, depending on type of volume. Because of the overhead bytes and blocks in a load module, the difference in space requirements for an 80-byte module and a 400-byte module is small.

These constants assume an average member of 8 blocks, including a file mark, with a total data length of 800 bytes. For example, on a 3380 with 523 bytes of block overhead, the assumed average is 4984 bytes. If a different average member data length and average number of blocks per member are anticipated, these constants should reflect the actual number of members per track.

To determine the number of members per track, divide the average member length, including block overhead, into the track capacity for the device, see [Table 28 on page 150](#).

Table 28. Members per track (T) for various devices

T	Device Type
17	3380, all models
20	3390, all models
16	9345, all models

The result, $(A + B) / T$, is the track requirement.

The number of directory blocks for SYS1.IMAGELIB is given by the formula:

$$\text{Number of directory blocks} = (A + C + D) / 6$$

where:

A

is the same value as **A** in the track requirement calculation.

C

is the number of modules used to calculate **B**, when calculating the track requirement.

D

is the number of aliases. The IBM-supplied 1403 UCS images have four aliases and the IBM-supplied 3211 UCS images have six aliases. If you are not going to use these aliases, you can scratch them after the system is installed.

Related reading :

- For more information about printer-supplied UCS or FCB images, see [z/OS DFSMSdfp Advanced Services](#).
- For information about track capacity for DASD, see [z/OS DFSMS Macro Instructions for Data Sets](#).

Maintaining the SYS1.IMAGELIB Data Set

You will normally maintain SYS1.IMAGELIB using several programs in conjunction with IEBIMAGE. For example, you may find it necessary to rename or delete modules or to compress or list the entire contents of the data set. Programs such as PDF, DFSMSdss, IEBCOPY, IEBPTPCH, IEHLIST, and IEHPROGM should be used to help maintain SYS1.IMAGELIB. The program AMASPZAP can also be used for diagnosis purposes, and is described in [z/OS MVS Diagnosis: Tools and Service Aids](#).

If you use programs other than IEBIMAGE for maintenance, you must specify the full module name. The module's full name consists of a 4-character prefix followed by a 1- to 4-character name that you have assigned to it. It is thus a 5- to 8-character member name in the form:

FCB2xxxx

an FCB module that may be used with a 3203, 3211, 3262 Model 5, 4248, or 4245 printer. Note that the 4248 accepts FCBs that will also work with a 3203, 3211, 3262 Model 5, or 4245 printer. Also note that FCB2 modules cannot be written using IEBIMAGE, although IEBIMAGE can use FCB2 modules as input for creating FCB4 modules.

FCB3xxxx

a 3800 FCB module

FCB4xxxx

an FCB module that may be used with a 4248 or 3262 Model 5 printer

MOD1xxxx

a 3800 copy modification module

XTB1xxxx

a 3800 character arrangement table module

GRAFxxxx

a graphic character modification module for a 3800 Model 1

GRF2xxxx

a graphic character modification module for a 3800 Model 3

LCS1nn

a library character set module for a 3800 Model 1

LCS2nn

a library character set module for a 3800 Model 3

where:

xxxx

is the 1- to 4-character user-assigned name of the module.

nn

is the 2-character user-assigned ID of the module.

Alias names are not supported by IEBIMAGE, so you should be careful if you use them. For example, if you change a module by specifying its alias name, the alias name becomes the main name of the new module, and the old module is no longer accessible via the alias but is still accessible via its original main name.

Related reading: For information about maintaining and creating FCB2 modules, see [z/OS DFSMSdfp Advanced Services](#).

General Module Structure

Each module contains 8 bytes of header information preceding the data. For the 3800 printing subsystem, the general module header is shown in [Figure 26 on page 151](#).

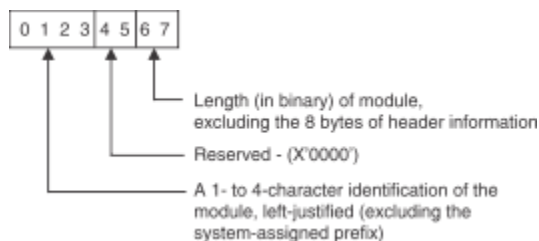


Figure 26. 3800 General Module Header

Header information for the 4248 printer FCB module is shown, with the module format, in [Figure 28 on page 153](#).

The SETPRT macro instruction uses the name to:

- Identify the module in the image library.
- Save the name to optimize future requests.

The SETPRT macro instruction uses the length to:

- Obtain sufficient storage for the module.
- Build channel programs to load the data into the printer.

Naming Conventions for Modules

Each module placed in a library by the IEBIMAGE utility has a 4-character system-assigned prefix as the first part of its name. These prefixes are described in [“Maintaining the SYS1.IMAGELIB Data Set”](#) on page 150.

You can assign a 1- to 4-character identifier (name) to the module you create by using the NAME control statement in the operation group you use to build the module. If the module is a library character set, the ID assigned to it must be exactly 2 characters. Each of those characters must be within the range 0 through 9, and A through F; the second character must represent an odd hexadecimal digit. However, the combinations X'7F' and X'FF' are not allowed. Except for library character set modules, this identifier is used in the JCL, the SETPRT macro instruction, or the character arrangement table to identify the module to be loaded.

While IEBIMAGE refers only to the 1- to 4-character name or the 2-character ID (the suffix) that is appended to the prefix, the full name must be used when using other utilities (such as IEBPTPCH or IEHPROGM).

Creating a Forms Control Buffer Module

The forms control buffer (FCB) module is of variable length and contains vertical line spacing information (6, 8, or 12 lines per inch for the 3800 Model 1; 6 or 8 lines per inch for the 4248; and 6, 8, 10, or 12 lines per inch for the 3800 Model 3). The FCB module can also identify one of 12 carriage-control channel codes for each line. For the 4248 printer, the module also contains information on the horizontal copy feature and the printer speed.

The FCB module is created and stored in an image library, using the FCB and NAME utility control statements. For the 4248 FCB module, the INCLUDE and OPTION statements can also be coded to indicate that an existing FCB module (prefix FCB2 or FCB4) is to be used as a model.

For the 3800, IBM supplies one default FCB image in SYS1.IMAGELIB, called FCB3STD1. For the 4248, although the last FCB image loaded is reloaded by the printer when the power is turned on, IBM supplies two FCB images that may also be used by printers other than the 4248. For the 3262 Model 5, a default FCB image is also supplied.

3800 FCB Module Structure

The FCB data following the header information is a series of 1-byte line control codes for each physical line of the form. There are 18 to 144 of these bytes, depending on the length of the form.

Each byte is a bit pattern describing one of 12 channel codes for vertical forms positioning and one of four lines-per-inch codes for vertical line spacing. The structure of the 3800 FCB module is shown in [Figure 27](#) on page 153.

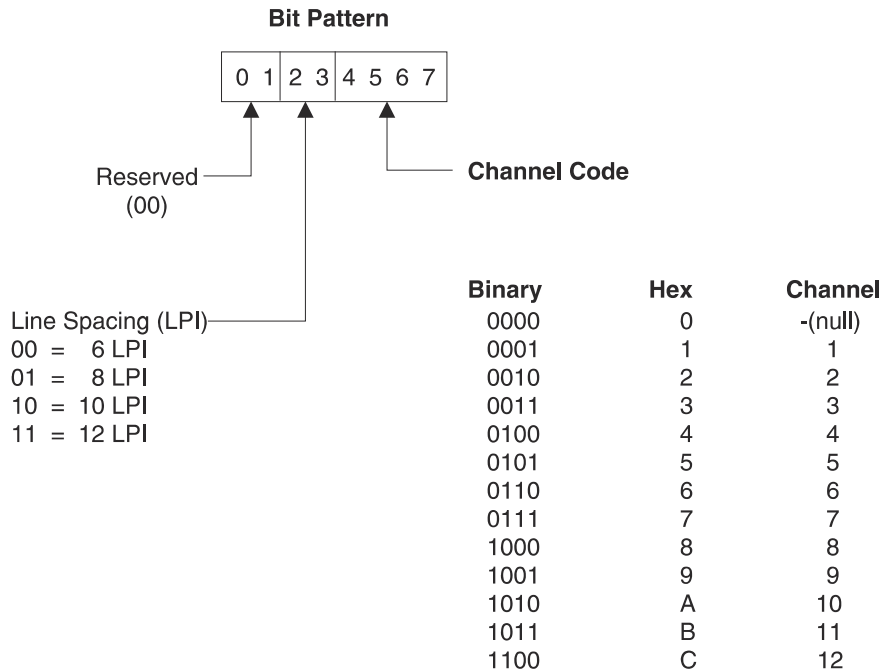


Figure 27. 3800 FCB module structure

- The first and last 1/2 inch of each page are unprintable, and the bytes corresponding to these positions must be void of any channel codes. Three bytes of binary zeros are supplied by the IEBIMAGE utility for the first and last 1/2 inch.
- The total number of lines defined in the module must be equal to the length of the form. The printable lines defined must start 1/2 inch below (after) the beginning and stop 1/2 inch from the end of the form.

4248 FCB Module Structure

The FCB data following the header information consists of at least five bytes: a flag byte (X'7E'), a control byte (containing information about the horizontal copy feature and printer speed), an offset byte, one or more FCB data bytes (similar to the 3800 data byte for each physical line of the form), and an end-of-sheet byte (X'FE'). The syntax of the 4248 FCB module is shown in the following figure:

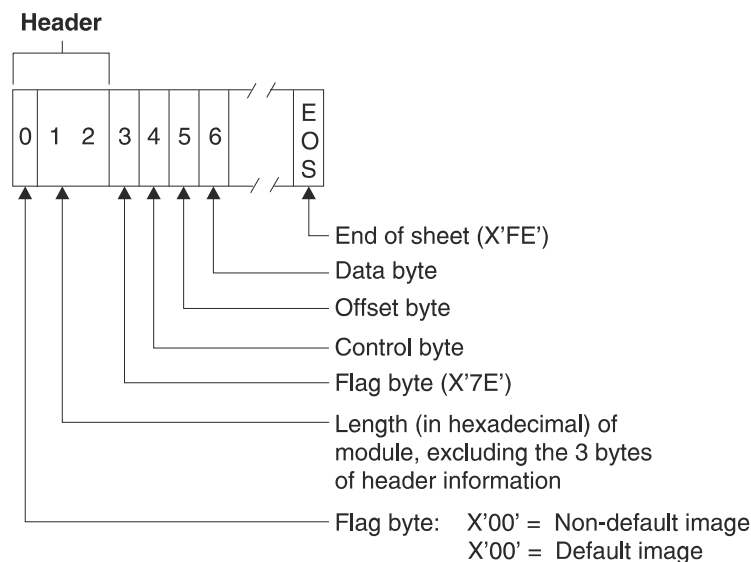


Figure 28. 4248 FCB module structure

The control byte is a bit pattern describing whether the horizontal copy feature is active and what printer speed is to be set when the FCB is loaded into the buffer. The structure of the control byte is shown in the following figure:

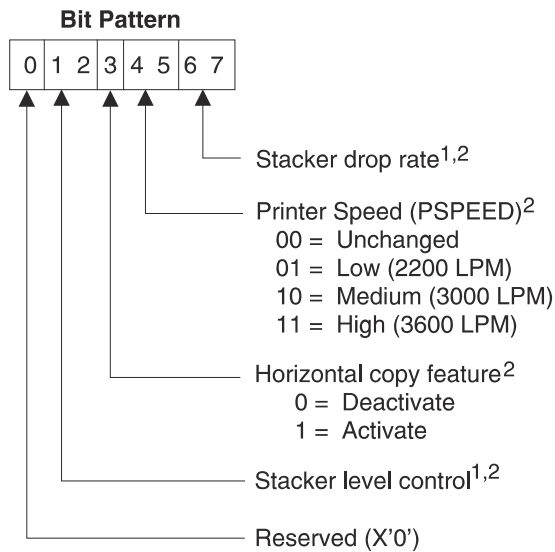


Figure 29. 4248 FCB module control byte

1

IEBIMAGE sets these bits to zero. For more information on the stacker drop rate and stacker level control bits, see the appropriate hardware manual for your printer.

2

If the module is used by a 3262 Model 5 printer, these bits are ignored.

The offset byte follows the control byte and is set either to zero or to the print position of the horizontal copy (2 through 168).

The data byte is a bit pattern similar to that produced for the 3800 printing subsystem. Each data byte describes one of 12 channel codes for vertical forms positioning and one of the allowed lines-per-inch codes for vertical line spacing. The structure of the data byte is shown in the following figure:

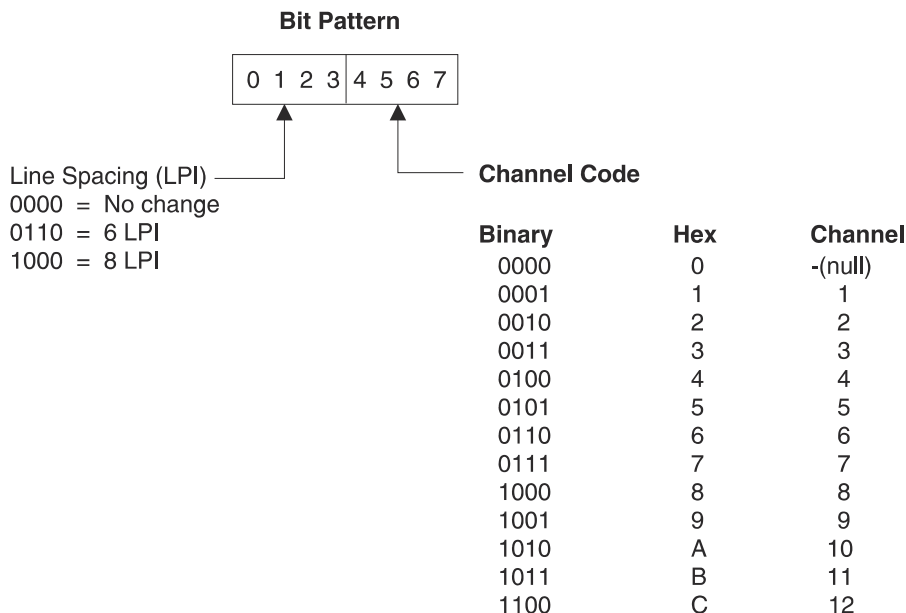


Figure 30. 4248 FCB module data byte

The total number of lines defined in the module must be equal to the length of the form.

FCB Module Listing

Figure 31 on page 155 shows the IEBIMAGE listing of a 3800 FCB module. The notes that follow the figure describe the encircled numbers in the figure.

For the 4248 FCB module, the IEBIMAGE listing also includes the horizontal copy feature, printer speed setting, and default settings.

```
PRINT LINE 1 AT 6 LINES PER INCH - HAS CHANNEL 1 CODE.
PRINT LINE 2 AT 8 LINES PER INCH
PRINT LINE 3 AT 8 LINES PER INCH
PRINT LINE 4 AT 12 LINES PER INCH
PRINT LINE 5 AT 12 LINES PER INCH
PRINT LINE 6 AT 12 LINES PER INCH
PRINT LINE 7 AT 12 LINES PER INCH
PRINT LINE 8 AT 12 LINES PER INCH
PRINT LINE 9 AT 12 LINES PER INCH
PRINT LINE 10 AT 12 LINES PER INCH
PRINT LINE 11 AT 12 LINES PER INCH
PRINT LINE 12 AT 12 LINES PER INCH
PRINT LINE 13 AT 12 LINES PER INCH
PRINT LINE 14 AT 12 LINES PER INCH
PRINT LINE 15 AT 12 LINES PER INCH
PRINT LINE 16 AT 12 LINES PER INCH
PRINT LINE 17 AT 12 LINES PER INCH
PRINT LINE 18 AT 12 LINES PER INCH
PRINT LINE 19 AT 12 LINES PER INCH
PRINT LINE 20 AT 12 LINES PER INCH
PRINT LINE 21 AT 12 LINES PER INCH
PRINT LINE 22 AT 12 LINES PER INCH
PRINT LINE 23 AT 12 LINES PER INCH
PRINT LINE 24 AT 12 LINES PER INCH
PRINT LINE 25 AT 12 LINES PER INCH
- - - - -
PRINT LINE 97 AT 12 LINES PER INCH
PRINT LINE 98 AT 12 LINES PER INCH
PRINT LINE 99 AT 12 LINES PER INCH
PRINT LINE 100 AT 12 LINES PER INCH - HAS CHANNEL 12 CODE.
PRINT LINE 101 AT 12 LINES PER INCH
PRINT LINE 102 AT 12 LINES PER INCH
PRINT LINE 103 AT 12 LINES PER INCH
PRINT LINE 104 AT 12 LINES PER INCH
PRINT LINE 105 AT 12 LINES PER INCH
PRINT LINE 106 AT 12 LINES PER INCH
PRINT LINE 107 AT 12 LINES PER INCH
PRINT LINE 108 AT 12 LINES PER INCH
PRINT LINE 109 AT 12 LINES PER INCH
PRINT LINE 110 AT 12 LINES PER INCH
PRINT LINE 111 AT 12 LINES PER INCH
PRINT LINE 112 AT 12 LINES PER INCH
PRINT LINE 113 AT 12 LINES PER INCH
PRINT LINE 114 AT 12 LINES PER INCH
PRINT LINE 115 AT 12 LINES PER INCH
PRINT LINE 116 AT 12 LINES PER INCH
PRINT LINE 117 AT 12 LINES PER INCH
PRINT LINE 118 AT 12 LINES PER INCH
```

Figure 31. IEBIMAGE listing of a forms control buffer module

1. The line number. Each line of the form is listed in this way.
2. The vertical spacing of the line, in lines per inch.

3. The channel code, which is printed for each line that includes a channel code.

Creating a Copy Modification Module

The 3800 copy modification module contains predefined data for modifying some or all copies of an output data set. Segments of the module contain predefined text, its position on each page of the output data set, and the copy or copies the text applies to.

The copy modification module is created and stored in an image library, using the INCLUDE, OPTION, COPYMOD, and NAME utility control statements.

The INCLUDE statement identifies a module that is to be copied and used as a basis for the newly created module. The OPTION statement with the OVERRUN parameter allows you to suppress the printing of line overrun condition messages for those vertical line spacings that are not applicable to the job. The OPTION statement with the DEVICE parameter specifies 3800 Model 3 compatibility mode processing. The COPYMOD statement is used to describe the contents of one of the new module's segments. The NAME statement is used to identify the new module and to indicate whether it is new or is to replace an existing module with the same name.

COPYMOD Module Structure

The copy modification data following the header information is a series of segments. Each segment is of variable length and is composed of the components shown in [Figure 32 on page 156](#).

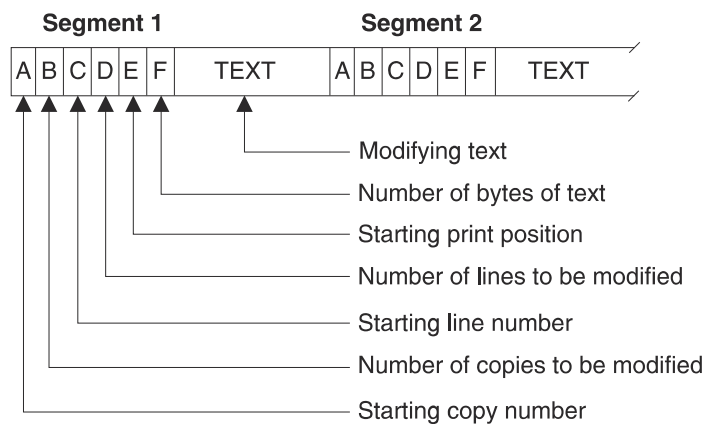


Figure 32. Structure of the copy modification module

A, B, C, D, E, and F are each 1-byte fields.

- If the module contains more than one segment, the starting copy number must be equal to or greater than the starting copy number in the previous segment.
- Any string of the same character within the text may be compressed into 3 bytes. The first such byte is X'FF', the second byte is the number of compressed characters, and the third byte is the data code for the character.
- The size of the module is limited to 8192 bytes of data and 8 bytes of header information.

COPYMOD Module Listing

Figure 33 on page 157 shows the listing of three segments of a copy modification module. This listing shows only the positioning of the modifying text. To print out the text itself, you can use the IEBPTPCH utility program or the AMASPZAP service aid. The numbered notes that follow the figure describe the items marked with the encircled numbers.

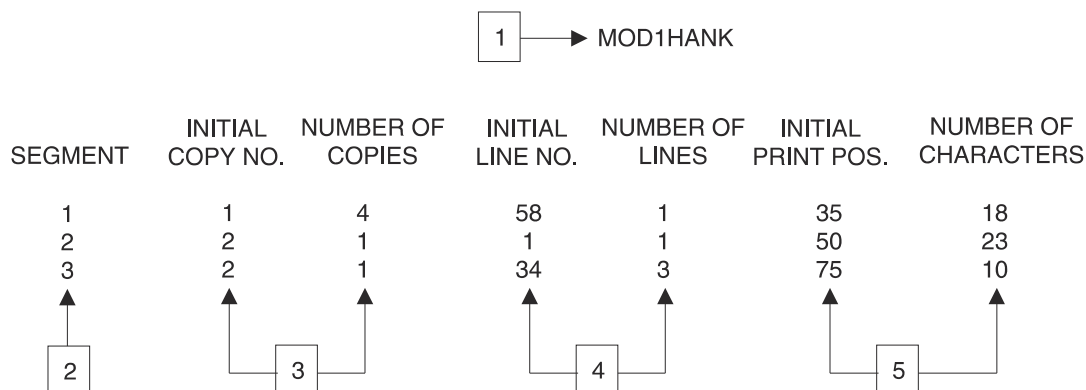


Figure 33. IEBIMAGE listing of three segments of a copy modification module

In this example, each note refers to the module's third segment.

1. The name of the copy modification module as it exists in the SYS1.IMAGELIB data set's directory (including the 4-byte system-assigned prefix).
2. The segment number of the modification segment.
3. This segment applies only to the second copy of the output data set.
4. The text of the segment is located on lines 34, 35, and 36.
5. The text on each line starts at the 75th character, and occupies 10 character spaces.

Creating a Character Arrangement Table Module

The 3800 character arrangement table module is fixed length and consists of three sections:

- System control information, which contains the module's name and length.
- The translation table, which contains 256 one-byte translation table entries, corresponding to the 8-bit data codes (X'00' through X'FF'). A translation table entry can identify one of 64 character positions in any one of four writable character generation modules (WCGMs) except the last position in the fourth WCGM (WCGM 3), which would be addressed by X'FF'. The code X'FF' is reserved to indicate an unprintable character. When an entry of X'FF' is detected by the printer as a result of attempting to translate an unusable 8-bit data code, the printer prints a blank and sets the data-check indicator on (unless the block-data-check option is in effect).
- Identifiers, which identify the character sets and the graphic character modification modules associated with the character arrangement table.

For the 3800 Model 1 or Model 3, if the character set identifier is even, the character set is accessed from the printer's flexible disk. If the identifier is odd, the character set is retrieved from the image library.

The character arrangement table is created using the INCLUDE, TABLE, and NAME utility control statements. The INCLUDE statement identifies an existing character arrangement table that is to be copied and used as a basis for the new module. The TABLE statement describes the contents of the new or modified module. The NAME statement identifies the character arrangement table and indicates whether it is new or is to replace an existing module with the same name.

The OPTION statement with the DEVICE=3800M3 parameter should be specified when printing an existing character arrangement table for a 3800 Model 3. This is to ensure that the system assigns the correct prefix to the graphic modification module name associated with the character arrangement table.

Note: The character arrangement table you select might not include all the characters in a character set. The character arrangement table corresponds to a print train, which is sometimes a subset of one or more complete character sets. When the character set is loaded, all characters of the set (up to 64) are loaded into the printer's WCGM; only those characters that are referred to by a translation table can be printed.

Related reading: For information about IBM-supplied character arrangement tables and character sets, see *IBM 3800 Printing Subsystem Programmer's Guide*.

TABLE Module Structure

The character arrangement table data following the header information is composed of the following components:

- A 256-byte translation table
- Four 2-byte fields for codes identifying character sets and their WCGM sequence numbers
- Four 4-byte fields for graphic character modification module names

The translation table consists of 256 one-byte entries, each pointing to one of 64 positions within one of four WCGMs:

- Bits 0 and 1 of each translation table byte refer to one of four WCGMs and bits 2 through 7 point to one of 64 addresses (0-63) within the WCGM. If SETPRT loads a character set into a WCGM other than the WCGM called for, SETPRT, using a copy of the translation table, alters bits 0 and 1 of each non-X'FF' byte of the translation table to correspond with the WCGM loaded. [Figure 34 on page 158](#) describes the structure of the character arrangement table module.

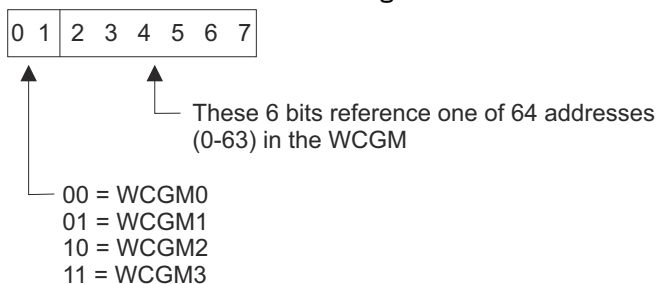


Figure 34. Module structure of the character arrangement table

- A byte value of X'FF' indicates an unusable character, prints as a blank, and gives a data check. The data check is suppressed if the block data check option is selected.
- One translation table can address multiple WCGMs, and multiple translation tables can address one WCGM. The translation tables supplied by IBM address either one or two WCGMs.

The next two components provide the linkage to character sets and graphic character modification modules. They consist of four 2-byte fields containing character set IDs with their corresponding WCGM sequence numbers, followed by four 4-character names of graphic character modification modules. The format is as follows:

- Each CGMID is a 1-byte character set ID containing two hexadecimal digits that refer to a library character set (as listed in *IBM 3800 Printing Subsystem Programmer's Guide*). Each WCGMNO refers to the corresponding WCGM sequence (X'00' to X'03'). Each name is the 4-character name of a graphic character modification module. [Figure 35 on page 158](#) shows the format of the Graphic Character Modification Modules.

CGMID0	WCGMNO0	CGMID1	WCGMNO1
CGMID2	WCGMNO2	CGMID3	WCGMNO3
Name1			
Name2			
Name3			
Name4			

Figure 35. Graphic character modification modules

- Most of the standard character arrangement tables do not need graphic character modification. The names are blank (X'40's) if no modules are referred to.

- The CGMIDx and the WCGMNOx are both 'X'00' when there are no character sets referred to after the first one.

TABLE Module Listing

The following figure shows the listing of a character arrangement table module. The numbered notes that follow the figure describe the items that are marked with the encircled numbers.

	X0	X1	X2	X3	X4	X5	X6	X7	X8	X9	XA	XB	XC	XD	XE	XF
0X	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
1X	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
2X	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
3X	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
4X 0 00	*	*	*	*	*	*	*	*	*	*	0 0A	0 0B	0 0C	0 0D	0 0E	0 0F
5X 0 10	*	*	*	*	*	*	*	*	*	*	0 1A	0 1B	0 1C	0 1D	0 1E	0 1F
6X 0 20 0 21	*	*	*	*	*	*	*	*	*	*	* 0 2B	0 2C	0 2D	0 2E	0 2F	
7X	*	*	*	*	*	*	*	*	*	*	0 3A	0 3B	0 3C	0 3D	0 3E	0 3F
8X	*	1 01	1 02	1 03	1 04	1 05	1 06	1 07	1 08	1 09	*	1 0D	1 0C	1 3C	1 3B	1 1A
9X	*	1 11	1 12	1 13	1 14	1 15	1 16	1 17	1 18	1 19	*	1 1D	0 2A	1 3D	1 0E	1 0F
AX 1 3A	1 10	1 22	1 23	1 24	1 25	1 26	1 27	1 28	1 29	*	1 2A	1 2C	1 0A	1 2E	1 0B	
BX 1 30	1 31	1 32	1 33	1 34	1 35	1 36	1 37	1 38	1 39	*	1 2D	1 2B	1 1B	1 21	1 1C	
CX	*	0 01	0 02	0 03	0 04	0 05	0 06	0 07	0 08	0 09	*	*	*	*	*	*
DX	*	0 11	0 12	0 13	0 14	0 15	0 16	0 17	0 18	0 19	*	*	*	*	*	*
EX	*	*	0 22	0 23	0 24	0 25	0 26	0 27	0 28	0 29	*	*	*	*	*	*
FX 0 30	0 31	0 32	0 33	0 34	0 35	0 36	0 37	0 38	0 39	*	*	*	*	*	*	*

CGM IDENTIFICATION ORDER

0 1 2 3

CGM IDENTIFICATION

BF 11 *

GRAPHIC MODIFICATION RECORDS

GRF2

Figure 36. IEBIMAGE listing of a character arrangement table module

Note:

1. The name of the character arrangement table module, as it exists in the directory of the image library (including the 4-byte system-assigned prefix).
2. The 1-byte identifier of an IBM-supplied character set (in this example, the Text 1 and Text 2 character sets, whose identifiers are X'8F' and X'11').

All character sets in SYS1.IMAGELIB or a user-specified image library are represented by odd-numbered identifiers. For a 3800 Model 3, if the character set identifier specified is even-numbered, it is increased by one at print time and the character set with that identifier is loaded.

3. The sequence number of the WCGM that is to contain the character set indicated following it (in this example, the second WCGM, whose identifier is 1).
4. The sequence number of the WCGM that contains the scan pattern for the 8-bit data code that locates this translation table entry.
5. Your 8-bit data code X'B9' transmitted to the 3800 Model 3 addresses this, the B9 location in the translation table, where the value X'39' in turn is the index into the WCGM that contains the scan pattern to be used. In this example, the scan pattern is Text 2 superscript 9.
6. An asterisk is shown in the listing for each translation table entry that contains X'FF'. This indicates that the 8-bit data code that addresses this location does not have a graphic defined for it and is therefore unprintable.
7. An asterisk in the list of character set identifiers indicates that no character set is specified to use the corresponding WCGM. If you specify 7F or FF as a character set identifier (to allow accessing a WCGM without loading it), a 7F or FF prints here.
8. The name of a graphic character modification module, as the name exists in the library's directory (including the system-assigned prefix).

When you specify a graphic character modification module to be associated with a character arrangement table, you must specify the OPTION statement with the DEVICE parameter (for the 3800 Model 3). Specifying the OPTION statement ensures that the system assigns the correct prefix (GRF2) to the graphic character modification module name.

Creating a Graphic Character Modification Module

The 3800 graphic character modification module is variable length and contains up to 64 segments. Each segment contains the 1 byte (for the 3800 Model 1) or 6 bytes (for the 3800 Model 3) of descriptive information and the 72-byte (for the 3800 Model 1) or 120-byte (for the 3800 Model 3) scan pattern of a graphic character.

The graphic character modification module is created using the INCLUDE, GRAPHIC, OPTION and NAME utility control statements.

The INCLUDE statement identifies an existing graphic character modification module that is to be copied and used as a basis for the new module.

To create graphic character modification modules in the syntax of the 3800 Model 3 compatibility mode module, the OPTION statement with the DEVICE parameter is required.

A GRAPHIC statement, when followed by one or more data statements, defines a user-designed character. A GRAPHIC statement can also select a character segment from another graphic character modification module. Each GRAPHIC statement causes a segment to be created for inclusion in the new module.

The NAME statement identifies the new module and indicates that the module is to be added to the library or is to replace an existing module of the same name. More than one GRAPHIC statement can be coded between the INCLUDE and NAME statements, and all such GRAPHIC statements apply to the same graphic character modification module.

GRAPHIC Module Structure

The graphic character modification data following the header information is a series of 73-byte segments for the 3800 Model 1 and 126-byte segments for the 3800 Model 3. A maximum of 64 such segments is allowed in a module. The module structure is shown in [Figure 37 on page 161](#).

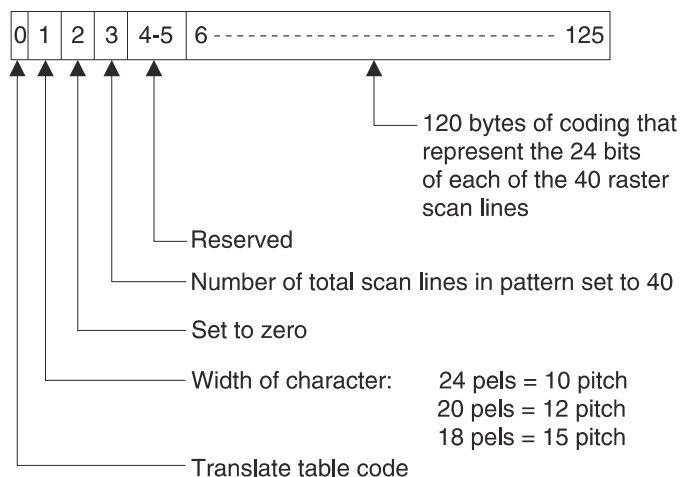


Figure 37. 3800 graphic character modification module structure for one character

When a graphic character is to be modified, the 3800 uses the translation table code to index into the translation table. The contents found at that location (a 1-byte WCGM code) determine the WCGM location into which the scan pattern and character data are to be placed.

For the 3800 Model 1 Printing Subsystem

The 72-byte graphic definition that makes up the scan pattern for one character is divided into twenty-four 3-byte groups. Each 3-byte group represents a horizontal row of eighteen 1-bit elements (plus parity information).

For the 3800 Model 3

The 120-byte graphic definition that makes up the scan pattern for one character is divided into forty 3-byte groups. Each 3-byte group represents a horizontal row of twenty-four 1-bit elements.

GRAPHIC Module Listing

Figure 38 on page 162 shows an extract from a listing of a graphic character modification module. This extract contains the listing of two segments of the module. Each of the notes following the figure describes the item in the figure that is marked with the encircled number.

A CHARSET statement, when followed by one or more data statements, defines a user-designed character. A CHARSET statement can also select a character segment from another library character set or from a graphic character modification module.

The NAME statement specifies the ID of the character set being created and indicates if it is to replace an existing module. More than one CHARSET statement can be coded between the INCLUDE and NAME statements; all such CHARSET statements apply to the same library character set module.

CHARSET Module Structure

The library character set data following the header information is a series of 73-byte segments for the 3800 Model 1 and 126-byte segments for the 3800 Model 3. Each module contains 64 segments. For each segment left undefined in a library character set module, IEBIMAGE inserts the graphic symbol for an undefined character. The structure of a library character set module is shown in [Figure 39 on page 163](#).

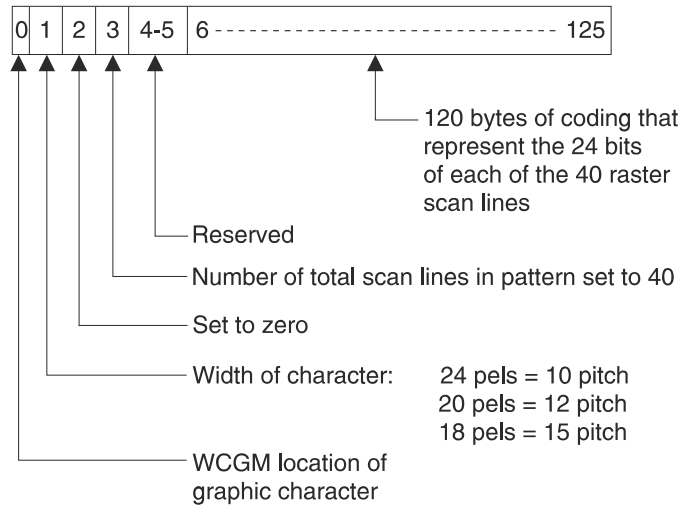


Figure 39. 3800 Model 3 library character set module structure for one character

A library character set is loaded directly into a WCGM. SETPRT uses the 6-bit code contained in the first byte of each 73-byte segment (for the 3800 Model 1) or 126-byte segment (for the 3800 Model 3) as the address of the WCGM location into which the remaining 72 bytes (for the 3800 Model 1) or 125 bytes (for the 3800 Model 3) are loaded.

For the 3800 Model 1

The 73-byte graphic definition that makes up the scan pattern for one character is divided into twenty-four 3-byte groups. Each 3-byte group represents a horizontal row of eighteen 1-bit elements.

For the 3800 Model 3

The 126-byte graphic definition that makes up the scan pattern for one character is divided into forty 3-byte groups. Each 3-byte group represents a horizontal row of twenty-four 1-bit elements.

CHARSET Module Listing

Figure 40 on page 164 shows an extract from a listing of a library character set module. This extract contains the listing of two segments of the library character set. The numbered notes that follow the figure describe the items marked with the encircled numbers.

- Forms control buffer modules (3800 or 4248)
- Copy modification modules (3800 only)
- Character arrangement table modules (3800 only)
- Graphic character modification modules (3800 only)
- Library character set modules (3800 only)

Note that, in building a 4248 FCB module, either a 4248 (prefix FCB4) or a 3211 (prefix FCB2) format FCB may be used as input. IEBIMAGE prefixes the name with FCB4 first; then, if no module exists with that name, the prefix is changed to FCB2. However, you cannot use IEBIMAGE to create an FCB2 module as output.

- An output data set listing for each new module, which includes:
 - Module identification
 - Utility control statements used in the job
 - Module contents
 - Messages and return codes

Related reading: For information about IEBIMAGE return codes, see [Appendix A, “Invoking Utility Programs from an Application Program,”](#) on page 315.

Control

IEBIMAGE is controlled by job and utility control statements.

Job Control Statements

Table 29 on page 165 shows the job control statements for IEBIMAGE.

<i>Table 29. Job control statements for IEBIMAGE</i>	
Statement	Use
JOB	Starts the job.
EXEC	Specifies the program name (PGM=IEBIMAGE) or, if the job control statements reside in the procedure library, the procedure name. No PARM parameters can be specified.
SYSPRINT DD	Defines the sequential message data set used for listing statements and messages on the system output device.
SYSUT1 DD	Defines the library data set (SYS1.IMAGELIB or a user-defined library).
SYSIN DD	Defines the control data set, which normally resides in the input stream.

SYSPRINT DD Statement

The block size for the SYSPRINT data set should be 121 or a multiple of 121. Any blocking factor can be specified. The first character of each 121-byte output record is an ISO/ANSI control character.

SYSUT1 DD Statement

To ensure that the library data set is not updated by other jobs while the IEBIMAGE job is running, DISP=OLD should be specified on the SYSUT1 DD statement.

SYSUT1 DD may specify a user image library. This library must have the same characteristics as SYS1.IMAGELIB, and can subsequently be identified to the system with the SETPRT macro instruction, or renamed "SYS1.IMAGELIB". If you are using JES, you cannot specify a user image library with SETPRT.

Related reading:

- For information about using your own image library with the 3800 Printing Subsystem, see *IBM 3800 Printing Subsystem Programmer's Guide*.
- For information about SETPRT, see [z/OS DFSMS Macro Instructions for Data Sets](#).

SYSIN DD Statement

The block size for the SYSIN data set must be 80 or a multiple of 80. Any blocking factor can be specified. DCB information for the SYSIN DD statement should be omitted from the JCL.

Utility Control Statements

IEBIMAGE is controlled by the utility control statements listed in [Table 30 on page 166](#).

Continuation requirements for utility control statements are discussed in [“Continuing utility control statements” on page 8](#).

Table 30. Utility control statements for IEBIMAGE

Statement	Use
FCB	Creates a 3800 or 4248 forms control buffer module and stores it in an image library.
COPYMOD	Creates a 3800 copy modification module and stores it in an image library.
TABLE	Creates a 3800 character arrangement table module and stores it in an image library.
GRAPHIC	Creates a 3800 graphic character modification module and stores it in an image library.
CHARSET	Creates a 3800 library character set module and stores it in an image library.
INCLUDE	Identifies an existing image library module to be copied and used as a basis for the new module.
NAME	Specifies the name of a new or existing library module.
OPTION	Specifies optional 3800 Model 3 or 4248 printer compatibility, or COPYMOD overrun lines per inch for an IEBIMAGE job.

Operation Groups

IEBIMAGE utility control statements are grouped together to create or print a library module. Each group of statements is called an **operation group**. Your job's input stream can include many operation groups. The operation groups (shown as follows without operands) that can be coded are:

- To create or print an FCB module:

```
[OPTION]
[INCLUDE]
FCB
NAME
```

Note: It is not possible to print a 4248 FCB module without coding some valid operation on the FCB statement.

- To create or print a copy modification module:

```
[INCLUDE]
[OPTION]
COPYMOD
[additional COPYMOD statements]
NAME
```

- To create or print a character arrangement table module:

```
[INCLUDE]
[OPTION]
TABLE
NAME
```

- To create or print a graphic character modification module:

```
[INCLUDE]
[OPTION]
{GRAPHIC|GRAPHIC, followed immediately by data statements}
[additional GRAPHIC statements]
NAME
```

- To create or print a library character set module:

```
[INCLUDE]
[OPTION]
{CHARSET|CHARSET, followed immediately by data statements}
[additional CHARSET statements]
NAME
```

To print a module, you need only supply the function statement (that is, FCB, COPYMOD, TABLE, GRAPHIC or CHARSET) with no operands specified, followed by the NAME statement naming the module. However, it is not possible to print a 4248 FCB module without coding some valid operation on the FCB statement.

FCB Statement

The FCB statement specifies the contents of a forms control buffer (FCB) module for the 3800, 3262 Model 5, or 4248 printer: spacing codes (lines per inch), channel codes (simulated carriage-control channel punches), and the size of the form. For the 4248 printer, the FCB statement also specifies print position for the horizontal copy feature and printer speed, and whether the FCB image is to be used as a default.

The FCB statement must always be followed by a NAME statement, and can only be preceded by an INCLUDE statement if DEVICE=4248 is specified on an OPTION statement.

An FCB statement with no operands specified, followed by a NAME statement that identifies a 3800 FCB module in the image library, causes the module to be formatted and printed. 3262 Model 5 and 4248 FCB modules cannot be printed by the FCB statement unless a valid operation is performed on them. To build an FCB module, you code the FCB statement with at least one operand. The format of a printed FCB module is shown in “FCB Module Listing” on page 155.

The syntax of the FCB statement is:

Label	Statement	Parameters
[label]	FCB	<pre>[LPI=((l[,n]) [, (l[,n])[,...]])] [, CHx=(line[,line [,...]])] [, CHx=(line[,line [,...]])[,...]] [, SIZE=length] [, LINES=lines] [, COPYP=position] [, PSPEED={L M H N}] [, DEFAULT={YES NO}]</pre>
Note: COPYP, PSPEED and DEFAULT can only be specified for a 4248 FCB module		

where:

LPI=((l[,n]) [, (l[,n])[,...]])

specifies the number of lines per inch and the number of lines to be printed at that line spacing.

l

specifies the number of lines per inch, and can be 6, 8, or 12 (for the 3800 Model 1); 6 or 8 (for the 3262 Model 5 or 4248); or 6, 8, 10, or 12 (for the 3800 Model 3).

n

specifies the number of lines at a line spacing of *l*. When the printer uses common-use paper sizes, *n* is a decimal value from 1 to 60 when *l* is 6; from 1 to 80 when *l* is 8; from 1 to 100 when *l* is 10; and from 1 to 120 when *l* is 12.

When the printer uses ISO paper sizes, *n* is a value from 1 to 66 when *l* is 6; from 1 to 88 when *l* is 8; from 1 to 110 when *l* is 10; or from 1 to 132 when *l* is 12. For the paper sizes, see *IBM 3800 Printing Subsystem Programmer's Guide*.

It is your responsibility to ensure that the total number of lines specified results in a length that is a multiple of 1/2 inch. That is 1.27cm.

The total number of lines cannot result in a value that exceeds the usable length of the form. For the 3800, do not specify coding for the first and last 1/2 inch of the form; IEBIMAGE does this for you.

When the SIZE, LINES, and LPI parameters are specified in the FCB statement, each parameter value is checked against the others to ensure that there are no conflicting page-length specifications. For example, SIZE=35 specifies a 3-1/2 inch length; acceptable LPI values for the 3800 cannot define more than the printable 2-1/2 inches of this length.

When you specify more than one (*l,n*) pair, *l* must be specified for each pair and *n* must be specified for each pair except the last.

When you specify 12 lines per inch, use one of the condensed character sets. If other character sets are printed at 12 lines per inch, the tops or bottoms of the characters may not print.

When only *l* is specified, or when *l* is the last parameter in the LPI list, all remaining lines on the page are at *l* lines per inch.

When LPI is not specified, all lines on the page are at 6 lines per inch.

If the total number of lines specified is less than the maximum number that can be specified, the remaining lines default to 6 lines per inch.

If INCLUDE is specified, the value for LPI may be taken from the included FCB module. See the discussion on the *module name* parameter for the INCLUDE statement.

CHx=(line[,line [...]])

specifies the channel code (or codes) and the line number (or numbers) to be skipped to when that code is specified.

CHx

specifies a channel code, where *x* is a decimal integer from 1 to 12.

line

specifies the line number of the print line to be skipped to, and is expressed as a decimal integer. The first printable line on the page is line number 1.

The value of *line* cannot be larger than the line number of the last printable line on the form.

Only one channel code can be specified for a print line. However, more than one print line can contain the same channel code.

Conventions:

- Channel 1 is used to identify the first printable line on the form. The job entry subsystem and the CLOSE routines for direct allocation to the 3800 with BSAM or QSAM require a channel 1 code even when the data being printed contains no skip to channel 1.
- Channel 9 is used to identify a special line. To avoid I/O interruptions that are caused by use of channel 9, count lines to determine the line position.
- Channel 12 is used to identify the last print line on the form to be used. To avoid I/O interruptions that are caused by use of channel 12, count lines to determine the page size.

- Use of an FCB that lacks a channel code to stop a skip operation causes a data check at the printer when the corresponding skip is issued. This data check cannot be blocked.

If INCLUDE is specified, values for CHx may be taken from the included FCB module. See the discussion under the *module name* parameter for the INCLUDE statement.

SIZE=length

specifies the vertical length of the form, in tenths of an inch. See *IBM 3800 Printing Subsystem Programmer's Guide* for the allowable lengths for the 3800. The complete length of the form is specified (for example, with the 3800, SIZE=110 for an 11-inch form) even though the amount of space available for printing is reduced by the 1/2-inch beginning and ending areas where no printing occurs.

When the SIZE, LINES and LPI keywords are specified in the FCB statement, each parameter value is checked against the others to ensure that there are no conflicting page-length specifications. For example, SIZE=35 specifies a 3-1/2 inch length; acceptable LPI values for the 3800 cannot define more than the printable 2-1/2 inches of this length.

When SIZE is not specified, the form length defaults to the value specified in LINES. If LINES is not specified, SIZE is assumed to be 11 inches (110).

If INCLUDE is specified, the value for SIZE may be taken from the included FCB module. See the discussion under the *module name* parameter for the INCLUDE statement.

LINES=lines

specifies the total number of lines to be contained in an FCB module.

lines

is the decimal number, from 1 to 256, which indicates the number of lines on the page.

When the LINES, SIZE, and LPI parameters are specified in the FCB statement, each parameter value is checked against the others to ensure that there are no conflicting page-length specifications.

When LINES is not specified, the form length defaults to the value of LPI multiplied by the value of SIZE, in inches. If no SIZE parameter is specified, LINES defaults to 11 times the value of LPI.

If INCLUDE is specified, the value for LINES may be taken from the included FCB module. See the discussion under the *module name* parameter of the INCLUDE statement.

COPYP=position

specifies the position (the number of character spaces from the beginning or left margin) at which the horizontal copy is to begin printing.

position

is a decimal number, from 2 to 168, which indicates where the horizontal copy printing will start.

If your 4248 printer has only 132 print positions, the maximum number you should specify here is 132.

If COPYP=0 is coded, any COPYP value previously set in an included FCB module is overridden, and the horizontal copy feature is turned off. You may not specify COPYP=1.

If INCLUDE is specified, and the included FCB module is formatted for a 4248 printer only, the default is the COPYP value for the included FCB module. Otherwise, if no COPYP value is specified, the default value is 0.

COPYP is not valid for 3800 FCB modules; it is ignored for 3262 Model 5 FCB modules.

The COPYP value specified affects the maximum amount of data that may be sent to the printer. Channel programs that are run with the horizontal copy feature activated **must** set the suppress incorrect length (SIL) bit and have a data length that does not exceed the size of either one half the number of print positions or the smaller of the two copy areas.

PSPEED={L|M|H|N}

specifies the print speed for the 4248 printer. Note that printer speed affects the quality of printing; LOW speed provides the best quality.

L or LOW

sets the printer speed to 2200 lines per minute (LPM).

M or MEDIUM

sets the printer speed to 3000 LPM.

H or HIGH

sets the printer speed to 3600 LPM.

N or NOCHANGE

indicates that the current printer speed should remain unchanged.

If INCLUDE is specified, and the included module is formatted for a 4248 printer only, the default is the PSPEED value for the included FCB module. Otherwise, the default is NOCHANGE (or N).

PSPEED is not valid for 3800 FCB modules. PSPEED is ignored for 3262 Model 5 FCB modules.

DEFAULT={YES|NO}

specifies if this 4248 FCB image is to be treated as the default image by OPEN processing. Default images are used by the system for jobs that do not request a specific image.

If a job does not request a specific FCB image, and the current image is not a default, the operator will be prompted for an FCB image at OPEN time.

If INCLUDE is used to copy a 4248 FCB module that was originally specified as a default image, the new module will also be considered a default image unless DEFAULT=NO is now specified.

DEFAULT is not valid for 3800 FCB images.

COPYMOD Statement

A copy modification module consists of header information, followed by one or more modification segments. The header information contains the module's name and length. Each modification segment contains the text to be printed, identifies the copy (or copies) the text applies to, and specifies the position of the text on each page of the copy.

A COPYMOD statement specifies the contents of one of the modification segments of a copy modification module. More than one COPYMOD statement can be coded in an operation group; all COPYMOD statements so coded apply to the same copy modification module.

IEBIMAGE analyzes the modification segments specified for a copy modification module to anticipate line overrun conditions that may occur when the module is used in the printer. A line overrun condition occurs when the modification of a line is not completed in time to print that line. The time available for copy modification varies with the vertical line spacing (lines per inch) at which the printer is being operated.

When IEBIMAGE builds a copy modification module from your specifications, the program calculates an estimate of the time the modification will require during the planned printing. If the modification can be done in the time available for printing a line at 12 LPI (lines per inch), it can also be done at 6 or 8 LPI (for the Model 1), or 6, 8, or 10 LPI (for the Model 3). (Note that 6, 8, 10 and 12 LPI are the only print densities available on the 3800 Model 3 printer.) However, if the copy modification module being built is too complex to be done in the time available for printing a line at 6 LPI, it certainly cannot be done at 8, 10 (for the Model 3 only), or 12 LPI. (Note that at 10 and 12 LPI there is much less time available for printing a line than at 6 LPI.)

When IEBIMAGE determines that a copy modification module is likely to cause an overrun if it is used when printing at a specified number of lines per inch, the program produces a warning message to that effect. If the warning applies to 6 LPI, the overrun condition is also applicable to 8, 10 (for the Model 3 only), and 12 LPI. If the warning applies to 8 LPI, the condition is also applicable for 10 (for the Model 3 only) and 12 LPI. If the warning applies to 10 LPI, the condition also applies to 12 LPI.

If you are planning to use a particular copy modification module only while printing at 6 LPI, you can request suppression of the unwanted warning messages for 8, 10 (for the Model 3 only), and 12 LPI by specifying the OPTION statement with 6 as the value of the OVERRUN parameter. If you are planning to print only at 8 LPI, you can use the OPTION statement with OVERRUN=8 to request suppression of the

unwanted warning messages for 10 (for the Model 3 only) and 12 LPI. The copy modification text can be printed using the same character size or style, or one different from the size or style used to print the data in the output data set.

Related reading:

- For more information on coding **OVERRUN**, see [“Using **OVERRUN**”](#) on page 180.
- For information about using your copy modification module, see *IBM 3800 Printing Subsystem Programmer's Guide*.

The **COPYMOD** statement must always be followed by a **NAME** statement or another **COPYMOD** statement and can be preceded by an **INCLUDE** statement. When more than one **COPYMOD** statement is coded, IEBIMAGE sorts the statements into order by line number within copy number. A **COPYMOD** statement with no operands specified, followed by a **NAME** statement that identifies a copy modification module, is used to format and print the module. The syntax of the printed module is shown under [“**COPYMOD** Module Listing”](#) on page 156.

The syntax of the **COPYMOD** statement, when used to create a copy modification module's segment, is:

Label	Statement	Parameters
[<i>label</i>]	COPYMOD	COPIES =(<i>starting-copy</i> [, <i>copies</i>]) , LINES =(<i>starting-line</i> [, <i>lines</i>]) , POS = <i>position</i> , TEXT =(([<i>d</i>] <i>t</i> , 'text') [, ([<i>d</i>] <i>t</i> , 'text')] [, ...])

where:

COPIES=(*starting-copy*[,*copies*])

specifies the starting copy number and the total number of copies to be modified.

starting-copy

specifies the starting copy number and is expressed as a decimal integer from 1 to 255. The *starting-copy* value is required.

copies

specifies the number of copies that are to contain the modifying text and is expressed as a decimal integer from 1 to 255. When *copies* is not specified, the default is 1 copy.

The sum of *starting-copy* and *copies* cannot exceed 256 (255 for JES3).

LINES=(*starting-line*[,*lines*])

specifies the starting line number and the total number of lines to be modified.

starting-line

specifies the starting line number, and is expressed as a decimal integer from 1 to 132. The *starting-line* value is required.

lines

specifies the number of lines that are to contain the modification segment's text, and is expressed as a decimal integer from 1 to 132. When *lines* is not specified, the default is 1 line.

The sum of *starting-line* and *lines* cannot exceed 133. If the sum exceeds the number of lines specified for the form size (in the **FCB** statement), the modifying text is not printed on lines past the end of the form.

POS=*position*

specifies the starting print position (the number of character positions from the beginning or left margin) of the modifying text.

position

specifies the starting print position and is expressed as an integer from 1 to 204. See the restriction noted for the **TEXT** parameter.

The maximum number of characters that can fit in a print line depends on the pitch of each character and the width of the form.

For the maximum number of characters that can fit in a print line for each form width, see *IBM 3800 Printing Subsystem Programmer's Guide*.

TEXT=([d]t, 'text') [, ([d]t, 'text')][, ...])

specifies the modifying text. The text is positioned on the form based on the LINES and POS parameters and replaces the output data set's text in those positions.

d

specifies a duplication factor (that is, the number of times the text is to be repeated). The d is expressed as a decimal integer from 1 to 204. If d is not specified, the default is 1.

t

specifies the form in which the text is entered: C for character, or X for hexadecimal. The t is required.

text

specifies the text and is enclosed in single quotation marks.

If the text type is C, you can specify any valid character. Blanks are valid characters. A single quotation mark is coded as two single quotation marks. You are not allowed to specify a character that results in a X'FF'. If the text type is X, the text is coded in increments of two characters that specify values between X'00' and X'FE'. You are not allowed to specify X'FF'.

The sum of the starting print position (see the POS parameter) and the total number of text characters cannot exceed 205. If the width of the form is less than the amount of space required for the text (based on character pitch, starting position, and number of characters), characters are not printed past the boundary (right margin) of the form.

If a text character specifies a character whose translation table entry contains X'FF', the printer sets the Data Check error indicator when the copy modification module is loaded. This error indicator can be blocked.

TABLE Statement

The TABLE statement is used to build a character arrangement table module. When a character arrangement table is built by IEBIMAGE and an INCLUDE statement is specified, the contents of the copied character arrangement table are used as a basis for the new character arrangement table. If an INCLUDE statement is not specified, each translation table entry in the new character arrangement table module is initialized to X'FF', the graphic character modification module name fields are set with blanks (X'40'), and the first character set identifier is set to X'83' (which is the Gothic 10-pitch set). The remaining identifiers are set to X'00'.

After the character arrangement table is initialized, IEBIMAGE modifies the table with data specified in the TABLE statement: character set identifiers, names of graphic character modification modules, and specified translation table entries. The character arrangement table, when built, must contain a reference to at least one printable character. Only one TABLE statement can be specified for each operation group. The TABLE statement can be preceded by an INCLUDE statement and an OPTION statement and must always be followed by a NAME statement.

A TABLE statement with no operands specified, followed by a NAME statement that identifies a character arrangement table module in the library, causes the module to be formatted and printed. The TABLE statement should be preceded by an OPTION statement with the DEVICE=3800M3 parameter for a 3800 Model 3. The format of the printed character arrangement table module is shown under [“TABLE Module Listing” on page 159](#).

The syntax of the TABLE statement is:

Label	Statement	Parameters
[label]	TABLE	[CGMID=(set0[, set1][, ...])] [, GCMLIST={ (gcm1[, gcm2][, ...]) DELETE}] [, LOC=((xloc[, {cloc[, setno] FF})[, ...])]

where:

CGMID=(set0[,set1][,...])

identifies the character sets that are to be used with the character arrangement table. (The IBM-supplied character sets and their identifiers are described in *IBM 3800 Printing Subsystem Programmer's Guide* .) When CGMID is specified, all character set identifiers are changed. If only one character set is specified, the other three identifiers are set to X'00'.

setx

is a 1-byte identifier of a character set. Up to four character set identifiers can be specified; set0 identifies the character set that is to be loaded into the first writable character generation module (WCGM); set1 is loaded into the second WCGM; and so forth. You should ensure that the character set identifiers are specified in the proper sequence, so that they are coordinated with the translation table entries.

GCMLIST={ (gcm1[, gcm2][,...]) | DELETE }

names up to four graphic character modification modules to be associated with the character arrangement table. When GCMLIST is specified, all graphic character modification module name fields are changed (if only one module name is specified, the other three name fields are set to blanks).

gcmx

is the 1- to 4-character name of the graphic character modification module. Up to four module names can be specified. The name is put into the character arrangement table, whether a graphic character modification module currently exists with that name. However, if the module does not exist, IEBIMAGE issues a warning message to you. The character arrangement table should not be used unless all graphic character modification modules it refers to are stored in an image library.

DELETE

specifies that all graphic character modification module name fields are to be set to blanks.

LOC=((xloc[, {cloc[,setno] | FF})[,...])

specifies values for some or all of the 256 translation table entries. Each translation table entry identifies one of 64 character positions within one of the WCGMs.

xloc

is an index into the translation table, and is specified as a hexadecimal value from X'00' to X'FF'; xloc identifies a translation table entry, not the contents of the entry.

cloc

identifies one of the 64 character positions within a WCGM, and is specified as a hexadecimal value between X'00' and X'3F'. When cloc is not specified, the default is X'FF', an incorrect character.

setno

identifies one of the WCGMs, and is specified as a decimal integer from 0 to 3. When setno is not specified, the default is 0. The setno cannot be specified unless cloc is also specified.

Cloc and setno specify the contents of the translation table entry located by xloc. You can specify the same cloc and setno values for more than one xloc.

GRAPHIC Statement

The GRAPHIC statement specifies the contents of one or more of the character segments of a graphic character modification module. A graphic character modification module consists of header information followed by from 1 to 64 character segments. Each character segment contains

- The character's 8-bit data code, its scan pattern, and its pitch (for the 3800 Model 1)
- Six bytes of descriptive information and the 120-byte scan pattern (for the 3800 Model 3)

By using the INCLUDE statement, you can copy an entire module, minus any segments deleted using the DELSEG keyword. In addition, you can select character segments from any module named with the GCM keyword on the GRAPHIC statement. The GRAPHIC statement can also specify the scan pattern and characteristics for a new character.

The GRAPHIC statement must always be followed by a NAME statement, another GRAPHIC statement, or one or more data statements. The OPTION statement with the DEVICE parameter must precede the GRAPHIC statement to create a graphic character modification module in the 3800 Model 3 compatibility mode module format. The GRAPHIC statement can be preceded by an INCLUDE statement. More than one GRAPHIC statement can be coded in the operation group. The operation group can include GRAPHIC statements that select characters from existing modules and GRAPHIC statements that create new characters. The GRAPHIC statement, preceded by an INCLUDE statement, can be used to delete one or more segments from the copy of an existing module to create a new module.

A GRAPHIC statement with no operands specified, followed by a NAME statement that identifies a graphic character modification module, is used to format and print the module. When you specify a graphic character modification module to be printed for a 3800 Model 3, you must specify the OPTION statement with the DEVICE parameter to ensure that the system assigns the correct prefix (GRF2) to the graphic character modification module name.

The syntax of the GRAPHIC statement, when it is used to select a character segment from another graphic character modification module, is:

Label	Statement	Parameters
[<i>label</i>]	GRAPHIC	[REF=((<i>segno</i> [, <i>xloc</i>]) [, (<i>segno</i> [, <i>xloc</i>])] [, ...]) [, GCM= <i>name</i>]]

where:

REF=((*segno*[,*xloc*]) [, (*segno*[,*xloc*])] [, ...])

identifies one or more character segments within an existing graphic character modification module. Each character segment contains the scan pattern for a character and the 6 bytes of descriptive information (used to locate its translate table entry). The 6 bytes of descriptive information can be respecified with the *xloc* subparameter. The REF parameter cannot be used to change a character's pitch or scan pattern.

segno

is the segment number, a decimal integer between 1 and 999. When a character segment is copied from the IBM-supplied World Trade National Use Graphics graphic character modification module, *segno* can be greater than 64. When the character segment is copied from a graphic character modification module built with the IEBIMAGE program, *segno* is a number from 1 to 64.

xloc

specifies an 8-bit data code for the character, and can be any value between X'00' and X'FF'. You should ensure that *xloc* identifies a translate table entry that points to a character position in the WCGM (that is, the translate table entry does not contain X'FF'). If *xloc* is not specified, the character's 8-bit data code remains unchanged when the segment is copied.

The REF parameter can be coded in a GRAPHIC statement that includes the ASSIGN parameter.

GCM=*name*

can be coded when the REF parameter is coded and identifies the graphic character modification module that contains the character segments referred to by the REF parameter.

name

specifies the 1- to 4-character user-specified name of the graphic character modification module.

If GCM is coded, REF must also be coded.

When GCM is not coded, the segments are copied from the IBM-supplied World Trade National Use Graphics graphic character modification module.

The syntax of the GRAPHIC statement, when it is used to specify the scan pattern and characteristics of a newly-created character, is:

Label	Statement	Parameters
[<i>label</i>]	GRAPHIC	ASSIGN=(<i>xloc</i> [, <i>pitch</i>]) <i>data statements</i> SEQ= <i>nn</i>

where:

ASSIGN=(*xloc* [, *pitch*])

identifies a newly-created character and its characteristics. The ASSIGN parameter specifies the new character's 8-bit data code and its pitch. When IEBIMAGE detects the ASSIGN parameter, it assumes that all following statements, until a statement without the characters SEQ= in columns 25 through 28 is encountered, are data statements that specify the character's scan pattern.

xloc

specifies the character's 8-bit data code, and can be any value between X'00' and X'FF'. You should ensure that *xloc* identifies a translation table entry that points to a character position in a WCGM (that is, the translation table entry does not contain X'FF'). The *xloc* is required when ASSIGN is coded.

pitch

specifies the character's horizontal size and is one of the decimal numbers 10, 12, or 15. If *pitch* is not specified, the default is 10.

At least one data statement must follow a GRAPHIC statement containing the ASSIGN parameter.

data statements

describe the design of the character as it is represented on a character design form. For details of how to design a character and how to use the character design form, see *IBM 3800 Printing Subsystem Programmer's Guide*.

Each data statement represents a line on the design form. Each nonblank line on the design form must be represented with a data statement; a blank line can also be represented with a data statement. You can code up to 24 (for 3800 Model 1) or 40 (for 3800 Model 3) data statements to describe the new character's pattern.

On each statement, columns 1 through 18 (for Model 1) or 24 (for Model 3) can contain nonblank grid positions when the character is 10-pitch. Any nonblank character can be punched in each column that represents a nonblank grid position. Columns 1 through 15 (for Model 1) or 20 (for Model 3) can contain nonblank grid positions when the character is 12-pitch. Columns 1 through 15 (for Model 1) or 1 through 16 (for Model 3) can contain nonblank grid positions when the character is 15-pitch.

SEQ=*nn*

specifies the sequence number that must appear in columns 25 through 30 of the data statement and identifies the line as a data statement; *nn* specifies a line number (corresponding to a line on the character design form) and is a 2-digit decimal number from 01 to 40.

CHARSET Statement

The CHARSET statement specifies the contents of one or more of the character segments of a library character set module. A library character set module consists of header information followed by 64 character segments. Each character segment contains the character's 6-bit code for a WCGM location, its scan pattern, and its pitch. You can use the INCLUDE statement to copy an entire module, minus any segments deleted using the DELSEG keyword. In addition, you can use the CHARSET statement to select

character segments from any module named with a library character set ID or the GCM keyword. The CHARSET statement can also specify the scan pattern and characteristics for a new character.

The CHARSET statement must always be followed by a NAME statement, another CHARSET statement, or one or more data statements. The CHARSET statement must be preceded by an OPTION statement with the DEVICE parameter if you want to create library character set modules in the 3800 Model 3 compatibility mode module format. The CHARSET statement can be preceded by an INCLUDE statement. More than one CHARSET statement can be coded in the operation group. The operation group can include CHARSET statements that select characters from existing modules and CHARSET statements that create new characters. The CHARSET statement, preceded by an INCLUDE statement, can be used to delete one or more segments from the copy of an existing module to create a new module.

A CHARSET statement with no operands specified, followed by a NAME statement that identifies a library character set module, is used to format and print the module.

The syntax of the CHARSET statement, when it is used to select a character segment from another module, is:

Label	Statement	Parameters
[<i>label</i>]	CHARSET	[REF=((<i>segno</i> , <i>cloc</i>) [, (<i>segno</i> , <i>cloc</i>)] [, . . .]) [, {GCM= <i>name</i> ID= <i>xx</i> }]]

where:

REF=((*segno*,*cloc*) [, (*segno*,*cloc*)] [, . . .])

identifies one or more character segments within an existing graphic character modification module or library character set module. If the reference is to a GCM, the scan pattern and pitch of the character referred to are used, and a 6-bit WCGM location code is assigned. If the reference is to a character in a library character set, the entire segment, including the 6-bit WCGM location code, is used, unless the *cloc* subparameter is specified for that segment. The REF parameter cannot be used to change a character's pitch or scan pattern.

segno

is the segment number, a decimal integer between 1 and 999. When a character segment is copied from the IBM-supplied World Trade National Use Graphics graphic character modification module, *segno* can be greater than 64. When the character segment is copied from a graphic character modification or library character set module built with the IEBIMAGE program, *segno* is a number from 1 to 64.

cloc

specifies a 6-bit code that points to a WCGM location, and can be any value between X'00' and X'3F'. When a library character set segment is referred to, if *cloc* is not specified, the character's 6-bit code remains unchanged when the segment is copied. If a graphic character modification segment is referred to, *cloc* must be specified.

The REF parameter can be coded in a CHARSET statement that includes the ASSIGN parameter.

GCM=*name*

can be coded when the REF parameter is coded and identifies the graphic character modification module that contains the character segments referred to by the REF parameter.

name

specifies the 1- to 4-character user-specified name of the graphic character modification module.

If GCM is coded, REF must also be coded. GCM should not be coded with ID.

When neither GCM nor ID is coded, the segments are copied from the IBM-supplied World Trade National Use Graphics graphic character modification module.

ID=*xx*

can be coded when the REF parameter is coded and identifies a library character set that contains the character segments referred to by the REF parameter.

xx

specifies the 2-hexadecimal-digit ID of the library character set module. The second digit must be odd, and '7F' and 'FF' are not allowed.

ID should not be coded with GCM.

When neither ID nor GCM has been coded, the segments are copied from the IBM-supplied World Trade National Use Graphics graphic character modification module.

The syntax of the CHARSET statement, when it is used to specify the scan pattern and characteristics of a newly-created character, is:

Label	Statement	Parameters
[<i>label</i>]	CHARSET	ASSIGN=(<i>cloc</i> [, <i>pitch</i>]) <i>data statements</i> SEQ= <i>nn</i>

where:

ASSIGN=(*cloc*[,*pitch*])

identifies a newly-created character and its characteristics. The ASSIGN parameter specifies the new character's 6-bit code and its pitch. When IEBIMAGE detects the ASSIGN parameter, the program assumes that all following statements, until a statement without the characters SEQ= in columns 25 through 28 is encountered, are data statements that specify the character's scan pattern.

cloc

specifies the character's 6-bit code for a WCGM location and can be any value between X'00' and X'3F'. *Cloc* is required when ASSIGN is coded.

pitch

specifies the character's horizontal size and is one of the following decimal numbers: 10, 12, or 15. If *pitch* is not specified, the default is 10.

At least one data statement must follow a CHARSET statement containing the ASSIGN parameter.

data statements

describe the design of the character as it is represented on a character design form. For details of how to design a character and how to use the character design form, see *IBM 3800 Printing Subsystem Programmer's Guide*.

Each data statement represents a line on the design form. Each nonblank line on the design form must be represented with a data statement; a blank line can also be represented with a data statement. You can code up to 24 (for 3800 Model 1) or 40 (for 3800 Model 3) data statements to describe the new character's pattern.

On each statement, columns 1 through 18 (for Model 1) or 24 (for Model 3) can contain nonblank grid positions when the character is 10-pitch. Any nonblank character can be punched in each column that represents a nonblank grid position. Columns 1 through 15 (for Model 1) or 20 (for Model 3) can contain nonblank grid positions when the character is 12-pitch. Columns 1 through 15 (for Model 1) or 1 through 16 (for Model 3) can contain nonblank grid positions when the character is 15-pitch.

SEQ=*nn*

specifies the sequence number that must appear in columns 25 through 30 of the data statement and identifies the line as a data statement; *nn* specifies a line number (corresponding to a line on the character design form) and is a 2-digit decimal number from 01 to 40.

INCLUDE Statement

When an IEBIMAGE operation group is used to create a new module, the INCLUDE statement can identify an existing image library module to be copied and used as a basis for the new module. When the operation group is used to update an image library module, the INCLUDE statement identifies the module to be referred to and must be specified.

- When the INCLUDE statement is coded in an operation group, it must precede any FCB, COPYMOD, TABLE, GRAPHIC, or CHARSET statements.
- Only one INCLUDE statement should be coded for each operation group. If more than one is coded, only the last is used; the others are ignored.
- You can code an INCLUDE statement for an FCB module only if the DEVICE=4248 parameter is specified on the OPTION statement. Either 3211 format or 4248 format FCBs may be included. IEBIMAGE tries to locate the 4248 format FCB first; if it is not found, IEBIMAGE looks for the 3211 format.
- You cannot copy a 3800 FCB module with INCLUDE.

The syntax of the INCLUDE statement is:

Label	Statement	Parameters
[<i>label</i>]	INCLUDE	<i>module name</i> [, DELSEG=(<i>segno</i> [, <i>segno</i>] [, . . .])]

where:

module name

names or identifies a library module. The module name is 1 to 4 alphanumeric and national (\$, #, and @) characters, in any order, or, for a library character set module, a 2-character ID that represents two hexadecimal digits (0-9, A-F), the second digit being odd. Note that 7F and FF cannot be used.

For a 3800 INCLUDE operation, the named module must be the same type as the module being created.

However, for the 4248 printer, if the named FCB module is not found to exist with the prefix FCB4, an existing 3211 FCB module (prefix FCB2) with the same module name will be used. In this case, the values specified for the LINES, SIZE, CHx, and LPI parameters on the FCB statement will default to the values previously specified in the included module if the new values are not compatible with the 3211 printer. If the 3211 module was a default image, the 4248 module will also be a default image unless the DEFAULT parameter is specified as NO.

DELSEG=(*segno* [, *segno*] [, . . .])

specifies the segments of the copied module that are to be deleted when the module is copied. Segment numbers can be specified in any order. In this parameter, segment 1 is used to refer to the first segment of the module. When you code the DELSEG parameter, you should use a current listing of the module's contents to ensure that you are correctly identifying the unwanted segments.

You can code the DELSEG parameter only when the named module is a copy modification module, a graphic character modification module, or a library character set module.

NAME Statement

The NAME statement can name a new library module to be built by the IEBIMAGE program. The NAME statement can also specify the name of an existing library module. The NAME statement is required, and must be the last statement in each operation group.

The syntax of the NAME statement is:

Label	Statement	Parameters
[<i>label</i>]	NAME	<i>module name</i> [(R)]

where:

module name

names or identifies a library module. The module name is 1 to 4 alphanumeric and national (\$, #, and @) characters, in any order, or, for a library character set module, a 2-character ID that represents two hexadecimal digits (0-9, A-F), the second digit being odd. Note that 7F and FF cannot be used.

If you are creating a 4248 FCB module, the name you specify will be prefixed with FCB4, even if you used a 3211 FCB module (prefix FCB2) as input on an INCLUDE statement. You cannot create or replace FCB2 modules with IEBIMAGE.

(R)

indicates that this module is to be replaced by a new module with the same name, if it exists. R must be coded in parentheses.

OPTION Statement

To create library character set modules and graphic character modification modules in a form usable on the 3800 Model 3, the OPTION statement with the DEVICE=3800M3 parameter is required. The OPTION statement with the DEVICE=3800M3 parameter is optional when creating copy modification modules and character arrangement table modules.

To create a forms control buffer module for the 3262 Model 5 or 4248 printer, the OPTION statement with the DEVICE=4248 parameter is required. DEVICE=4248 cannot be used to create any module other than an FCB.

The OPTION statement with the OVERRUN parameter is used only in a COPYMOD operation group and can be placed before or after any INCLUDE statement for the group. The value in the OVERRUN parameter specifies the greatest line density for which you want the overrun warning message IEBA33I to be printed. See [“Using OVERRUN” on page 180](#) for information about overrun conditions and suppression of overrun warning messages.

An effective use of the OPTION statement with the OVERRUN parameter would be to determine the greatest print-line density (6, 8, 10, 12) at which the copy modification module will be used, then specify that density in the OVERRUN parameter to eliminate the warning messages for higher line densities.

The OPTION statement applies only to the operation group that follows it. If used, the OPTION statement must be specified for each operation group in the job input stream.

The syntax of the OPTION statement is:

Label	Statement	Parameters
[label]	OPTION	[DEVICE={3800M3 4248}] [,OVERRUN={0 6 8 10 12}]

where:

DEVICE={3800M3|4248}

specifies printer compatibility mode module formats and processing considerations.

3800M3

specifies 3800 Model 3 compatibility.

4248

specifies that the module created or modified with the FCB statement should be formatted for the 3262 Model 5 or 4248 printer.

If the DEVICE parameter is omitted, modules are created for the 3800 Model 1.

OVERRUN={0|6|8|10|12}

specifies the greatest number of lines per inch for which message IEBA33I is to be printed for a COPYMOD operation. For example, OVERRUN=8 allows the message for 6 and 8 lines per inch, but suppresses it for 10 and 12 lines per inch. Specifying OVERRUN=0 suppresses message IEBA33I for every case. If you specify OVERRUN=12, none will be suppressed.

OVERRUN=10 is valid only for the 3800 Model 3.

If the OPTION statement is omitted, the OVERRUN parameter default value is 12, and messages are not suppressed. If the OVERRUN parameter is omitted, the default value is also 12.

If the parameter specification is incorrect (for instance, if `OVERRUN=16` is specified), the entire operation group does not complete successfully.

Related reading :

- For the syntax of the 4248 FCB module, see [Figure 29 on page 154](#).
- For information about using the `OVERRUN` parameter with `COPYMOD`, see [“Using OVERRUN” on page 180](#).

Using `OVERRUN`

[Table 31 on page 180](#) shows the listing of segments of a copy modification module where an overrun warning was in order. Even if the `OPTION` statement specifies `OVERRUN=0` and the overrun warning message is not printed, a note is printed for each segment description for which an overrun is possible.

Factors used in determining a line overrun condition are:

- Number of modifications per line
- Number of segments per module

Combining `COPYMOD` segments reduces the possibility of a line overrun condition.

Related reading: For the algorithm for calculating when a copy modification module may cause a line overrun condition, see *Reference Manual for IBM 3800 Printing Subsystem Model 1*.

Table 31. IEBIMAGE listing of a copy modification module with overrun notes

Notes	Segment	Initial Copy Number	Number of Copies	Initial Line Number	Number of Lines	Initial Print Position	Number of Characters
Note(0)“1” on page 180	1	1	200	10	96	10	180
Note(1)“2” on page 180	2	2	200	10	96	11	180
Note(1)“2” on page 180	3	3	200	10	96	12	180
Note(2)“3” on page 180	4	4	200	10	96	10	180
Note(2)“3” on page 180	5	5	200	10	96	11	180
Note(3)“4” on page 180	6	6	200	10	96	12	180
Note(3)“4” on page 180	7	7	200	10	96	10	180
Note(3)“4” on page 180	8	8	200	10	96	11	180
Note(3)“4” on page 180	9	9	200	10	96	12	180

Notes:

1. Indicates that you may have a copy modification overrun if you are printing at 12 LPI.
2. Indicates that you may have a copy modification overrun if you are printing at 8 LPI.
3. Indicates that you may have a copy modification overrun if you are printing at 8 or 12 LPI.
4. Indicates that you may have a copy modification overrun if you are printing at 6, 8, or 12 LPI; in other words, you may have an overrun at any LPI.

IEBIMAGE Examples

These examples illustrate some of the uses of IEBIMAGE. You can use [Table 32 on page 181](#) as a quick-reference guide to the examples that follow.

Usually, examples for the IBM 3800 Model 3 can be changed to IBM 3800 Model 1 examples by deleting the `OPTION DEVICE=3800M3` statement and specifying the `OVERRUN` parameter equal to a number other than 10.

Table 32. IEBIMAGE example directory

Module Created	Printer	Comments	Example
CHARSET	3800 Model 1	Entire library character set with scan patterns printed.	“Example 19: List a Library Character Set Module” on page 194
CHARSET	3800 Model 3	Segments copied from IBM-supplied GRAPHIC module.	“Example 20: Build a Library Character Set Module” on page 195
CHARSET	3800 Model 3	New module contains a user-designed character. Existing character arrangement (TABLE) modified to include new character.	“Example 21: Build a Library Character Set Module and Modify a Character Arrangement Table to Use It” on page 195
CHARSET	3800 Model 1	Segments copied from existing module. User-designed character created.	“Example 22: Build a Library Character Set Module from Multiple Sources” on page 196
COPYMOD	3800 Model 1	4 modification segments.	“Example 8: Build a New Copy Modification Module” on page 186
COPYMOD	3800 Model 3	Existing module used as basis for new module. <code>OVERRUN</code> specified.	“Example 9: Build a New Copy Modification Module from an Existing Copy” on page 187
FCB	3800 Model 1	11-inch form	“Example 1: Build a New 3800 Forms Control Buffer Module” on page 182
FCB	3800 Model 1	5-1/2 inch form, replaces existing SYS1.IMAGELIB member. Multiple channel codes specified.	“Example 2: Replace a 3800 Forms Control Buffer Module” on page 182
FCB	3800 Model 1	3-1/2 inch form, replaces existing SYS1.IMAGELIB member. Varied vertical spacing.	“Example 3: Replace a 3800 Forms Control Buffer Module” on page 183
FCB	3800 Model 1	7-inch form, varied vertical spacing.	“Example 4: Build a New 3800 Forms Control Buffer Module” on page 184
FCB	3800 Model 1	12-inch ISO form. Replaces IBM-supplied module.	“Example 5: Replace the 3800 Forms Control Buffer Module STD3” on page 184
FCB	3800 Model 3	7-1/2 inch ISO form. Varied vertical spacing.	“Example 6: Build a New 3800 Forms Control Buffer Module for Additional ISO Paper Sizes” on page 185
FCB	4248	11-inch form, based on existing module. New print speed and copy position specified.	“Example 7: Build a 4248 Forms Control Buffer Module” on page 185
GRAPHIC	3800 Model 1	Entire IBM-supplied module printed.	“Example 14: List the World Trade National Use Graphics Graphic Character Modification Module” on page 190
GRAPHIC	3800 Model 3	Segments copied from IBM-supplied module.	“Example 15: Build a Graphic Character Modification Module from the Character Modification Module World Trade GRAFMOD” on page 190

Table 32. IEBIMAGE example directory (continued)

Module Created	Printer	Comments	Example
GRAPHIC	3800 Model 3	New module contains a user-designed character. Existing character arrangement (TABLE) modified to include new character.	“Example 16: Build a New Graphic Character Modification Module and Modify a Character Arrangement Table to Use It” on page 191
GRAPHIC	3800 Model 1	Segments copied from existing module. User-designed character created.	“Example 17: Build a Graphic Character Modification Module from Multiple Sources” on page 192
GRAPHIC	3800 Model 3	New GRAPHIC module contains a user-designed character. Existing character arrangement (TABLE) modified to include new character. COPYMOD created to print new character. Result tested.	“Example 18: Define and Use a Character in a Graphic Character Modification Module” on page 193
TABLE	3800 Model 3	IBM-supplied module modified to include another character.	“Example 10: Add a New Character to a Character Arrangement Table Module” on page 187
TABLE	3800 Model 3	Existing module used as basis for new module. Pitch changed.	“Example 11: Build a New Character Arrangement Table Module from an Existing Copy” on page 188
TABLE	3800 Model 1	Existing module used as basis for new module. Includes user-designed characters of GRAPHIC module.	“Example 12: Build Graphic Characters in a Character Arrangement Table Module” on page 189
TABLE	3800 Model 3	Existing module used as basis for new module. New module deletes all GRAPHIC references and resets translation table entries.	“Example 13: Delete Graphic References From a Character Arrangement Table Module” on page 189

Example 1: Build a New 3800 Forms Control Buffer Module

3800 Model 1

In this example, the vertical spacing and channel codes for an 11-inch form are specified, and the module is added to the SYS1.IMAGELIB data set as a new member.

```
//FCBMOD1 JOB ...
//STEP1 EXEC PGM=IEBIMAGE
//SYSUT1 DD DSN=SYS1.IMAGELIB,DISP=OLD
//SYSPRINT DD SYSOUT=A
//SYSIN DD *
FCB CH1=1,CH12=80,LPI=8
NAME IJ
/*
```

The control statements are discussed as follows:

- The SYSUT1 DD statement includes DISP=OLD to ensure that no other job can modify the data set while this job is executing.
- CH1=1 specifies channel 1 code for line 1, allowing for positioning at line 1.
- CH12=80 specifies channel 12 code for line 80, allowing for positioning at line 80 and a unit exception indication at line 80 (the last printable line on the page.)
- LPI=8 specifies that the entire form is to be at a vertical spacing of 8 lines per inch. Because the SIZE parameter is omitted, the form length defaults to 11 inches. Because there are 10 inches of printable space in an 11-inch form, 80 lines are printed at 8 lines per inch.
- The name of the new FCB module is IJ; it is stored as a member of the SYS1.IMAGELIB data set.

Example 2: Replace a 3800 Forms Control Buffer Module

3800 Model 1

In this example, the size and channel codes for a 5-1/2 inch form are specified, and the module is added to the SYS1.IMAGELIB data set as a replacement for an existing member. The new module is added to the end of the data set; the name in the data set's directory is updated so that it points to the new module; the old module can no longer be accessed through the data set's directory.

```
//FCBMOD2 JOB ...
//STEP1 EXEC PGM=IEBIMAGE
//SYSUT1 DD DSN=SYS1.IMAGELIB,DISP=OLD
//SYSPRINT DD SYSOUT=A
//SYSIN DD *
FCB CH1=(1,7,13,20),CH12=26,SIZE=55
NAME S55(R)
/*
```

The control statements are discussed as follows:

- The SYSUT1 DD statement includes DISP=OLD to ensure that no other job can modify the data set while this job is executing.
- CH1=(1,7,13,20) specifies channel 1 code for printable line 1, line 7, line 13, and line 20.
- CH12=26 specifies channel 12 code for printable line 26.
- SIZE=55 specifies the length of the form as 55 tenths of an inch, or 5-1/2 inches.
- Because the LPI parameter is omitted, the vertical spacing defaults to 6 lines per inch. Because there are 4-1/2 inches of printable lines in a 5-1/2 inch form, there are 27 print lines on this form.
- The name of the FCB module is S55, and it replaces an existing FCB module of the same name. The new FCB module is stored as a member of the SYS1.IMAGELIB data set.

Example 3: Replace a 3800 Forms Control Buffer Module

3800 Model 1

In this example, the vertical spacing, channel codes, and size for a form are specified, and the module is added to the SYS1.IMAGELIB data set as a replacement for an existing member. The new module is added to the end of the data set; the name in the data set's directory is updated so that it points to the new module; the old module can no longer be accessed through the data set's directory.

```
//FCBMOD3 JOB ...
//STEP1 EXEC PGM=IEBIMAGE
//SYSUT1 DD DSN=SYS1.IMAGELIB,DISP=OLD
//SYSPRINT DD SYSOUT=A
//SYSIN DD *
FCB CH1=1,CH2=4,CH5=11,SIZE=35,LPI=((6,2),(8,3),(6,4),(8,9))
NAME HL(R)
/*
```

The control statements are discussed as follows:

- The SYSUT1 DD statement includes DISP=OLD to ensure that no other job can modify the data set while this job is executing.
- CH1=1 specifies channel 1 code for printable line 1.
- CH2=4 specifies channel 2 code for line 4.
- CH5=11 specifies channel 5 code for line 11.
- LPI=((6,2),(8,3),(6,4),(8,9)) specifies vertical spacing for the first 18 printable lines in the form:
 - (6,2) specifies lines 1 through 2 are at a vertical spacing of 6 lines per inch, and take up 2/6 inch.
 - (8,3) specifies lines 3 through 5 are at a vertical spacing of 8 lines per inch, and take up 3/8 inch.
 - (6,4) specifies lines 6 through 9 are at a vertical spacing of 6 lines per inch, and take up 4/6 inch.
 - (8,9) specifies lines 10 through 18 are at a vertical spacing of 8 lines per inch, and take up 1-1/8 inch.
- SIZE=35 specifies the length of the form as 35 tenths of an inch, or 3-1/2 inches. Because there are 2-1/2 inches of printable space on a 3-1/2 inch form, and because the LPI parameter specifies vertical spacing for 2-1/2 inches of lines, the vertical spacing of all lines in the form is accounted for.

- The name of the FCB module is HL; it replaces an existing module of the same name. The new FCB module is stored as a member of the SYS1.IMAGELIB data set.

Example 4: Build a New 3800 Forms Control Buffer Module

3800 Model 1

In this example, the vertical spacing, channel codes, and length of a form are specified, and the module is added to the SYS1.IMAGELIB data set as a new member.

```
//FCBMOD4 JOB ...
//STEP1 EXEC PGM=IEBIMAGE
//SYSUT1 DD DSN=SYS1.IMAGELIB,DISP=OLD
//SYSPRINT DD SYSOUT=A
//SYSIN DD *
FCB CH1=1,CH6=33,SIZE=70,LPI=((8,32),(12,2))
NAME TGT
/*
```

The control statements are discussed as follows:

- The SYSUT1 DD statement includes DISP=OLD to ensure that no other job can modify the data set while this job is executing.
- CH1=1 specifies channel 1 code for printable line 1.
- CH6=33 specifies channel 6 code for line 33.
- LPI=((8,32),(12,2)) specifies that the first 32 printable lines of the form are to be at a vertical spacing of 8 lines per inch, and the next 2 printable lines are to be at a vertical spacing of 12 lines per inch.
- SIZE=70 specifies that the length of the form is 70 tenths of an inch, or 7 inches. Because there are 6 inches of printable lines in a 7-inch form and the LPI parameter specifies 32 lines at 8 lines per inch, or 4 inches, and 2 lines at 12 lines per inch, or 1/6 inch, the vertical spacing for the remaining 1-5/6 inches defaults to 6 lines per inch.

Therefore, the form consists of lines 1 through 32 at 8 lines per inch, lines 33 through 34 at 12 lines per inch, and lines 35 through 45 at 6 lines per inch, with channel codes at line 1 and line 33.

- The name of the new FCB module is TGT; it is stored as a member of the SYS1.IMAGELIB data set.

Example 5: Replace the 3800 Forms Control Buffer Module STD3

3800 Model 1

In this example, an FCB module is defined that uses ISO paper sizes, replacing the IBM-supplied module named STD3. This must be done before the dump-formatting routines that print high-density dumps can print them at 8 lines per inch on that printer.

```
//FCBMOD5 JOB ...
//STEP1 EXEC PGM=IEBIMAGE
//SYSUT1 DD DSN=SYS1.IMAGELIB,DISP=OLD
//SYSPRINT DD SYSOUT=A
//SYSIN DD *
FCB CH1=1,CH12=88,LPI=(8,88),SIZE=120
NAME STD3(R)
/*
```

The control statements are discussed as follows:

- The SYSUT1 DD statement includes DISP=OLD to ensure that no other job can modify the data set while this job is executing.
- CH1=1 specifies channel 1 code for printable line 1; CH12=88 specifies channel 12 code for line 88.
- LPI=(8,88) specifies that all 88 printable lines of the form are to be at a vertical spacing of 8 lines per inch.
- SIZE=120 specifies that the length of the form is 120 tenths of an inch, or 12 inches, which is the longest ISO paper size.

- The name of the new FCB module is STD3; it is to replace the existing module of that same name on SYS1.IMAGELIB.

Example 6: Build a New 3800 Forms Control Buffer Module for Additional ISO Paper Sizes

3800 Model 1

In this example, an FCB module is defined that uses ISO paper sizes and has the ISO Paper Sizes Additional Feature installed.

```
//FCBMOD      JOB          ...                      72
//STEP1       EXEC        PGM=IEBIMAGE
//SYSUT1      DD          DSN=SYS1.IMAGELIB,DISP=OLD
//SYSPRINT    DD          SYSOUT=A
//SYSIN       DD          *
FCB  CH1=1,CH12=75,SIZE=85,
      LPI=((10,35),(12,4),(10,35),(6,1))          X
      NAME ARU
/*
```

The control statements are discussed as follows:

- The SYSUT1 DD statement includes DISP=OLD to ensure that no other job can modify the data set while this job is executing.
- CH1=1 specifies channel 1 code for line 1, allowing for positioning at line 1.
- CH12=75 specifies channel 12 code for line 75, allowing for positioning at line 75 and a unit exception indication at 75 (the last printable line on the page.)
- LPI=((10,35),(12,4),(10,35),(6,1)) specifies vertical spacing for the entire printable area on the form. The last printable line on the form must have vertical spacing of 6 lines per inch. The sum of the lines allocated must be a multiple of 1/2.

EXAMPLE

```
(10,35)=3 1/2"          (12,4)=2/6"          (6,1)=1/6"
and 3 1/2 + 2/6 + 3 1/2 + 1/6 = 7 1/2 which is a multiple of 1/2
```

- SIZE=85 specifies the length of the form as 85 tenths of an inch, or 8-1/2 inches, although the printable area is 7-1/2 inches.
- The name of the new FCB module is ARU; it is stored as a member of the SYS1.IMAGELIB data set.

Example 7: Build a 4248 Forms Control Buffer Module

In this example, a new 4248 default FCB module is built using an existing FCB module as a model. The new module, NEW1, is added to SYS1.IMAGELIB as a new member. The existing module, OLD1, remains unchanged. OLD1 may be a 4248 FCB called FCB4OLD1, or it may be a 3211 FCB called FCB2OLD1. (If both modules existed, FCB4OLD1 would be used.)

```
//FCBMOD7     JOB          ...
//STEP1       EXEC        PGM=IEBIMAGE
//SYSUT1      DD          DSN=SYS1.IMAGELIB,DISP=OLD
//SYSPRINT    DD          SYSOUT=A
//SYSIN       DD          *
OPTION        DEVICE=4248
INCLUDE       OLD1
FCB          COPYP=67,PSPEED=M,DEFAULT=YES
NAME         NEW1
/*
```

The control statements are discussed as follows:

- The SYSUT1 DD statement includes DISP=OLD to ensure that no other job can modify the data set while this job is executing.

- DEVICE=4248 on the OPTION statement specifies that this module is to be created for the 4248 printer.
- The INCLUDE statement specifies that a copy of the existing module OLD1 is to be used as a basis for the new module, NEW1.
- COPYP=67 indicates that the horizontal copy feature should be activated, and that horizontal copies should begin printing in the 67th print position from the beginning (left) margin. This setting overrides any COPYP value previously set in module OLD1; it applies to module NEW1, but does not change the value set in OLD1.

Note that the value 67 divides a 132-hammer printer into two equal copy areas for two equally-sized horizontal copies. With COPYP=67, a maximum of 66 bytes can be sent to the printer.

- PSPEED=M indicates that the printer speed should be set to medium (3000 LPM). This setting overrides any PSPEED value previously set in module OLD1; it applies to module NEW1, but does not change the value set in OLD1.
- DEFAULT=YES indicates that this module, NEW1, should become a default FCB module for this installation.
- Because these parameters are not specified, the values of LINES, SIZE, LPI, and CHx default to the values which already exist in module OLD1.
- The NAME statement indicates that this module should be called NEW1.

Example 8: Build a New Copy Modification Module

3800 Model 1

In this example, a copy modification module that contains four modification segments is built. The module is added to the SYS1.IMAGELIB data set as a new member.

```
//COPMOD1 JOB ...
//STEP1 EXEC PGM=IEBIMAGE
//SYSUT1 DD DSNAME=SYS1.IMAGELIB,DISP=OLD
//SYSPRINT DD SYSOUT=A
//SYSIN DD *
COPY1 COPYMOD COPIES=(1,1),LINES=(1,1),POS=50,
TEXT=(C,'CONTROLLER'S COPY')
COPY2A COPYMOD COPIES=(2,1),LINES=(1,1),POS=50,
TEXT=(C,'SHIPPING MANAGER'S COPY')
COPY2B COPYMOD COPIES=(2,1),LINES=(34,3),POS=75,
TEXT=(10C,' ')
COPYALL COPYMOD COPIES=(1,4),LINES=(58,1),POS=35,
TEXT=((C,'***'),(C,'CONFIDENTIAL'),(3X,'5C'))
NAME RT01
/*
```

The control statements are discussed as follows:

- The SYSUT1 DD statement includes DISP=OLD to ensure that no other job can modify the data set while this job is executing.
- The COPY1 COPYMOD statement specifies text that applies to each page of the first copy of the output data set:

LINES=(1,1) and POS=50 specify that the text is to be on the first printable line of each page, starting at the 50th print position from the beginning (left).

The TEXT parameter identifies each page of the copy as being the *Controller's Copy*.

- The COPY2A COPYMOD statement specifies text that applies to each page of the second copy of the output data set. The text is to be on the first line of each page, at the 50th print position from the left, with each page of the copy being the *Shipping Manager's Copy*.
- The COPY2B COPYMOD statement specifies that part of the second copy's output data set text is to be blanked out, so that the first, third, and subsequent copies contain information that is not printed on the second copy. The blank area is to be on lines 34, 35, and 36, beginning at the 75th print position from the left. The text on lines 34, 35, and 36, between print positions 75 and 84, is to be blank (that is, the character specified between the TEXT parameter's single quotation marks is a blank).

- The COPYALL COPYMOD statement specifies text that applies to the first four copies of the output data set. This example assumes that no more than four copies are printed each time the job that produces the output data set is processed. The text is to be on the 58th line on each page, at the 35th print position from the left. The legend "***CONFIDENTIAL***" is to be on each page of the copy. Note that the text can be coded in both character and hexadecimal format.
- The name of the copy modification module is RTO1; it is stored as a member of the SYS1.IMAGELIB data set.

Example 9: Build a New Copy Modification Module from an Existing Copy

3800 Model 1

In this example, a copy of an existing copy modification module, RTO1, is used as the basis for a new copy modification module. The new module is added to the SYS1.IMAGELIB data set as a new member. The existing module, RTO1, remains unchanged and available for use.

```

//COPMOD2 JOB          ...
//STEP1   EXEC PGM=IEBIMAGE
//SYSUT1  DD  DSNAME=SYS1.IMAGELIB,DISP=OLD
//SYSPRINT DD  SYSOUT=A
//SYSIN   DD  *
            INCLUDE  RTO1,DELSEG=1
            OPTION    OVERRUN=8,DEVICE=3800M3
            COPYMOD   COPIES=(2,3),LINES=(52,6),POS=100,
                      TEXT=(X,'404040404040405C5C')
            NAME      AP
/*

```

The control statements are discussed as follows:

- The SYSUT1 DD statement includes DISP=OLD to ensure that no other job can modify the data set while this job is executing.
- The INCLUDE statement specifies that a copy of the copy modification module named RTO1 is used as a basis for the new module, and that the first modification segment of RTO1 is to be deleted from the copy.
- OVERRUN=8 in the OPTION statement specifies that the IEBIMAGE program is to print a warning message if the copy modification could cause a line overrun condition when printing at 6 and 8 lines per inch. The program is also to suppress any warning messages that apply to printing at 10 and 12 lines per inch. DEVICE=3800M3 in the OPTION statement specifies 3800 Model 3 compatibility mode processing.
- The COPYMOD statement specifies text that applies to each page of the second, third, and fourth copies of the output data set:

LINES=(52,6) and POS=100 specify that the text is to be on the 52nd line and repeated for the 53rd through 57th lines of each page, starting at the 100th print position from the left (beginning).

The TEXT statement specifies the text in hexadecimal form: eight blanks followed by two asterisks (in this example, the assumption is made that X'40' prints as a blank and that X'5C' prints as an asterisk; in actual practice, the character arrangement table used with the copy modification module might translate X'40' and X'5C' to other printable characters).

- The name of the new copy modification module is AP; it is stored as a member of the SYS1.IMAGELIB data set.

Example 10: Add a New Character to a Character Arrangement Table Module

3800 Model 1

In this example, an IBM-supplied character arrangement table module is modified to include another character, and then added to the SYS1.IMAGELIB data set as a replacement for the IBM-supplied module.

```

//CHARMOD1 JOB          ...
//STEP1   EXEC PGM=IEBIMAGE

```

```
//SYSUT1 DD DSN=SYS1.IMAGELIB,DISP=OLD
//SYSPRINT DD SYSOUT=A
//SYSIN DD *
INCLUDE GF10
OPTION DEVICE=3800M3
TABLE LOC=((2A,2A),(6A,2A),(AA,2A),(EA,2A))
NAME GF10(R)
/*
```

The control statements are discussed as follows:

- The SYSUT1 DD statement includes DISP=OLD to ensure that no other job can modify the data set while this job is executing.
- The INCLUDE statement specifies that a copy of the character arrangement table named GF10 is to be used as a basis for the new module.
- The OPTION statement with the DEVICE parameter specifies 3800 Model 3 compatibility mode processing.
- The TABLE statement specifies updated information for four translation table entries: X'2A', X'6A', X'AA', and X'EA'. (These four locations are unused in the IBM-supplied GF10 table.) Each of the four translation table entries is to point to the '2A' (43rd character) position in the first WCGM, which contains the scan pattern for a lozenge.
- The name of the character arrangement table is GF10, and it is stored as a new module in the SYS1.IMAGELIB data set. The data set's directory is updated so that the name GF10 points to the new module; the old GF10 module can no longer be accessed through the data set's directory.

Example 11: Build a New Character Arrangement Table Module from an Existing Copy

3800 Model 1

In this example, an existing character arrangement table module is copied and used as a basis for a new module. The new character arrangement table is identical to the old one, except that it uses the Gothic 15-pitch character set instead of Gothic 10-pitch.

```
//CHARMOD2 JOB ...
//STEP1 EXEC PGM=IEBIMAGE
//SYSUT1 DD DSN=SYS1.IMAGELIB,DISP=OLD
//SYSPRINT DD SYSOUT=A
//SYSIN DD *
INCLUDE A11
OPTION DEVICE=3800M3
TABLE CGMID=87
NAME A115
/*
```

The control statements are discussed as follows:

- The SYSUT1 DD statement includes DISP=OLD to ensure that no other job can modify the data set while this job is executing.
- The INCLUDE statement specifies that a copy of the character arrangement table named A11 is to be used as a basis for the new module. The A11 character arrangement table translates 8-bit data codes to printable characters in the Gothic 10-pitch character set.
- The OPTION statement with the DEVICE parameter specifies 3800 Model 3 compatibility mode processing.
- The TABLE statement specifies a new character set identifier, X'87', which is the identifier for the Gothic 15-pitch character set. No other changes are made to the character arrangement table. The new table calls for characters in the Gothic 15-pitch character set.
- The name of the new character arrangement table is A115; it is stored as a member of the SYS1.IMAGELIB data set.

Example 12: Build Graphic Characters in a Character Arrangement Table Module

3800 Model 1

In this example, an existing character arrangement table module is copied and used as the basis for a new module that will include user-designed characters of a graphic character modification module. The new module is then added to the SYS1.IMAGELIB data set.

```
//CHARMOD3 JOB ...
//STEP1 EXEC PGM=IEBIMAGE
//SYSUT1 DD DSN=SYS1.IMAGELIB,DISP=OLD
//SYSPRINT DD SYSOUT=A
//SYSIN DD *
INCLUDE ONB
TABLE GCMLIST=ONB1,LOC=((6F,2F,1),(7C,3C,1),(6A,2A,0))
NAME ONBZ
/*
```

The control statements are discussed as follows:

- The SYSUT1 DD statement includes DISP=OLD to ensure that no other job can modify the data set while this job is executing.
- The INCLUDE statement specifies that a copy of the character arrangement table named ONB is to be used as a basis for the new module. ONB refers to two WCGMs.
- The TABLE statement identifies a graphic character modification module and stipulates the translation table entries for each of its segments:

GCMLIST=ONB1 identifies the graphic character modification module named ONB1. The LOC parameter specifies the translate table entry location, character position, and WCGM number for each segment of the module:

The first segment corresponds to the 8-bit data code X'6F'. The segments' scan pattern is to be loaded at character position X'2F' (that is, the 48th character position) in the second WCGM.

The second segment corresponds to the 8-bit data code X'7C'. The segment's scan pattern is to be loaded at character position X'3C' (that is, the 61st character position) in the second WCGM.

The third segment corresponds to the 8-bit data code X'6A'. The segment's scan pattern is to be loaded at character position X'2A' (that is, the 43rd character position) in the first WCGM.

The name of the new character arrangement table is ONBZ; it is stored as a new module in the SYS1.IMAGELIB data set.

Example 13: Delete Graphic References From a Character Arrangement Table Module

3800 Model 3

In this example, an existing character arrangement table module is copied and used as a basis for a new one. The new character arrangement table deletes references to all graphic character modification modules and resets the translate table entries that were used to point to character positions for the segments of a graphic character modification module.

```
//CHARMOD4 JOB ...
//STEP1 EXEC PGM=IEBIMAGE
//SYSUT1 DD DSN=SYS1.IMAGELIB,DISP=OLD
//SYSPRINT DD SYSOUT=A
//SYSIN DD *
INCLUDE ZYL
OPTION DEVICE=3800M3
TABLE GCMLIST=DELETE,LOC=((6A),(6B))
NAME ZYLA
/*
```

The control statements are discussed as follows:

- The SYSUT1 DD statement includes DISP=OLD to ensure that no other job can modify the data set while this job is executing.
- The INCLUDE statement specifies that a copy of the character arrangement table named ZYL is to be used as a basis for the new module.
- The OPTION statement with the DEVICE parameter specifies 3800 Model 3 compatibility mode processing.
- The TABLE statement deletes references to graphic character modification modules and resets two translation table entries:

GCMLIST=DELETE specifies that all names of graphic character modification modules included with the module when the ZYL character arrangement table was copied are to be reset to blanks (X'40').

The LOC parameter identifies two locations in the translation table, X'6A' and X'6B', that are to be set to X'FF' (the default value when no character position or WCGM values are specified).

- The name of the new character arrangement table is ZYLA; it is stored as a member of the SYS1.IMAGELIB data set.

Example 14: List the World Trade National Use Graphics Graphic Character Modification Module

3800 Model 1

In this example, each segment of the IBM-supplied graphic character modification module containing the World Trade National Use Graphics is printed. Each segment is unique, although the scan patterns for some segments are identical to other segment's scan patterns with only the 8-bit data code being different.

```
//GRAFMOD1 JOB ...
//STEP1 EXEC PGM=IEBIMAGE
//SYSUT1 DD DSN=SYS1.IMAGELIB,DISP=SHR
//SYSPRINT DD SYSOUT=A
//SYSIN DD *
GRAPHIC
NAME *
/*
```

The control statements are discussed as follows:

- DISP=SHR is coded because the library is not being updated.
- The World Trade National Use Graphics graphic character modification module is identified with the pseudonym of "*". The scan pattern of each of the characters in the module is printed.

Example 15: Build a Graphic Character Modification Module from the Character Modification Module World Trade GRAFMOD

3800 Model 3

In this example, a graphic character modification module is built. Its characters are segments copied from the World Trade National Use Graphics graphic character modification module. (See the *IBM 3800 Printing Subsystem Programmer's Guide* for the EBCDIC assignments for the characters.) The new module is stored in the SYS1.IMAGELIB system data set.

```
//GRAFMOD2 JOB ...
//STEP1 EXEC PGM=IEBIMAGE
//SYSUT1 DD DSN=SYS1.IMAGELIB,DISP=OLD
//SYSPRINT DD SYSOUT=A
//SYSIN DD *
OPTION DEVICE=3800M3
GRAPHIC REF=((24),(25),(26),(27),(28),(31),(33),(35),(38),(40))
NAME CSTW
/*
```

The control statements are discussed as follows:

- The SYSUT1 DD statement includes DISP=OLD to ensure that no other job can modify the data set while this job is executing.
- DEVICE=3800M3 in the OPTION statement specifies 3800 Model 3 compatibility mode module format.
- By not specifying the GCM keyword, the GRAPHIC statement identifies the World Trade National Use Graphics graphic character modification module. Ten of its segments are to be copied and used with the new module.
- The name of the graphic character modification module is CSTW; it is stored as a new module in the SYS1.IMAGELIB data set.

Example 16: Build a New Graphic Character Modification Module and Modify a Character Arrangement Table to Use It

3800 Model 3

In this example, a graphic character modification module is built. The module contains one user-designed character, a reverse 'E', whose 8-bit data code is designated as X'E0' and whose pitch is 10. An existing character arrangement table is then modified to include the reverse E.

```
//GRAFMOD3 JOB      ...
//STEP1    EXEC    PGM=IEBIMAGE
//SYSUT1   DD      DSNAME=SYS1.IMAGELIB,DISP=OLD
//SYSPRINT DD      SYSOUT=A
//SYSIN    DD      *
              OPTION  DEVICE=3800M3
              GRAPHIC  ASSIGN=(E0,10)
XXXXXXXXXXXXX SEQ=10
XXXXXXXXXXXXX SEQ=11
XXXXXXXXXXXXX SEQ=12
              XXXX   SEQ=13
              XXXX   SEQ=14
              XXXX   SEQ=15
              XXXX   SEQ=16
              XXXX   SEQ=17
              XXXX   SEQ=18
              XXXX   SEQ=19
              XXXX   SEQ=20
XXXXXXXXXXXXX SEQ=21
XXXXXXXXXXXXX SEQ=22
              XXXX   SEQ=23
              XXXX   SEQ=24
              XXXX   SEQ=25
              XXXX   SEQ=26
              XXXX   SEQ=27
              XXXX   SEQ=28
              XXXX   SEQ=29
XXXXXXXXXXXXX SEQ=30
XXXXXXXXXXXXX SEQ=31
XXXXXXXXXXXXX SEQ=32
              NAME    BODE
              INCLUDE  GS10
              OPTION   DEVICE=3800M3
              TABLE   CGMID=(83,FF),GCMLIST=BODE,LOC=(E0,03,1)
              NAME     RE10
/*
```

The control statements are discussed as follows:

- The SYSUT1 DD statement includes DISP=OLD to ensure that no other job can modify the data set while this job is executing.
- DEVICE=3800M3 in the OPTION statement preceding the GRAPHIC statement specifies 3800 Model 3 compatibility mode processing.
- The GRAPHIC statement's ASSIGN parameter establishes the 8-bit data code, X'E0', and the width, 10-pitch, for the user-designed character. The *data statements* that follow the GRAPHIC statement describe the character's scan pattern.
- The name of the graphic character modification module is BODE, and it is stored as a new module in the SYS1.IMAGELIB data set.

- The INCLUDE statement specifies that a copy of the GS10 character arrangement table is to be used as the basis for the new table.
- The TABLE statement specifies the addition of the reverse E to that copy of the GS10 table.

CGMID=(83,FF) specifies the character set identifier X'83' for the Gothic-10 set (which is the set already used by the GS10 table) and specifies X'FF' as a character set identifier to allow accessing of the second WCGM without loading it.

GCMLIST=BODE identifies the graphic character modification module containing the reverse E for inclusion in the table.

LOC=(E0,03,1) specifies that the reverse E, which has been assigned the 8-bit data code X'E0', is to be loaded into position X'03' in the second WCGM. Because this second WCGM is otherwise unused, any position in it could have been used for the reverse E.

- The new character arrangement table is named RE10; it is stored as a new module in the SYS1.IMAGELIB data set.

Example 17: Build a Graphic Character Modification Module from Multiple Sources

3800 Model 1

In this example, a graphic character modification module is created. Its contents come from three different sources: nine segments are copied from an existing module with the INCLUDE statement; the GRAPHIC statement is used to select another segment to be copied; the GRAPHIC statement is also used to establish characteristics for a user-designed character. The new graphic character modification module, when built, is added to the SYS1.IMAGELIB.

```
//GRAFMOD4 JOB ...
//STEP1 EXEC PGM=IEBIMAGE
//SYSUT1 DD DSN=SYS1.IMAGELIB,DISP=OLD
//SYSPRINT DD SYSOUT=A
//SYSIN DD *
INCLUDE CSTW,DELSEG=3
GRAPHIC REF=(1,6A),GCM=BODE,ASSIGN=9A
***** SEQ=06
***** SEQ=07
**** SEQ=08
*** SEQ=09
*** SEQ=10
*** ***** SEQ=11
*** ***** SEQ=12
*** ***** SEQ=13
*** ***** SEQ=14
*** ***** SEQ=15
*** ***** SEQ=16
*** ***** SEQ=17
*** ***** SEQ=18
*** ***** SEQ=19
NAME JPCK
/*
```

The control statements are discussed as follows:

- The SYSUT1 DD statement includes DISP=OLD to ensure that no other job can modify the data set while this job is executing.
- The INCLUDE statement specifies that a copy of the graphic character modification module named CSTW is to be included with the new module. All segments of CSTW, except the third segment (as a result of DELSEG=3), are to be copied into the new module and become the module's first through ninth modification segments.
- The GRAPHIC statement specifies the module's tenth and eleventh segments:

REF=(1,6A) and GCM=BODE specify that the 10th segment of the new module is to be obtained by copying the first segment from the graphic character modification module named BODE. In addition, the segment's 8-bit data code is to be changed so that its character is identified with the code X'6A'.

ASSIGN=9A specifies that the new module's 11th segment is a user-designed character whose 8-bit data code is X'9A' and whose width is 10-pitch (the default when no pitch value is specified). The GRAPHIC statement is followed by data statements that specify the character's scan pattern.

- The name of the graphic character modification module is JPCK, it is stored as a new module in the SYS1.IMAGELIB data set.

Example 18: Define and Use a Character in a Graphic Character Modification Module

3800 Model 3

In this example, a graphic character modification module containing a user-designed character is built. Next, a format character arrangement table is modified to include that new character. Then, a copy modification module is created to print the new character enclosed in a box of format characters. Finally, the result is tested to allow comparison of the output with the input.

```
//CHAR      JOB      ...
//BUILD     EXEC     PGM=IEBIMAGE
//SYSUT1    DD      DSNAME=SYS1.IMAGELIB,DISP=OLD
//SYSPRINT  DD      SYSOUT=A
//SYSIN     DD      *
          OPTION    DEVICE=3800M3
STEP1      GRAPHIC  ASSIGN=5C
XXX        XXX      SEQ=01
XXX        XXX      SEQ=02
XXX        XXX      SEQ=03
XXX        XXX      SEQ=04
XXXXXXXXXXXXXXXXXXXXXXXXX SEQ=05
XXXXXXXXXXXXXXXXXXXXXXXXX SEQ=06
XXXXXXXXXXXXXXXXXXXXXXXXX SEQ=07
XXX        XXX      SEQ=08
XXX        XXX      SEQ=09
XXX        XXX      SEQ=10
XXX        XXX      SEQ=11
          SEQ=12
          SEQ=13
          SEQ=14
XXXXXXXXXXXXXXXXXXXXXXXXX SEQ=15
XXXXXXXXXXXXXXXXXXXXXXXXX SEQ=16
XXXXXXXXXXXXXXXXXXXXXXXXX SEQ=17
XXX        XXX      XXX SEQ=18
XXX        XXX      XXX SEQ=19
XXX        XXX      XXX SEQ=20
XXX        XXX      XXX SEQ=21
XXXX       XXXXX     XXXX SEQ=22
XXXX       XXXXXXX    XXXX SEQ=23
          XXXXXXXX    SEQ=24
          XXXXX      XXXXXX SEQ=25
          SEQ=26
          SEQ=27
          SEQ=28
XXXXXXXXXXXXXXXXXXXXXXXXX SEQ=29
XXXXXXXXXXXXXXXXXXXXXXXXX SEQ=30
XXXXXXXXXXXXXXXXXXXXXXXXX SEQ=31
          XXXXXXXX    SEQ=32
          XXXXXXXX    SEQ=33
XXXXXXXXXXXXXXXXXXXXX     SEQ=34
XXXXXXXXXXXXXXXXXXXXX     SEQ=35
          XXXXXXXX    SEQ=36
          XXXXXXXX    SEQ=37
XXXXXXXXXXXXXXXXXXXXXXXXX SEQ=38
XXXXXXXXXXXXXXXXXXXXXXXXX SEQ=39
XXXXXXXXXXXXXXXXXXXXXXXXX SEQ=40
          NAME AIBM
```

```
STEP2      OPTION    DEVICE=3800M3
          INCLUDE    FM10
          TABLE     GCMLIST=AIBM,LOC=(5C,2C)
          NAME       BIBM
STEP3      OPTION    DEVICE=3800M3
          COPYMOD     COPIES=1,LINES=58,POS=5,TEXT=(C,'W6X')
          COPYMOD     COPIES=1,LINES=59,POS=5,TEXT=(C,'7*7')
```

```

                COPYMOD COPIES=1,LINES=60,POS=5,TEXT=(X,'E9F6E8')
                NAME     CIBM
/*
//TEST          EXEC PGM=IEBIMAGE
//SYSUT1        DD  DSN=SYS1.IMAGELIB,DISP=OLD
//SYSPRINT      DD  SYSOUT=A,CHARS=(GF10,BIBM),
//              MODIFY=(CIBM,1)
//SYSIN         DD  *
                OPTION  DEVICE=3800M3
                GRAPHIC
                NAME     AIBM
/*

```

The control statements are discussed as follows:

- The SYSUT1 DD statement includes DISP=OLD to ensure that no other job can modify the data set while this job is executing.
- The GRAPHIC statement's ASSIGN parameter specifies that the 8-bit data code for the user-designed character is X'5C' and the width is 10-pitch (the default when no pitch is specified). The GRAPHIC statement is followed by data statements that specify the character's scan pattern for vertical line spacing of 6 lines per inch.
- The name of the graphic character modification module is AIBM, and it is stored as a new module in SYS1.IMAGELIB.
- At STEP2, the INCLUDE statement specifies that a copy of the FM10 character arrangement table is to be used as a basis for the new module.
- The TABLE statement identifies the graphic character modification module named AIBM, created in the previous step. The TABLE statement's LOC parameter specifies the translation table entry location (the character's 8-bit data code) of X'5C' and the position (X'2C') where that character is to be loaded into the WCGM.
- The name of the new character arrangement table, which is added to SYS1.IMAGELIB, is BIBM.
- At STEP3, the three COPYMOD statements specify text that is to be placed on lines 58, 59, and 60 of the first copy of the output data set, starting at print position 5 on each line. When used with the BIBM character arrangement table, the characters W, 6, and X print as a top left corner, horizontal line segment, and top right corner, all in line weight 3. The characters 7, *, and 7 print as a weight-3 vertical line segment on both sides of the user-designed character built at STEP1 (the asterisk has the EBCDIC assignment 5C, which addresses that character). The hexadecimal E9, F6, and E8 complete the line-weight-3 Format box around the character.
- The name of the copy modification module is CIBM; it is stored as a new module on SYS1.IMAGELIB.
- At TEST, the EXEC statement calls for another execution of the IEBIMAGE program to test the modules just created. On the SYSPRINT DD statement the BIBM character arrangement table is the second of two specified, and the CIBM copy modification module is specified with a table reference character of 1, to use that BIBM table.
- The GRAPHIC statement with no operand specified calls for printing of the module, AIBM, specified with the NAME statement that follows it. Each page of the output listing for this IEBIMAGE run has a small image of the modification printed in the lower left corner.
- The OPTION statement with the DEVICE parameter at STEP1, STEP2, and STEP3 specifies 3800 Model 3 compatibility mode module format and processing considerations.

Example 19: List a Library Character Set Module

3800 Model 1

In this example, each segment of a library character set is printed. The scan pattern of each of the characters in the module is printed.

```

//LIBMOD1      JOB    ...
//STEP1        EXEC PGM=IEBIMAGE
//SYSUT1       DD  DSN=SYS1.IMAGELIB,DISP=SHR
//SYSPRINT     DD  SYSOUT=A

```

```
//SYSIN DD *
CHARSET
NAME 83
/*
```

The control statements are discussed as follows:

- NAME specifies the name of the library character set (83).

Example 20: Build a Library Character Set Module

3800 Model 3

In this example, a library character set module is built. Its characters are segments copied from the World Trade National Use Graphics graphic character modification module. For the listing of all the segments of that module, see *IBM 3800 Printing Subsystem Programmer's Guide*. The EBCDIC assignments for the characters are replaced by WCGM-location codes. The new module is stored in the SYS1.IMAGELIB system data set.

```
//LIBMOD2 JOB ... 72
//STEP1 EXEC PGM=IEBIMAGE
//SYSUT1 DD DSN=SYS1.IMAGELIB,DISP=OLD
//SYSPRINT DD SYSOUT=A
//SYSIN DD *
OPTION DEVICE=3800M3
CHARSET REF=((24,01),(25,02),(26,03),(27,04),(28,05), X
(31,06),(33,07),(35,08),(38,09),(40,0A))
NAME 73
/*
```

The control statements are discussed as follows:

- The SYSUT1 DD statement includes DISP=OLD to ensure that no other job can modify the data set while this job is executing.
- DEVICE=3800M3 in the OPTION statement specifies 3800 Model 3 compatibility mode module format.
- By not specifying the GCM keyword or a library character set ID, the CHARSET statement identifies the World Trade National Use Graphics graphic character modification module. Ten of its segments are to be copied and used with the new module. For example, the 24th segment is to be copied and assigned the WCGM location 01. See the REF parameter (24,01).
- The name of the library character set module is 73, and it is stored as a new module in the SYS1.IMAGELIB data set.

Example 21: Build a Library Character Set Module and Modify a Character Arrangement Table to Use It

3800 Model 3

In this example, a library character set module is built. The module contains one user-designed character, a reverse 'E', whose 6-bit WCGM-location code is designated as X'03', and whose pitch is 10. An existing character arrangement table is then modified to include the reverse E.

```
//LIBMOD3 JOB ...
//STEP1 EXEC PGM=IEBIMAGE
//SYSUT1 DD DSN=SYS1.IMAGELIB,DISP=OLD
//SYSPRINT DD SYSOUT=A
//SYSIN DD *
OPTION DEVICE=3800M3
CHARSET ASSIGN=(03,10)
XXXXXXXXXXXXXXXXX SEQ=10
XXXXXXXXXXXXXXXXX SEQ=11
XXXXXXXXXXXXXXXXX SEQ=12
XXXX SEQ=13
XXXX SEQ=14
XXXX SEQ=15
XXXX SEQ=16
XXXX SEQ=17
XXXX SEQ=18
```

```

          XXXX      SEQ=19
XXXXXXXXXXXX      SEQ=20
XXXXXXXXXXXX      SEQ=21
XXXXXXXXXXXX      SEQ=22
          XXXX      SEQ=23
          XXXX      SEQ=24
          XXXX      SEQ=25
          XXXX      SEQ=26
          XXXX      SEQ=27
          XXXX      SEQ=28
          XXXX      SEQ=29
XXXXXXXXXXXX      SEQ=30
XXXXXXXXXXXX      SEQ=31
XXXXXXXXXXXX      SEQ=32
NAME          73
INCLUDE      GS10
OPTION      DEVICE=3800M3
TABLE      CGMID=(83,73),LOC=(E0,03,1)
NAME          RE10
/*

```

The control statements are discussed as follows:

- The SYSUT1 DD statement includes DISP=OLD to ensure that no other job can modify the data set while this job is executing.
- DEVICE=3800M3 in the OPTION statement specifies 3800 Model 3 compatibility mode module format and processing considerations.
- The CHARSET statement's ASSIGN parameter establishes the 6-bit WCGM-location code, X'03', and the width, 10-pitch, for the user-designed character. The data statements that follow the CHARSET statement describe the character's scan pattern.
- The name of the library character set module is 73, and it is stored as a new module in the SYS1.IMAGELIB data set.
- The INCLUDE statement specifies that a copy of the GS10 character arrangement table is to be used as the basis for the new table.
- The TABLE statement specifies the addition of the library character set containing the reverse E to that copy of the GS10 table.

CGMID=(83,73) specifies the character set identifier X'83' for the Gothic-10 set (which is the set already used by the GS10 table) and specifies X'73' as a character set identifier to allow loading of the second WCGM with the library character set 73.

LOC=(E0,03,1) specifies that the reverse E, which has been assigned the WCGM location 03 in the second WCGM, is to be referenced by the EBCDIC code X'E0'.

- The new character arrangement table is named RE10; it is stored as a new module in the SYS1.IMAGELIB data set.

Example 22: Build a Library Character Set Module from Multiple Sources

3800 Model 3

In this example, a library character set module is created. Its contents come from three different sources: 62 segments are copied from an existing module with the INCLUDE statement; the CHARSET statement is used to select another segment to be copied; a second CHARSET statement is used to establish characteristics for a user-designed character. The new library character set module, when built, is added to the SYS1.IMAGELIB.

```

//LIBMOD4  JOB    ...
//STEP1    EXEC  PGM=IEBIMAGE
//SYSUT1   DD    DSN=SYS1.IMAGELIB,DISP=OLD
//SYSPRINT DD    SYSOUT=A
//SYSIN    DD    *
          INCLUDE 33,DELSEG=(3,4)
          CHARSET REF=(1,02),GCM=BODE,ASSIGN=03
          *****      SEQ=06
          *****      SEQ=07
          ****          SEQ=08
          ***           SEQ=09

```



```

***      ****      SEQ=10
***      ****      SEQ=11
***      ****      SEQ=12
***      ****      SEQ=13
***      ****      SEQ=14
***      ****      SEQ=15
***      ****      SEQ=16
***      ****      SEQ=17
***      ****      SEQ=18
***      ****      SEQ=19
NAME 53
/*

```

The control statements are discussed as follows:

- The SYSUT1 DD statement includes DISP=OLD to ensure that no other job can modify the data set while this job is executing.
- The INCLUDE statement specifies that a copy of the library character set module named 33 is to be included with the new module. All segments of 33, except the third and fourth segments (as a result of DELSEG=3,4), are to be copied into the new module and become the basis for the new module.
- The CHARSET statement specifies the module's third and fourth segments:
 REF=(1,02) and GCM=BODE specify that the third segment of the new module is to be obtained by copying the first segment from the graphic character modification module named BODE. The segment's 6-bit WCGM-location code is to be set so that its character is identified with the code X'02'.
 ASSIGN=03 specifies that the new module's fourth segment is a user-designed character whose 6-bit WCGM-location code is X'03' and whose width is 10-pitch (the default when no pitch value is specified). The CHARSET statement is followed by data statements that specify the character's scan pattern.
- The name of the library character set module is 53, it is stored as a new module in the SYS1.IMAGELIB data set.

Chapter 9. IEBISAM Program

The IEBISAM program is no longer distributed. Starting in z/OS V1R7, ISAM data sets can no longer be processed (created, opened, copied or dumped). ISAM data sets that are still in use must be converted to VSAM key-sequenced data sets.

Prior to z/OS V1R7, you could use access method services to allocate a VSAM key-sequenced data set and copy an ISAM data set into it. Access method services are also used to manipulate VSAM key-sequenced data sets. See [*z/OS DFSMS Using Data Sets*](#) and the REPRO command in [*z/OS DFSMS Access Method Services Commands*](#) for information on converting ISAM data sets to VSAM key-sequenced data sets.

Chapter 10. IEBPDSE (PDSE Validation) Program

You can use IEBPDSE to validate a PDSE data set and determine whether it is valid or corrupted.

Input and Output

IEBPDSE uses the following input:

- A PDSE data set, to be validated.

IEBPDSE produces the following output:

- A message data set that contains informational messages (for example if the data set was found to be corrupted), the results of the validation check, and error messages.

Related reading: For information about the IEBPDSE return codes, see [Appendix A, “Invoking Utility Programs from an Application Program,”](#) on page 315.

Control

IEBPDSE is controlled by job control statements. Utility control statements are not used.

Job Control Statements

Table 33 on page 201 shows the job control statements for IEBPDSE.

Table 33. Job control statements for IEBPDSE

Statement	Use
JOB	Starts the job.
EXEC	<p>Specifies the program name (PGM=IEBPDSE) or, if the job control statements reside in a procedure library, the procedure name. A PARM keyword may be specified.</p> <p>PARM [DUMP NODUMP,]</p> <p>DUMP specifies that a dump of the SMSPDSE(1) address space should be taken if the PDSE is broken. The dump will include all the index records that were successfully read and may help IBM to service the problem. NODUMP specifies that no dump should be taken when the PDSE is broken. The default is NODUMP.</p> <p>If the DUMP option is specified, the PDSE validation utility issues an ABEND in the PDSE address space, which results on an SVC dump.</p> <p>[FLUSH NOFLUSH,]</p> <p>FLUSH specifies that pages of the PDSE that have been previously read should not be used in the analysis. FLUSH is the default. NOFLUSH specifies that pages which have been previously read should be used in the analysis.</p> <p>[NOANALYSIS,]</p> <p>specifies that the PDSE should not be analyzed. This parameter is normally specified with PERFORMPENDINGDELETE.</p> <p>[PERFORMPENDINGDELETE]</p> <p>specifies to perform pending delete processing to remove any pending delete members which are no longer needed.</p>

Table 33. Job control statements for IEBPDSE (continued)

Statement	Use
SYSPRINT DD	Defines a sequential output message data set, which can be written on a system printer, a magnetic tape volume, or a direct access volume. If this statement is omitted, the output appears in the job log.
SYSLIB DD	Defines the PDSE that will be accessed by the PDSE utility when performing the operations specified on the control statements. The DSNNAME parameter is required.
SYSIN DD	If specified, the SYSIN data set must be empty.

SYSPRINT DD Statement

The block size for the SYSPRINT data set must be a multiple of 121. If not, the job step will end with a return code of 8.

IEBPDSE Examples

These examples show some of the uses of IEBPDSE.

Example 1: Validate one PDSE

In this example, one PDSE is validated.

```
//STEPCHK EXEC PGM=IEBPDSE
//SYSPRINT DD SYSOUT=A
//SYSLIB DD DSN=IBMUSER.SIMPLE.V2.PDSE,DISP=OLD
```

Example 2: Validate two PDSEs

This example validates two PDSEs.

```
//STEPCHK2 EXEC PGM=IEBPDSE
//SYSLIB DD DSN=IBMUSER.SIMPLE.V2.PDSE,DISP=OLD
// DD DSN=IBMUSER.SIMPLE.V3.PDSE,DISP=OLD
// DD DSN=SYS1.LINKLIB,DISP=SHR
```

Example 3: Validate a PDSE with the DUMP option

This example validates a PDSE and specifies an SVC dump.

```
//STEPLINK EXEC PGM=IEBPDSE,
// PARM='DUMP'
//SYSLIB DD DSN=SYS1.SIEALNKE,DISP=SHR
```

Example 4: Perform pending delete processing without analyzing the PDSE

This example performs pending delete processing without an analysis.

```
//STEPLINK EXEC PGM=IEBPDSE,
// PARM='PerformPendingDelete,NoAnalysis'
//SYSLIB DD DSN=IBMUSER.BUSY.PDSE,DISP=SHR
```

Chapter 11. IEBTPCH (Print-Punch) Program

You can use IEBTPCH to print or punch all, or selected portions, of a sequential or partitioned data set or PDSE. Data can also be "printed" or "punched" to disk or tape.

IEBTPCH can be used to print or punch:

- A sequential or partitioned data set or PDSE, in its entirety
- Selected members from a partitioned data set or PDSE
- Selected records from a sequential or partitioned data set or PDSE
- The directory of a partitioned data set or PDSE
- An edited version of a sequential or partitioned data set or PDSE
- A data set containing double-byte character set data

You can specify the format for the records that you are printing or punching, or you can use IEBTPCH's default formats. The default formats are:

- Each logical output record begins on a new printed line or punched card.
- Each printed line consists of groups of eight characters separated by two blanks. Up to 96 data characters can be included on a printed line. Each punched card contains up to 80 contiguous bytes of information.
- Characters that cannot be printed appear as blanks.
- When the input is blocked, each logical output record is delimited by "*" and each block is delimited by "***".
- Sixty lines per page will be printed.

If you specify your own format, using the RECORD utility control statement, make sure that your output record length does not exceed the capability of the output device.

IEBTPCH provides optional editing facilities and exits for routines that you want to use to process labels or manipulate input or output records.

Printing or Punching an Entire Data Set or Selected Member

You can use IEBTPCH to print or punch an entire sequential or partitioned data set or PDSE, or only selected members of a partitioned data set or PDSE. Members can be selected using the MEMBER utility control statement. When processing an entire partitioned data set or PDSE, members will be printed or punched in TTR order. When processing only selected members, the members will be processed in the order in which they are specified in the MEMBER control statements.

Data sets can be printed in hexadecimal if you choose. If you are printing a data set containing packed decimal data, the packed decimal data should be converted to unpacked decimal or hexadecimal mode to ensure that all characters are printable. Use the RECORD utility control statement to specify data conversion.

Printing or Punching an Edited Data Set

IEBTPCH can be used to print or punch an edited version of a sequential or a partitioned data set or PDSE. Utility control statements can be used to specify editing information that applies to a record, a group of records, selected groups of records, or an entire member or data set. You can print up to 144 characters per line.

An edited data set is produced by:

- Rearranging or omitting defined data fields within a record

- Converting data from packed decimal to unpacked decimal or from alphanumeric to hexadecimal representation

Printing or Punching Double-Byte Character Set Data

Using IEBTPCH, you can print or punch data sets that contain double-byte character set (DBCS) data. A double-byte character set is used to represent languages too complex for the standard single-byte character set. Japanese, for example, requires a double-byte character set. To indicate that DBCS data must be processed, code the DBCS=YES parameter on the PRINT or PUNCH statements.

Double-byte character set strings are identified by being enclosed in the shift-out (<) and shift-in (>) characters. When IEBTPCH sees the shift-out character, it understands that your data is now "shifting out" of a single-byte character set string, and when it sees the shift-in character, it understands that your data is now "shifting into" a single-byte character set string.

Each byte in a double-byte character must have a value between X'41' and X'FE' inclusive, or the DBCS character must be a DBCS space (X'4040'). You can use IEBTPCH to verify that your data conforms to this standard before it is printed or punched. If needed, IEBTPCH can also insert the shift-out/shift-in characters. This checking can be specified using the conversion variables in the FIELD parameter of the RECORD statement.

The default printing format for data sets with DBCS data is different from the default for single-byte character set data. The maximum number of DBCS characters will be printed for each output line, when you code PRINT DBCS=YES. IEBTPCH will ensure that an output record will not end within a DBCS string. If an entire DBCS string will not fit on one printed line, IEBTPCH will enclose each line in shift-out/shift-in characters. These control characters will not be printed, but will ensure that the printer recognizes the data as DBCS strings.

Printing or Punching Selected Records

IEBTPCH makes it possible for you to select only certain records from a data set for printing or punching. Utility control statements can be used to specify:

- The termination of a print or punch operation after a specified number of records has been printed or punched
- The printing or punching of every n^{th} record
- The starting of a print or punch operation after a specified number of records

Printing or Punching a Partitioned Directory

You can print or punch the contents of a partitioned directory using IEBTPCH. If the directory is printed in hexadecimal representation, the first four printed characters of each directory block indicate the total number of bytes used in that block. If the directory is punched, data from the directory block is punched in contiguous columns in the punched cards representing that block.

Although PDSE directories do not contain blocks, they will be printed or punched in the same format as the directory of a partitioned data set.

Related reading: For more information about the format of the directory, see [z/OS DFSMS Using Data Sets](#).

Printing or Punching to Disk or Tape

You can use IEBTPCH to "print" or "punch" a data set to disk or tape. If you do so, the first character of each record will be an ASA carriage control character, which is used to control the printing or punching operation. You can print or punch the data set at a later time using the PREFORM=A parameter of the PRINT and PUNCH statements.

If you punch a data set to disk or tape, you can, at your option, include sequence numbers in positions 73 through 80.

If the data set contains double-byte character set data, shift-out/shift-in characters (< and >) will be inserted at the beginning and end of a record if the DBCS string exceeds the logical record length of the output data set.

Input and Output

IEBTPCH uses the input as follows:

- An input data set, which contains the data that is printed or punched. The input data set can be either sequential or partitioned.
- A control data set, which contains utility control statements. The control data set is required for each use of IEBTPCH.

IEBTPCH produces the output as follows:

- An output data set, which is the printed or punched data set.
- A message data set, which contains informational messages (for example, the contents of the control statements) and any error messages.

If IEBTPCH is invoked from an application program, you can dynamically allocate the data sets by issuing SVC 99 before calling IEBTPCH.

Related reading: For information about IEBTPCH return codes, see [Appendix A, “Invoking Utility Programs from an Application Program,”](#) on page 315.

Control

IEBTPCH is controlled by job and utility control statements. The job control statements are required to process or load the IEBTPCH program and to define the data sets that are used and produced by the program. The utility control statements are used to control the functions of IEBTPCH.

Job Control Statements

Table 34 on page 205 shows the job control statements for IEBTPCH.

Table 34. Job control statements for IEBTPCH

Statement	Use
JOB	Starts the job step.
EXEC	Specifies the program name (PGM=IEBTPCH) or, if the job control statements reside in a procedure library, the procedure name.
SYSPRINT DD	Defines a sequential data set for messages. The data set can be written to a system output device, a tape volume, or a direct access device.
SYSUT1 DD	Defines an input sequential or partitioned data set, or PDSE.
SYSUT2 DD	Defines the output (print or punch) data set.
SYSIN DD	Defines the control data set. The control data set normally resides in the input stream; however, it can be defined as a member in a partitioned data set or PDSE.

SYSPRINT DD Statement

The SYSPRINT DD statement is required for each use of IEBTPCH. The record format is always FBA, the logical record length is always 121. Output can be blocked by specifying a block size that is a multiple of 121 on the SYSPRINT DD statement. The default block size is 121.

SYSUT1 DD Statement

The SYSUT1 DD statement is required for each use of IEBTPCH. The record format (except for undefined records), block size and logical record length (except for undefined and fixed unblocked records) must be present on the DD statement, in the DSCB, or on the tape label.

The input data set can contain fixed, variable, undefined, or variable spanned records. Variable spanned records are permitted only when the input is sequential. The block size cannot exceed 32760 bytes.

Directories of partitioned data sets or PDSEs are considered sequential data sets. Specify TYPORG=PS on the PRINT or PUNCH statement. You must specify RECFM=U, BLKSIZE=256, and LRECL=256 on the SYSUT1 DD statement.

SYSUT2 DD Statement

The SYSUT2 DD statement is required every time IEBTPCH is used. The record format is FBA or, if PREFORM=M is coded, FBM. The LRECL parameter, or, if no logical record length is specified, the BLKSIZE parameter, specifies the number of characters to be written per printed line or per punched card (this count must include the control character). The number of characters specified must be in the range of 2 through 145. The default values for edited output lines are 121 characters per printed line and 81 characters per punched card.

The SYSUT2 data set can be blocked by specifying both the LRECL and the BLKSIZE parameters, in which case, block size must be a multiple of logical record length. The block size cannot exceed 32760 bytes.

Both the output data set and the message data set can be written to the system output device if it is a printer.

If the logical record length of the input records is such that the output would exceed the output record length, IEBTPCH divides the record into multiple lines or cards in the case of standard printed output, standard punched output, or when the PREFORM parameter is specified. For nonstandard output, or if the PREFORM parameter is not specified, only part of the input record is printed or punched (maximums determined by the specific characteristics of your output device).

SYSIN DD Statement

The SYSIN DD statement is required for each use of IEBTPCH. The record format is always FB, the logical record length is always 80. Any blocking factor that is a multiple of 80 up to 32760 can be specified for the block size. The default block size is 80.

Utility Control Statements

IEBTPCH is controlled by utility control statements. The control statements in [Table 35 on page 206](#) are shown in the order in which they must appear.

Control statements are included in the control data set, as required. Any number of MEMBER and RECORD statements can be included in a job step.

A nonblank character in column 72 is optional for IEBTPCH continuation statements. Continuation requirements for utility control statements are described in [“Continuing utility control statements” on page 8](#).

Table 35. IEBTPCH utility control statements

Statement	Use
PRINT	Specifies that the data is printed.
PUNCH	Specifies that the data is punched.
TITLE	Specifies that a title is to precede the printed or punched data.
EXITS	Specifies that you are providing exit routines.

Table 35. IEBTPCH utility control statements (continued)

Statement	Use
MEMBER	Specifies which member of a partitioned data set or PDSE you want printed or punched.
RECORD	Specifies the format in which you want your data printed or punched.
LABELS	Specifies whether your labels are to be treated as data.

PRINT and PUNCH Statements

You use the PRINT and PUNCH statements to specify whether the data set is to be printed or punched. You must include one of these statements as the first statement of your control data set. You cannot use both statements at once, and you cannot use a statement more than once.

The syntax of the PRINT and PUNCH statements is:

Label	Statement	Parameters
[label]	{PRINT PUNCH}	[PREFORM={A M}] [, TYPORG={PS PO}] [, TOTCONV={XE PZ}] [, CNTRL={n 1}] [, STRTAFT=n] [, STOPAFT=n] [, SKIP=n] [, MAXNAME=n] [, MAXFLDS=n] [, MAXGPS=n] [, MAXLITS=n] [, DBCS={YES NO}] [, INITPG=n] [, MAXLINE=n] [, CDSEQ=n] [, CDINCR=n]
Notes: <ul style="list-style-type: none"> INITPG and MAXLINE can only be specified with PRINT CDSEQ and CDINCR can only be specified with PUNCH 		

where:

PREFORM={A|M}

specifies that a control character is provided as the first character of each record to be printed or punched. The control characters are used to control the spacing, number of lines per page, page ejection, and selecting a stacker. That is, the output has been previously formatted, and should be printed or punched according to that format. If an error occurs, the print/punch operation is stopped.

If PREFORM is coded, any additional PRINT or PUNCH operands other than TYPORG, and all other control statements except for LABELS, are ignored. Any ignored statement or operands are checked for correct syntax, however. PREFORM must not be used for printing or punching data sets with VS or VBS records longer than 32K bytes. These values are coded as follows:

A

specifies that an ASA control character is provided as the first character of each record to be printed or punched. If the input record length exceeds the output record length, the utility uses the ASA character for printing the first line, with a single space character on all subsequent lines of the record (for PRINT), or duplicates the ASA character on each output card of the record (for PUNCH).

If you are printing or punching a data set that was formatted using IEBTPCH, PREFORM=A should be coded.

M

specifies that a machine-code control character is provided as the first character of each record to be printed or punched. If the input record length exceeds the output record length, the utility prints all lines of the record with a *print-skip-one-line* character until the last line of the record, which will contain the actual character provided as input (for PRINT), or duplicates the machine control character on each output card of the record (for PUNCH).

TYPORG={PS|PO}

specifies the organization of the input data set. These values are coded as follows:

PS

specifies that the input data set is organized sequentially. This is the default.

PO

specifies that the input data set is partitioned.

TOTCONV={XE|PZ}

specifies the representation of data to be printed or punched. TOTCONV can be overridden by any user specifications (RECORD statements) that pertain to the same data. These values are coded as follows:

XE

specifies that data is punched in 2-character-per-byte hexadecimal representation (for example, C3 40 F4 F6). If XE is not specified, data is punched in 1-character per byte alphanumeric representation. The preceding example would appear as C 46.

The converted portion of the input record (length L) occupies 2L output characters.

PZ

specifies that data (packed decimal mode) is converted to unpacked decimal mode. IEBTPCH does not check for packed decimal mode.

The converted portion of the input record (length L) occupies 2L-1 output characters when punching, and 2L output characters

Default: If TOTCONV is omitted, data is not converted.

CNTRL={n|1}

For PRINT, CNTRL specifies an ASA control character for the output device that indicates line spacing, as follows: 1 indicates single spacing (the default), 2 indicates double spacing, and 3 indicates triple spacing.

For PUNCH, CNTRL specifies an ASA control character for the output device that is used to select the stacker, as follows: 1 indicates the first stacker (the default), 2 indicates the second stacker, and 3 indicates the third stacker, if any.

STRTAFT=n

specifies, for sequential data sets, the number of logical records (physical blocks in the case of variable spanned (VS) or variable block spanned (VBS) records longer than 32K bytes) to be skipped before printing or punching begins. For partitioned data sets or PDSEs, STRTAFT specifies the number of logical records to be skipped in each member before printing or punching begins. The *n* value must not exceed 32767. If STRTAFT is specified and RECORD statements are present, the first RECORD statement of a member describes the format of the first logical record to be printed or punched.

STOPAFT=n

specifies, for sequential data sets, the number of logical records (or physical blocks in the case of VS or VBS records longer than 32K bytes) to be printed or punched. For partitioned data sets or PDSEs, this specifies the number of logical records (or physical blocks in the case of VS or VBS records longer than 32K bytes) to be printed or punched in each member to be processed. The *n* value must not exceed 32767. If STOPAFT is specified and the IDENT parameter of the RECORD statement is also specified, the operation is stopped when the STOPAFT count is satisfied or at the end of the first record group, whichever occurs first.

SKIP=*n*

specifies that every n^{th} record (or physical block in the case of VS or VBS records longer than 32K bytes) is printed or punched.

Default: Successive logical records are printed or punched.

MAXNAME=*n*

specifies a number no less than the total number of member names and aliases appearing in subsequent MEMBER statements. The value must not exceed 32767.

If MAXNAME is omitted when there is a MEMBER statement present, the print or punch request is stopped.

MAXFLDS=*n*

specifies a number no less than the total number of FIELD parameters appearing in subsequent RECORD statements. The value must not exceed 32767.

If MAXFLDS is omitted when there is a FIELD parameter present, the print or punch request is stopped.

MAXGPS=*n*

specifies a number no less than the total number of IDENT parameters appearing in subsequent RECORD statements. The value must not exceed 32767.

If MAXGPS is omitted when there is an IDENT parameter present, the print or punch request is stopped.

MAXLITS=*n*

specifies a number no less than the total number of characters contained in the IDENT literals of subsequent RECORD statements. The value must not exceed 32767.

If MAXLITS is omitted when there is a literal present, the print or punch request is ended.

DBCS={YES|NO}

specifies whether the data set to be printed or punched contains double-byte character set data. NO is the default.

INITPG=*n*

specifies the initial page number; the pages are numbered sequentially thereafter. The INITPG value must not exceed 9999. The default is 1.

INITPG can only be coded with the PRINT statement.

MAXLINE=*n*

specifies the maximum number of lines to a printed page. Spaces, titles, and subtitles are included in this number. The default is 60 lines per page.

MAXLINE can only be coded with the PRINT statement.

CDSEQ=*n*

specifies the first sequence number of a deck of punched cards. This value must be contained in columns 73 through 80. Sequence numbering is initialized for each member of a partitioned data set. The default is that cards are not numbered.

CDSEQ can only be coded with the PUNCH statement.

CDINCR=*n*

specifies the increment to be used in generating sequence numbers. The default increment value is 10, unless CDSEQ is not coded, in which case the records are not numbered.

CDINCR can only be coded with the PUNCH statement.

TITLE Statement

Use the TITLE statement to specify any titles or subtitles you want printed or punched with your data set. Two TITLE statements can be included for each use of IEBTPCH. A first TITLE statement defines the title, and a second defines the subtitle. The TITLE statement, if included, follows the PRINT or PUNCH statement in the control data set.

If you are printing a data set, the titles you specify will be printed on every page.

The syntax of the TITLE statement is:

Label	Statement	Parameters
[<i>label</i>]	TITLE	ITEM=(' <i>title</i> ' [, <i>output-location</i>]) [, ITEM= . . .]

where:

ITEM=('title'[*output-location*])

specifies title or subtitle information. The values that can be coded are:

'title'

specifies the title or subtitle literal (maximum length of 40 bytes), enclosed in apostrophes. If the literal contains apostrophes, each apostrophe must be written as two consecutive apostrophes. The literal coded for 'title' is not affected by the TOTCONV parameter of the PRINT or PUNCH statements. You can specify a double-byte character set string as your title. To do so, enclose the DBCS string in shift-out/shift-in characters (< and >).

You can also specify the title in hexadecimal. To do so, code the title as *title* . This is especially useful if you do not have a keyboard that has all the characters you need. The shift-out/shift-in characters are X'0E' and X'0F', respectively.

output-location

specifies the starting position at which the literal for this item is placed in the output record. When used with *output-location*, the specified title's length plus *output-location* may not exceed the output logical record length minus one.

Default: The first position (byte) is assumed.

You can specify ITEM more than once on a TITLE statement. In this way, you can have titles longer than 40 characters, or you can format your title according to your needs.

EXITS Statement

The EXITS statement is used to identify exit routines that you want IEBTPCH to use for label or record processing. Exits to label processing routines are ignored if the input data set is partitioned. Linkage to and from user routines are discussed in [Appendix C, “Specifying User Exits with Utility Programs,”](#) on page 343.

The EXITS statement, if used, must immediately follow any TITLE statement or follow the PRINT or PUNCH statement.

The syntax of the EXITS statement is:

Label	Statement	Parameters
[<i>label</i>]	EXITS	[INHDR= <i>routinename</i>] [, INTLR= <i>routinename</i>] [, INREC= <i>routinename</i>] [, OUTREC= <i>routinename</i>]

where:

INHDR=routinename

specifies the name of the routine that processes user input header labels.

INTLR=routinename

specifies the name of the routine that processes user input trailer labels.

INREC=routinename

specifies the name of the routine that manipulates each logical record (or physical block in the case of VS or VBS records longer than 32K bytes) before it is processed.

OUTREC=routinename

specifies the name of the routine that manipulates each logical record (or physical block in the case of VS or VBS records longer than 32K bytes) before it is printed or punched.

MEMBER Statement

You use the MEMBER statement to identify members of partitioned data sets or PDSEs that you want printed or punched. All RECORD statements that follow a MEMBER statement pertain to the member indicated in that MEMBER statement only. When RECORD and MEMBER statements are used, at least one MEMBER statement must precede the first RECORD statement. If no RECORD statement is used, the member is processed according to the default format.

If no MEMBER statement appears, and a partitioned data set or PDSE is being processed, all members of the data set are printed or punched. Any number of MEMBER statements can be included in a job step.

If a MEMBER statement is present in the input stream, MAXNAME must be specified in a PRINT or PUNCH statement.

The syntax of the MEMBER statement is:

Label	Statement	Parameters
[<i>label</i> !]	MEMBER	NAME={ <i>membername</i> <i>aliasname</i> }

where:

NAME={*membername*|*aliasname*}

specifies a member of a partitioned data set or PDSE to be printed or punched. The values that can be coded are:

membername

specifies a member by its member name.

aliasname

specifies a member by its alias name.

If a MEMBER statement is present in the input stream, MAXNAME must be specified in a PRINT or PUNCH statement.

RECORD Statement

The RECORD statement is used to define a group of records, called a *record group*, that is printed or punched to your specifications. A record group consists of any number of records to be edited identically.

If no RECORD statements appear, the entire data set, or named member, is printed or punched in the default format. The default format has the following characteristics:

- Each printed line contains groups (8 characters each) of hexadecimal information.
- Each input record begins a new line of printed output.
- The size of the input record and the carriage width determine how many lines of printed output are required per input record.

If a RECORD statement is used, all data following the record group it defines (within a partitioned member or within an entire sequential data set) must be defined with other RECORD statements. Any number of RECORD statements can be included in a job step.

A RECORD statement referring to a partitioned data set or PDSE for which no members have been named need contain only FIELD parameters. These are applied to the records in all members of the data set.

If a FIELD parameter is included in the RECORD statement, MAXFLDS must be specified in the PRINT or PUNCH statement.

If an IDENT parameter is included in the RECORD statement, MAXGPS and MAXLITS must be specified in the PRINT or PUNCH statement.

The syntax of the RECORD statement is:

Label	Statement	Parameters
[<i>label</i>]	RECORD	[IDENT=(<i>length</i> , ' <i>name</i> ', <i>input-location</i>)] [, FIELD=(<i>length</i> , [<i>input-location</i>] , [<i>conversion</i>], [<i>output-location</i>])] [, FIELD=...]

where:

IDENT=(*length*, '*name*', *input-location*)

identifies the last record of the record group to which the FIELD parameters apply. The values that can be coded are:

length

specifies the length (in bytes) of the field that contains the identifying name in the input records. The length cannot exceed 8 bytes.

'*name*'

specifies the exact literal, enclosed in apostrophes, that identifies the last record of a record group. If the literal contains apostrophes, each must be written as two consecutive apostrophes.

You can specify '*name*' in hexadecimal by coding *name* . You can also specify a DBCS string for '*name*', either in DBCS characters or in the hexadecimal representation of DBCS characters. If you use hexadecimal for a DBCS string, the hexadecimal values of the shift-out/shift-in characters are X'0E' and X'0F', respectively.

input-location

specifies the starting location of the field that contains the identifying name in the input records.

The sum of the length and the input location must be equal to or less than the input logical record length plus one.

Default: If IDENT is omitted and STOPAFT is not included with the PRINT or PUNCH statement, record processing stops after the last record in the data set. If IDENT is omitted and STOPAFT is included with the PRINT or PUNCH statement, record processing halts when the STOPAFT count is satisfied or after the last record of the data set is processed, whichever occurs first.

If an IDENT parameter is included in the RECORD statement, MAXGPS and MAXLITS must be specified in the PRINT or PUNCH statement.

FIELD=(*length*, [*input-location*], [*conversion*], [*output-location*])

specifies field-processing and editing information.

Note that the variables on the FIELD parameter are positional; that is, if any of the options are not coded, the associated comma preceding that variable must be coded.

These values can be coded:

length

specifies the length (in bytes) of the input field to be processed. The length must be equal to or less than the first input logical record length.

input-location

specifies the starting position of the input field to be processed. The sum of the length and the input location must be equal to or less than the input logical record length plus one.

Default: The first position (byte) is assumed.

conversion

specifies the type of conversion to be performed on this field before it is printed or punched. The values that can be coded are:

CV

specifies that double-byte character set characters are combined with single-byte character set characters, and that the DBCS characters should be checked to ensure that they are printable. No shift-out/shift-in characters will be inserted to enclose DBCS strings.

DBCS=YES must be specified on the PRINT or PUNCH statement.

PZ

specifies that packed decimal data is to be converted to unpacked decimal data. The converted portion of the input record (length L) occupies 2L - 1 output characters when punching, and 2L output characters when printing.

VC

specifies that double-byte character set characters should be checked to ensure that they are printable, and that shift-out/shift-in characters (< and >) are to be inserted to enclose the DBCS strings.

DBCS=YES must be specified on the PRINT or PUNCH statement.

XE

specifies that alphanumeric data is to be converted to hexadecimal data. The converted portion of the input record (length L) occupies 2L output characters.

Default: The field is moved to the output area without change.

output-location

specifies the starting location of this field in the output records. Unspecified fields in the output records appear as blanks in the printed or punched output. Data that exceeds the SYSUT2 printer or punch size is not printed or punched. When specifying one or more FIELDS, the sum of all lengths and all extra characters needed for conversions must be equal to or less than the output LRECL minus one.

Default: The first position (byte) is assumed.

If a FIELD parameter is included in the RECORD statement, MAXFLDS must be specified in the PRINT or PUNCH statement.

LABELS Statement

You use the LABELS statement to specify whether you want your data set labels treated as data. For a detailed discussion of this option, refer to [“Processing User Labels” on page 347](#).

You must specify LABELS DATA=NO to make standard user label (SUL) exits inactive when you are processing an input data set with nonstandard labels (NSL).

If more than one valid LABELS statement is included, all but the last LABELS statement are ignored.

The syntax of the LABELS statement is:

Label	Statement	Parameters
[label]	LABELS	[CONV={PZ XE}] [, DATA={YES NO ALL ONLY}]

where:

CONV={PZ|XE}

specifies the type of conversion to be performed on this field before it is printed or punched. The values that can be coded are:

PZ

specifies that data (packed decimal) is converted to unpacked decimal data. The converted portion of the input record (length L) occupies 2L - 1 output characters when punching, and 2L output characters when printing.

XE

specifies that alphanumeric data is to be converted to hexadecimal data. The converted portion of the input record (length L) occupies 2L output characters.

Default: The field is moved to the output area without change.

DATA={YES|NO|ALL|ONLY}

specifies whether user labels are to be treated as data. The values that can be coded are:

YES

specifies that any user labels are to be treated as data unless they have been rejected by a label processing routine you have specified on the EXITS statement. Processing of labels as data stops in compliance with standard return codes. YES is the default.

NO

specifies that user labels are not to be treated as data. NO must be specified when processing input/output data sets with nonstandard labels (NSL) in order to make standard user label (SUL) exits inactive.

ALL

specifies that all user labels are to be treated as data. A return code of 16 causes the utility to complete the processing of the remainder of the group of user labels and to stop the job step.

ONLY

specifies that only user header labels are to be treated as data. User header labels are processed as data regardless of any return code. The job ends upon return from the OPEN routine.

IEBTPCH Examples

These examples illustrate some of the uses of IEBTPCH. You can use [Table 36 on page 214](#) as a quick-reference guide to IEBTPCH examples.

Table 36. IEBTPCH example directory

Operation	Data Set Organization	Devices	Comments	Example
PRINT	Partitioned	Disk and System Printer	Default format. Conversion to hexadecimal. Ten records from each member are printed.	“Example 8: Print Directory of a Partitioned Data Set” on page 218
PRINT	Partitioned	Disk and System Printer	Default format. Conversion to hexadecimal. Two members are printed.	“Example 9: Print Selected Records of a Partitioned Data Set” on page 219
PRINT	Partitioned	Disk and System Printer	Default format. DBCS data is checked and printed.	“Example 10: Convert to Hexadecimal and Print Partitioned Data” on page 219
PRINT	Sequential	System Printer	Conversion to hexadecimal.	“Example 3: Duplicate a Card Deck” on page 216

Table 36. IEBTPCH example directory (continued)

Operation	Data Set Organization	Devices	Comments	Example
PRINT	Sequential	Tape and System Printer	Default format.	“Example 4: Print Sequential Data Set According to Default Format” on page 216
PRINT	Sequential	Disk and System Printer	User-specified format. User routines are provided. Processing ends after the third record group is printed or STOPAFT is satisfied.	“Example 5: Print Sequential Data Set According to User Specifications” on page 216
PRINT	Sequential	System Printer	Print with user exit routine.	“Example 6: Print Three Record Groups” on page 217
PRINT	Sequential, Partitioned	Disk and System Printer	SYSOUT format. Conversion to hexadecimal.	“Example 7: Print a Pre-Formatted Data Set” on page 218
PUNCH	Sequential	Disk and Card Punch	User-specified format. Sequence numbers are assigned and punched.	“Example 1: Print Partitioned Data Set” on page 215
PUNCH	Sequential	Card Reader and Card Punch	User-specified format. A copy of a set of cards is made.	“Example 2: Punch Sequential Data Sets” on page 215

Examples that use **disk** or **tape** in place of actual device numbers must be changed before use. The actual device numbers depend on how your installation has defined the devices to your system.

Example 1: Print Partitioned Data Set

In this example, a member of partitioned data set is printed according to user specifications.

```
PRINT TYPORG=PO,MAXNAME=1,MAXFLDS=1
MEMBER NAME=UTILUPD8
RECORD FIELD=(80)
```

If the member card entry is not used, the entire data set will be printed.

Example 2: Punch Sequential Data Sets

In this example, a sequential data set is punched according to user specifications.

```
//PHSEQNO JOB ...
//STEP1 EXEC PGM=IEBTPCH
//SYSPRINT DD SYSOUT=A
//SYSUT1 DD DSN=MASTER.SEQSET,LABEL=(,SUL),DISP=SHR
//SYSUT2 DD SYSOUT=B
//SYSIN DD *
PUNCH MAXFLDS=1,CDSEQ=0,CDINCR=20
RECORD FIELD=(72)
LABELS DATA=YES
/*
```

The control statements are as follows:

- SYSUT1 DD defines the input data set, MASTER.SEQSET, which resides on a disk or tape volume. The data set contains 80-byte, fixed blocked records.
- SYSUT2 DD defines the system output class (punch is assumed). That portion of each record from the input data set defined by the FIELD parameter is represented by one punched card.
- SYSIN DD defines the control data set, which follows in the input stream.
- PUNCH begins the punch operation, indicates that one FIELD parameter is included in a subsequent RECORD statement (MAXFLDS=1), and assigns a sequence number for the first punched card

(00000000) and an increment value for successive sequence numbers (20). Sequence numbers are placed in columns 73 through 80 of the output records.

- RECORD indicates that positions 1 through 72 of the input records are to be punched. Bytes 73 through 80 of the input records are replaced by the new sequence numbers in the output card deck.
- LABELS specifies that user header labels and user trailer labels are punched.

Labels cannot be edited; they are always moved to the first 80 bytes of the output buffer. No sequence numbers are present on the cards containing user header and user trailer records.

Example 3: Duplicate a Card Deck

In this example, a card deck containing valid punch card code or BCD is duplicated.

```
//PUNCH      JOB      ...
//STEP1      EXEC     PGM=IEBTPCH
//SYSPRINT   DD      SYSOUT=A
//SYSIN      DD      DSN=PDSLIB(PNCHSTMT),DISP=(OLD,KEEP)
//SYSUT2     DD      SYSOUT=B
//SYSUT1     DD      DATA

(input card data set including // cards)
/*
```

The control statements are as follows:

- SYSIN DD defines the control data set PDSLIB which contains the member PNCHSTMT. (The data set is cataloged.) The record format must be FB and the logical record length must be 80.
- SYSUT2 DD defines the system output class (punch is assumed).
- SYSUT1 DD defines the input card data set, which follows in the input stream.

Example 4: Print Sequential Data Set According to Default Format

In this example, a sequential data set is printed according to the default format. The printed output is converted to hexadecimal.

```
//PRINT      JOB      ...
//STEP1      EXEC     PGM=IEBTPCH
//SYSPRINT   DD      SYSOUT=A
//SYSUT1     DD      DSN=INSET,UNIT=tape,
//              LABEL=(,NL),VOLUME=SER=001234,
//              DISP=(OLD,KEEP),DCB=(RECFM=U,BLKSIZE=2000)
//SYSUT2     DD      SYSOUT=A
//SYSIN      DD      *
              PRINT    TOTCONV=XE
              TITLE    ITEM=('PRINT SEQ DATA SET WITH CONV TO HEX',10)
/*
```

The control statements are as follows:

- SYSUT1 DD defines the input data set on a tape volume. The data set contains undefined records; no record is larger than 2,000 bytes.
- SYSUT2 DD defines the output data set. The data set is written to the system output device (printer assumed).
- SYSIN DD defines the control data set, which follows in the input stream.
- PRINT begins the print operation and specifies conversion from alphanumeric to hexadecimal representation.
- TITLE specifies a title to be placed beginning in column 10 of the printed output. The title is not converted to hexadecimal.

Example 5: Print Sequential Data Set According to User Specifications

In this example, a sequential data set is printed according to user specifications.

```

//PTNONSTD JOB ...
//STEP1 EXEC PGM=IEBTPCH
//SYSPRINT DD SYSOUT=A
//SYSUT1 DD DSN=SEQSET,UNIT=tape,LABEL=(2,SUL),
// DISP=(OLD,KEEP),VOLUME=SER=001234
//SYSUT2 DD SYSOUT=A
//SYSIN DD *
PRINT MAXFLDS=1
EXITS INHDR=HDRIN,INTLR=TRLIN
RECORD FIELD=(80)
LABELS DATA=YES
/*

```

The control statements are as follows:

- SYSUT1 DD defines the input data set, SEQSET, which is the second data set on a tape volume.
- SYSUT2 DD defines the output data set on the system output device (printer assumed). Each printed line contains 80 contiguous characters (one record) of information.
- SYSIN DD defines the control data set, which follows in the input stream.
- PRINT begins the print operation and indicates that one FIELD parameter is included in a subsequent RECORD statement (MAXFLDS=1).
- EXITS indicates that exits will be taken to user header label and trailer label processing routines when these labels are encountered on the SYSUT1 data set.
- RECORD indicates that each input record is processed in its entirety (80 bytes). Each input record is printed in columns 1 through 80 on the printer.
- LABELS specifies that user header and trailer labels are printed according to the return code issued by the user exits.

Example 6: Print Three Record Groups

In this example, three record groups are printed. A user routine is provided to manipulate output records before they are printed.

```

//PRINT JOB ...
//STEP1 EXEC PGM=IEBTPCH
//SYSPRINT DD SYSOUT=A
//SYSUT1 DD DSN=SEQDS,UNIT=disk,DISP=(OLD,KEEP),
// LABEL=(,SUL),VOLUME=SER=111112
//SYSUT2 DD SYSOUT=A
//SYSIN DD *
PRINT MAXFLDS=9,MAXGPS=9,MAXLITS=23,STOPAFT=32767
TITLE ITEM=('TIMECONV-DEPT D06'),
ITEM=('JAN10-17',80)
EXITS OUTREC=NEWTIME,INHDR=HDRS,INTLR=TLRS
RECORD IDENT=(6,'498414',1),
FIELD=(8,1,,10),FIELD=(30,9,XE,20)
RECORD IDENT=(2,'**',39),
FIELD=(8,1,,10),FIELD=(30,9,XE,20)
RECORD IDENT=(6,'498414',1),
FIELD=(8,1,,10),FIELD=(30,9,XE,20)
LABELS CONV=XE,DATA=ALL
/*

```

The control statements are as follows:

- SYSUT1 DD defines the input data set, called SEQDS. The data set resides on a disk volume.
- SYSUT2 DD defines the output data set on the system output device (printer assumed).
- SYSIN DD defines the control data set, which follows in the input stream.
- The PRINT statement:
 1. Initializes the print operation.
 2. Indicates that not more than nine FIELD parameters are included in subsequent RECORD statements (MAXFLDS=9).

3. Indicates that not more than nine IDENT parameters are included in subsequent RECORD statements (MAXGPS=9).
 4. Indicates that not more than 23 literal characters are included in subsequent IDENT parameters (MAXLITS=23).
 5. Indicates that processing is ended after 32767 records are processed or after the third record group is processed, whichever comes first. Because MAXLINE is omitted, 60 lines are printed on each page.
- TITLE specifies two titles, to be printed on one line. The titles are not converted to hexadecimal.
 - EXITS specifies the name of a user routine (NEWTIME), which is used to manipulate output records before they are printed.
 - The first RECORD statement defines the first record group to be processed and indicates where information from the input records is placed in the output records. Positions 1 through 8 of the input records appear in positions 10 through 17 of the printed output, and positions 9 through 38 are printed in hexadecimal representation and placed in positions 20 through 79.
 - The second RECORD statement defines the second group to be processed. The parameter in the IDENT operand specifies that an input record last record edited according to the FIELD operand in this RECORD statement. The FIELD operand specifies that positions 1 through 8 of the input records are placed in positions 10 through 17 of the printed output, and positions 9 through 38 are printed in hexadecimal representation and appear in positions 20 through 79.
 - The third and last RECORD statement is equal to the first RECORD statement. An input record that meets the parameter in the IDENT operand ends processing, unless the STOPAFT parameter in the PRINT statement has not already done so.
 - LABELS specifies that all user header or trailer labels are to be printed regardless of any return code, except 16, issued by the user's exit routine. It also indicates that the labels are converted from alphanumeric to hexadecimal representation (CONV=XE).

Example 7: Print a Pre-Formatted Data Set

In this example, the input is a SYSOUT (sequential) data set, which was previously written as the second data set of a standard label tape. It is printed in SYSOUT format.

```
//PTSYSOUT JOB ...
//STEP1 EXEC PGM=IEBTPCH
//SYSPRINT DD SYSOUT=A
//SYSUT1 DD UNIT=tape, LABEL=(2,SL), DSNAME=LISTING,
// DISP=(OLD,KEEP), VOL=SER=001234
//SYSUT2 DD SYSOUT=A
//SYSIN DD *
PRINT PREFORM=A
/*
```

The control statements are as follows:

- SYSUT1 DD defines the input data set, which was previously written as the second data set of a standard label tape. The data set has been assigned the name LISTING.
- SYSUT2 DD defines the output data set on the system output device (printer assumed).
- SYSIN DD defines the control data set, which follows in the input stream.
- The PRINT statement begins the print operation and indicates that an ASA control character is provided as the first character of each record to be printed (PREFORM=A).

Example 8: Print Directory of a Partitioned Data Set

In this example, the directory of a partitioned data set is printed according to the default format. The printed output is converted to hexadecimal.

```
//PRINTDIR JOB ...
//STEP1 EXEC PGM=IEBTPCH
//SYSPRINT DD SYSOUT=A
//SYSUT1 DD DSNAME=MAIN.PDS,
```

```
//          DISP=(OLD,KEEP),DCB=(RECFM=U,BLKSIZE=256)
//SYSUT2   DD  SYSOUT=A
//SYSIN    DD  *
PRINT     TYPORG=PS,TOTCONV=XE
TITLE     ITEM=('PRINT PARTITIONED DIRECTORY OF PDS',10)
TITLE     ITEM=('FIRST TWO BYTES SHOW NUM OF USED BYTES',10)
LABELS    DATA=NO
/*
```

The control statements are as follows:

- SYSUT1 DD defines the input data set (the partitioned directory), which resides on a disk volume. The DCB keywords describe the directory, not the member contents.
- SYSUT2 DD defines the output data set on the system output device (printer assumed).
- SYSIN DD defines the control data set, which follows in the input stream.
- PRINT begins the print operation, indicates that the partitioned directory is organized sequentially, and specifies conversion from alphanumeric to hexadecimal representation.
- The first TITLE statement specifies a title, and the second TITLE statement specifies a subtitle. Neither title is converted to hexadecimal.
- LABELS specifies that no user labels are printed.

Note: Not all of the bytes in a directory block need to contain data pertaining to the partitioned data set. Unused bytes are sometimes used by the operating system as temporary work areas. With conversion to hexadecimal representation, the first four characters of printed output indicate how many bytes of the 256-byte block pertain to the partitioned data set. Any unused bytes occur in the latter portion of the directory block. They are not interspersed with the used bytes.

Example 9: Print Selected Records of a Partitioned Data Set

In this example, a partitioned data set (ten records from each member) is printed according to the default format. The printed output is converted to hexadecimal.

```
//PRINTPDS JOB  ...
//STEP1    EXEC PGM=IEBTPCH
//SYSPRINT DD  SYSOUT=A
//SYSUT1   DD  DSNAME=WAYNE.TEST.DATA,DISP=SHR
//SYSUT2   DD  SYSOUT=A
//SYSIN    DD  *
PRINT     TOTCONV=XE,TYPORG=PO,STOFAFT=10
TITLE     ITEM=('PRINT PDS - 10 RECS EACH MEM',20)
/*
```

The control statements are as follows:

- SYSUT1 DD defines the input data set, called WAYNE.TEST.DATA, on a disk volume.
- SYSUT2 DD defines the output data set on the system output device (printer assumed).
- SYSIN DD defines the control data set, which follows in the input stream.
- PRINT begins the print operation, specifies conversion from alphanumeric to hexadecimal representation, indicates that the input data set is partitioned, and specifies that 10 records from each member are to be printed.
- TITLE specifies a title to be placed beginning in column 20 of the printed output. The title is not converted to hexadecimal.

Example 10: Convert to Hexadecimal and Print Partitioned Data

In this example, two partitioned members are printed according to the default format. The printed output is converted to hexadecimal.

```
//PRNTMEMS JOB  ...
//STEP1    EXEC PGM=IEBTPCH
//SYSPRINT DD  SYSOUT=A
//SYSUT1   DD  DSNAME=BROWN.MAIN.LIB,DISP=SHR
//SYSUT2   DD  SYSOUT=A
```

```
//SYSIN DD *
PRINT TYPORG=PO,TOTCONV=XE,MAXNAME=2
TITLE ITEM=('PRINT TWO MEMBS WITH CONV TO HEX',10)
MEMBER NAME=MEMBER1
MEMBER NAME=MEMBER2
/*
```

The control statements are as follows:

- SYSUT1 DD defines the input data set, called BROWN.MAIN.LIB, on a disk volume.
- SYSUT2 DD defines the output data set on the system output device (printer assumed).
- SYSIN DD defines the control data set, which follows in the input stream.
- PRINT begins the print operation, indicates that the input data set is partitioned, specifies conversion from alphanumeric to hexadecimal representation, and indicates that two MEMBER statements appear in the control data set (MAXNAME=2).
- TITLE specifies a title to be placed beginning in column 10 of the printed output. The title is not converted to hexadecimal.
- MEMBER specifies the member names of the members to be printed (MEMBER1 and MEMBER2).

Example 11: Print Member Containing DBCS Data

In this example, a member of a partitioned data set that contains DBCS data is printed after the DBCS data is checked to ensure that all DBCS characters are printable.

```
//DBCS JOB ...
//STEP1 EXEC PGM=IEBTPCH
//SYSPRINT DD SYSOUT=A
//SYSUT1 DD DSN=DSNAME,PDS,UNIT=disk,DISP=(OLD,KEEP)
//SYSUT2 DD SYSOUT=A
//SYSIN DD *
PRINT TYPORG=PO,DBCS=YES,MAXFLDS=1,MAXNAME=1
MEMBER NAME=MEM1
RECORD FIELD=(,CV)
/*
```

The control statements are as follows:

- SYSUT1 DD defines the input data set, PDS, on a disk volume.
- SYSUT2 DD defines the system printer as the output data set.
- SYSIN DD defines the control data set, which follows in the input stream.
- PRINT begins the print operation, indicates that the input data set is partitioned, and indicates that double-byte character set data will be printed. The statement also indicates that one MEMBER statement appears in the control data set, and that one FIELD parameter appears on a subsequent RECORD statement.
- MEMBER specifies the member, MEM1, that is to be printed.
- RECORD specifies that the DBCS data is to be checked to ensure that it is printable.

Chapter 12. IEBUPDTE (Update Data Set) Program

You can use IEBUPDTE to create or modify sequential or partitioned data sets or PDSEs. However, the program can be used only with data sets containing fixed-length records of no more than 80 bytes. (It is used primarily for updating procedure, source, and macro libraries, such as those containing JCL.)

You can use IEBUPDTE to perform these tasks:

- Incorporate IBM or your source language modifications into sequential or partitioned data sets, or PDSEs.
- Create and update data set libraries.
- Modify existing sequential data sets or members of partitioned data sets or PDSEs.
- Change the organization of a data set from sequential to partitioned or PDSE, or the reverse.

You can also use your own exit routines to process header and trailer labels.

Creating and Updating Data Set Libraries

IEBUPDTE can be used to create a library of partitioned members, if those members have logical record lengths of 80 or less. In addition, members can be added directly to an existing library, provided that the original space allocations are sufficient to incorporate the new members. In this manner, a cataloged procedure can be placed in a procedure library, or a set of job or utility control statements can be placed as a member in a partitioned library. These libraries may be either partitioned data sets or PDSEs.

Modifying an Existing Data Set

IEBUPDTE can be used to modify an existing partitioned or sequential data set, or PDSE. Logical records can be replaced, deleted, renumbered, or added to the member or data set.

A sequential data set residing on a tape volume can be used to create a new master (that is, a modified copy) of the data set. A sequential data set residing on a direct access device can be modified either by creating a new master or by modifying the data set directly on the volume on which it resides.

A partitioned data set or PDSE can be modified either by creating a new master or by modifying the data set directly on the volume on which it resides.

Changing Data Set Organization

IEBUPDTE can be used to change the organization of a data set from sequential to partitioned or PDSE, or to change a single member of a partitioned data set or PDSE to a sequential data set. If only a member is changed, the remainder of the original data set remains unchanged. In addition, logical records can be replaced, deleted, renumbered, or added to the member or data set.

Input and Output

IEBUPDTE uses the input as follows:

- An input data set (also called the old master data set), which is modified or used as source data for a new master. The input data set is either a sequential data set or a member of a partitioned data set or PDSE.
- A control data set, which contains utility control statements and, if applicable, input data. The data set is required for each use of IEBUPDTE.

IEBUPDTE produces the output as follows:

- An output data set, which is the result of the IEBUPDTE operation. The data set can be sequential, partitioned, or PDSE. It can be either a new data set (that is, created during the present job step) or an existing data set, modified during the present job step.
- A message data set, which contains the utility program identification, control statements used in the job step, modification made to the input data set, and diagnostic messages, if applicable. The message data set is sequential.

Related reading: For IEBUPDTE return codes, see [Appendix A, “Invoking Utility Programs from an Application Program,”](#) on page 315.

Control

IEBUPDTE is controlled by job and utility control statements. The job control statements are required to process or load IEBUPDTE and to define the data sets that are used and produced by the program. The utility control statements are used to control the functions of IEBUPDTE and, in certain cases, to supply new or replacement data.

Job Control Statements

Table 37 on page 222 shows the job control statements for IEBUPDTE.

Table 37. Job control statements for IEBUPDTE

Statement	Use
JOB	Starts the job.
EXEC	Specifies the program name (PGM=IEBUPDTE), or, if the job control statements reside in a procedure library, the procedure name. Additional information can be specified in the PARM parameter of the EXEC statement.
SYSPRINT DD	Defines a sequential data set for messages. The data set can be written to a system output device, a tape volume, or a direct access volume.
SYSUT1 DD	Defines the input (old master) data set. It can define a sequential data set on a card reader, a tape volume, or a direct access volume. It can define a partitioned data set or PDSE on a direct access volume.
SYSUT2 DD	Defines the output data set. It can define a sequential data set on a card punch, a printer, a tape volume, or a direct access device. It can define a partitioned data set or PDSE on a direct access volume.
SYSIN DD	Defines the control data set. The control data set normally resides in the input stream; however, it can be defined as a member of a partitioned data set or PDSE.

EXEC Statement

Additional information can be coded in the PARM parameter of the EXEC statement, as follows:

Label	Statement	Parameters
// [stepname]	EXEC	PGM=IEBUPDTE [, PARM= ' {NEW MOD} [, inhdr] [, intlr] ']

where:

PGM=IEBUPDTE

specifies that IEBUPDTE is the program you want to run.

PARM= ' {NEW|MOD} [, inhdr] [, intlr] '

specifies optional information as follows:

NEW

specifies that the input consists solely of the control data set. Do not code a SYSUT1 DD statement if you specify NEW.

MOD

specifies that the input consists of both the control data set and the input data set. MOD is the default.

inhdr

specifies the name of the routine that processes the user header labels for the control data set.

intlr

specifies the name of the routine that processes the user trailer labels for the control data set.

If only one value is coded in PARM, the single quotes (or parentheses) are optional.

SYSPRINT DD Statement

The message data set has a logical record length of 121 bytes, and consists of fixed length, blocked or unblocked records with an American National Standards Institute (ANSI) control character in the first byte of each record.

SYSUT1 and SYSUT2 DD Statements

If the SYSUT1 and SYSUT2 DD statements define the same sequential data set, only those operations that add data to the end of the existing data set can be made. In these cases:

- The PARM parameter of the EXEC statement must imply or specify MOD.
- The DISP parameter of the SYSUT1 DD statement must specify OLD.

If SYSUT1 and SYSUT2 define the same partitioned data set or PDSE, new extents resulting from updates on SYSUT2 are not retrievable in SYSUT1.

The input and output data sets contain blocked or unblocked logical records with record lengths of up to 80 bytes. The input and output data sets may have different block sizes as long as they are multiples of the logical record length. However, if insufficient space is allocated for reblocked records, the update request is ended.

If an ADD operation is specified with PARM=NEW in the EXEC statement, the SYSUT1 DD statement need not be coded.

If the SYSUT1 DD statement defines a sequential data set on tape, the file sequence number of that data set must be included in the LABEL keyword (unless the data set is the first or only data set on the volume).

If the output data set (SYSUT2) does not already exist and is to reside on a direct access device, space must be allocated for it. The SYSUT2 DD statement must not specify a DUMMY data set.

When adding a member to an existing partitioned data set or PDSE using an ADD function statement, any DCB parameters specified on the SYSUT1 and SYSUT2 DD statements (or the SYSUT2 DD statement if that is the only one specified) must be the same as the DCB parameters already existing for the data set.

If an UPDATE=INPLACE operation is specified, the SYSUT2 DD statement should not be coded.

If both the SYSUT1 and SYSUT2 DD statements specify standard user labels (SUL), IEBUPDTE copies user labels from SYSUT1 to SYSUT2.

If the SYSUT1 and SYSUT2 DD statements define the same partitioned data set or PDSE, the old master data set can be updated without creating a new master data set; in this case, a copy of the updated member or members is written within the extent of the space originally allocated to the old master data set. Subsequent referrals to the updated members will point to the newly written members. The member names themselves should not appear on the DD statements; they should be referred to only through IEBUPDTE control statements. The old directory entry for each member is not copied.

Related reading: For more information, see [“EXEC Statement” on page 222](#).

SYSIN DD Statement

The SYSIN DD statement is required for each use of IEBUPDTE. The control data set contains 80-byte, blocked or unblocked records.

Utility Control Statements

Table 38 on page 224 shows the utility control statements used to control IEBUPDTE.

Table 38. IEBUPDTE utility control statements

Statement	Use
Function	Begins an IEBUPDTE operation (ADD, CHANGE, REPL, REPRO).
Detail	Used with the function statement for special applications.
Data	A logical record of data to be used as a new or replacement record in the output data set.
LABEL	Indicates that the data statements are to be treated as user labels.
ALIAS	Assigns aliases.
ENDUP	Stops IEBUPDTE.

Requirement: Unlike other utility control statements, all IEBUPDTE utility control statements (not including data statements) must begin with a "/" (period, slash) in columns 1 and 2.

Continuation requirements for utility control statements are described in [“Continuing utility control statements”](#) on page 8.

Function Statement

The function statement (ADD, CHANGE, REPL, or REPRO) is used to begin an IEBUPDTE operation. At least one function statement must be provided for each member or data set to be processed.

A member or a data set can be added directly to an old master data set if the space originally allocated to the old master is sufficient to incorporate that new member or data set. ADD specifies that a member or a data set is added to an old master data set. If a member is added and the member name already exists in the old master data set, processing is ended. If, however, PARM=NEW is specified on the EXEC statement, the member is replaced. For a sequential output master data set, PARM=NEW must always be specified on the EXEC statement.

When a member replaces an identically named member on the old master data set or a member is changed and rewritten on the old master, the alias (if any) of the original member still refers to the original member. However, if an identical alias is specified for the newly written member, the original alias entry in the directory is changed to refer to the newly written member.

REPL specifies that a member of a data set is being entered in its entirety as a replacement for a sequential data set or for a member of the old master data set. The member name must already exist in the old master data set. CHANGE specifies that modifications are to be made to an existing member or data set. Use of the CHANGE function statement without a NUMBER or DELETE detail statement, or a data statement causes an error condition. REPRO specifies that a member or a data set is copied in its entirety to a new master data set.

Members are logically deleted from a copy of a library by being omitted from a series of REPRO function statements within the same job step.

One sequential data set can be copied in a given job step. A sequential data set is logically deleted from a new volume by being omitted from a series of job steps which copy only the desired data sets to the new volume. If the NEW subparameter is coded in the EXEC statement, only the ADD function statement is permitted.

The syntax of the FUNCTION statement is:

Label	Statement	Parameters
. / [<i>label</i>]	{ADD CHANGE REPL REPRO}	[LIST=ALL] [, SEQFLD={ <i>ddl</i> (<i>ddl</i> , <i>ddl</i>) }] [, NEW={PO PS}] [, MEMBER= <i>membername</i>] [, COLUMN={ <i>nn</i> <u>1</u> }] [, UPDATE=INPLACE] [, INHDR= <i>routinename</i>] [, INTLR= <i>routinename</i>] [, OUTHDR= <i>routinename</i>] [, OUTTLR= <i>routinename</i>] [, TOTAL=(<i>routinename</i> , <i>size</i>)] [, NAME= <i>membername</i>] [, LEVEL= <i>hh</i>] [, SOURCE= <i>x</i>] [, SSI= <i>hhhhhhhh</i>]
Note: COLUMN and UPDATE=INPLACE can only be used with CHANGE		

where:

LIST=ALL

specifies that the SYSPRINT data set is to contain the entire updated member or data set and the control statements used in its creation.

Default: For old data sets, if LIST is omitted, the SYSPRINT data set contains modifications and control statements only. If UPDATE was specified, the entire updated member is listed only when renumbering has been done. For new data sets, the entire member or data set and the control statements used in its creation are always written to the SYSPRINT data set.

SEQFLD={*ddl*|(*ddl*,*ddl*)}

ddl specifies, in decimal, the starting column (up to column 80) and length (8 or less) of sequence numbers within existing logical records and subsequent data statements. Note that the starting column specification (*dd*) plus the length (*l*) cannot exceed the logical record length (LRECL) plus 1. Sequence numbers on incoming data statements and existing logical records must be padded to the end with enough zeros to fill the length of the sequence field.

(*ddl*,*ddl*)

may be used when an alphanumeric sequence number generation is required. The first *ddl* specifies the sequence number columns as previously described. The second *ddl* specifies, in decimal, the starting column (up to column 80) and length (8 or less) of the numeric portion of the sequence numbers in subsequent NUMBER statements. This information is used to determine which portion of the sequence number specified by the NEW1 parameter may be increased and which portions should be copied to generate a new sequence number for inserted or renumbered records.

The numeric columns must fall within the sequence number columns specified (or defaulted) by the first *ddl*.

Default: 738 is assumed, that is, an 8-byte sequence number beginning in column 73. Therefore, if existing logical records and subsequent data statements have sequence numbers in columns 73 through 80, this keyword need not be coded.

NEW={PO|PS}

specifies the organization of the old master data set and the organization of the updated output. NEW should not be specified unless the organization of the new master data set is different from the organization of the old master. NEW can only be coded on the first control record. These values can be coded:

PO

specifies that the old master data set is a sequential data set, and that the updated output is to become a member of a partitioned data set or PDSE.

PS

specifies that the old master data set is a partitioned data set or PDSE, and that a member of that data set is to be converted into a sequential data set.

MEMBER=membername

specifies a name to be assigned to the member placed in the partitioned data set or PDSE defined by the SYSUT2 DD statement. MEMBER is used only when SYSUT1 defines a sequential data set, SYSUT2 defines a partitioned data set or PDSE, and NEW=PO is specified.

COLUMN={nn|1}

specifies, in decimal, the starting column of a data field within a logical record. The field extends to the end of the logical record. Within an existing logical record, the data in the defined field is replaced by data from a subsequent data statement.

COLUMN may only be coded with CHANGE.

UPDATE=INPLACE

specifies that the old master data set is to be updated within the space it actually occupies. The old master data set must reside on a direct access device. UPDATE=INPLACE is valid only when coded with CHANGE. No other function statements (ADD, REPL, REPRO) may be in the same job step.

INHDR=routinename

specifies the name of the user routine that handles any user input (SYSUT1) header labels. This parameter is valid only when a sequential data set is being processed.

INTLR=routinename

specifies the name of the user routine that handles any user input (SYSUT1) trailer labels. INTLR is valid only when a sequential data set is being processed, but not when UPDATE=INPLACE is coded.

OUTHDR=routinename

specifies the name of the user routine that handles any user output (SYSUT2) header labels. OUTHDR is valid only when a sequential data set is being processed, but not when UPDATE=INPLACE is coded.

OUTTLR=routinename

specifies the name of the user routine that handles any user output (SYSUT2) trailer labels. OUTTLR is valid only when a sequential data set is being processed, but not when UPDATE=INPLACE is coded.

TOTAL=(routinename,size)

specifies that exits to a user's routine are to be provided prior to writing each record. This parameter is valid only when a sequential data set is being processed. These values are coded:

routinename

specifies the name of your totaling routine.

size

specifies the number of bytes required for your data. The size should not exceed 32K, nor be less than 2 bytes. In addition, the keyword OPTCD=T must be specified for the SYSUT2 (output) DD statement.

NAME=membername

indicates the name of the member placed into the partitioned data set or PDSE. The member name need not be specified in the DD statement itself. NAME must be provided to identify each input member. Do not specify NAME if the input is a sequential data set. This parameter is valid only when a member of a partitioned data set or PDSE is being processed.

LEVEL=hh

specifies the change (update) level in hexadecimal (00-FF). The level number is recorded in the directory entry of the output member. The level number is in the first byte of the SSI, system status information. If the member entry contains ISPF information, the level number will overlay part of it. This parameter is valid only when a member of a partitioned data set or PDSE is being processed. LEVEL has no effect when SSI is specified.

SOURCE=x

specifies your modifications when the x value is 0, or IBM modifications when the x value is 1. The source is recorded in the directory entry of the output member. This value is in the second byte of the

SSI, system status information. If the member entry contains ISPF information, this value will overlay one bit of it. This parameter is valid only when a member of a partitioned data set or PDSE is being processed. SOURCE has no effect when SSI is specified.

SSI=hhhhhhh

specifies eight hexadecimal characters of system status information (SSI) to be placed in the directory of the new master data set as four packed decimal bytes of user data. If the member entry contains ISPF information, the SSI will overlay part of it. PTFs (software updates from IBM) typically set the SSI. This parameter is valid only when a member of a partitioned data set or PDSE is being processed. SSI overrides any LEVEL or SOURCE parameter given on the same function statement.

Related reading:

- For information about the use of NEW with NAME and MEMBER, see [Table 39 on page 228](#).
- For information about function restrictions, see [“Function Restrictions” on page 227](#).
- For information about updating user input header labels, see [Table 8 on page 34](#).
- For a information about linkage conventions for user routines, see [Appendix C, “Specifying User Exits with Utility Programs,” on page 343](#).

Function Restrictions

When UPDATE=INPLACE is specified:

- The SYSUT2 DD statement is not coded.
- The PARM parameter of the EXEC statement must imply or specify MOD.
- The NUMBER detail statement can be used to specify a renumbering operation.
- Data statements can be used to specify replacement information only.
- One CHANGE function statement and one UPDATE=INPLACE parameter are permitted per job step.
- No functions other than replacement, renumbering, and header label modification (via the LABEL statement) can be specified.
- Unless the entire data set is renumbered, only replaced records are listed.
- System status information cannot be changed.

When REPRO is specified, the ADD statement can be used in the same job step only if both SYSUT1 and SYSUT2 are partitioned data sets or PDSEs; otherwise, unpredictable results can occur.

Within an existing logical record, the data in the field defined by the COLUMN parameter is replaced by data from a subsequent data statement, as follows:

1. IEBUPDTE matches a sequence number of a data statement with a sequence number of an existing logical record. In this manner, the COLUMN specification is applied to a specific logical record.
2. The information in the field within the data statement replaces the information in the field within the existing logical record. For example, COLUMN=40 indicates that columns 40 through 80 (assuming 80-byte logical records) of a subsequent data statement are to be used as replacement data for columns 40 through 80 of a logical record identified by a matching sequence number. (A sequence number in an existing logical record or data statement need not be within the defined field.)

The COLUMN specification applies to the entire function, with the exception of:

- Logical records deleted by a subsequent DELETE detail statement.
- Subsequent data statements not having a matching sequence number for an existing logical record.
- Data statements containing information to be inserted in the place of a deleted logical record or records.

[Table 39 on page 228](#) shows the use of NEW, MEMBER, and NAME parameters for different input and output data set organizations.

Table 39. NEW, MEMBER and NAME parameters of the function statements

Input Data Set Organization	Output Data Set Organization	Parameter Combinations
None	Partitioned or PDSE (New)	With each ADD function statement, use NAME to assign a name for each member to be placed in the data set.
Partitioned or PDSE	Partitioned or PDSE	<p>With an ADD function statement, use NAME to specify the name of the member to be placed in the data set defined by the SYSUT2 DD statement. If an additional name is required, an ALIAS statement can also be used.</p> <p>With a CHANGE, REPL, or REPRO function statement, use NAME to specify the name of the member within the data set defined by the SYSUT1 DD statement. If a different or additional name is desired for the member in the data set defined by the SYSUT2 DD statement, use an ALIAS statement also.</p>
Partitioned or PDSE	Sequential	With any function statement, use NAME to specify the name of the member in the data set defined by the SYSUT1 DD statement. Use NEW=PS to specify the change in organization from partitioned to sequential. (The name and file sequence number, if any, assigned to the output master data set are specified in the SYSUT2 DD statement.)
Sequential	Partitioned or PDSE	With applicable function statement, use MEMBER to assign a name to the member to be placed in the data set defined by the SYSUT2 DD statement. Use NEW=PO to specify the change in organization from sequential to partitioned.

Detail Statement

A detail statement is used with a function statement for certain applications, such as deleting or renumbering selected logical records. The NUMBER detail statement specifies, when coded with a CHANGE function statement, that the sequence number of one or more logical records is changed. It specifies, when coded with an ADD or REPL function statement, the sequence numbers to be assigned to the records within new or replacement members or data sets. When used with an ADD or REPL function statement, no more than one NUMBER detail statement is permitted for each ADD or REPL function statement.

The DELETE detail statement specifies, when coded with a CHANGE function statement, that one or more logical records are to be deleted from a member or data set.

Logical records cannot be deleted in part; that is, a COLUMN parameter specification in a function statement is not applicable to records that are to be deleted. Each specific sequence number is handled only once in any single operation.

The syntax of the DETAIL statement is:

Label	Statement	Parameters
. / [label]	{NUMBER DELETE}	[SEQ1={cccccccc ALL}] [, SEQ2=cccccccc] [, NEW1=cccccccc] [, INCR=cccccccc] [, INSERT=YES]
Note: NEW1, INCR and INSERT can only be used with NUMBER		

where:

SEQ1={cccccccc|ALL}

specifies records to be renumbered, deleted, or assigned sequence numbers. These values can be coded:

cccccccc

specifies the sequence number of the first logical record to be renumbered or deleted. This value is not coded in a NUMBER detail statement that is used with an ADD or REPL function statement.

When this value is used in an insert operation, it specifies the existing logical record after which an insertion is to be made. It must not equal the number of a statement just replaced or added. Refer to the INSERT parameter for additional discussion.

ALL

specifies a renumbering operation for the entire member or data set. ALL is used only when a CHANGE function statement and a NUMBER detail statement are used. ALL must be coded if sequence numbers are to be assigned to existing logical records having blank sequence numbers. If ALL is not coded, all existing logical records having blank sequence numbers are copied directly to the output master data set. When ALL is coded, SEQ2 need not be coded and one NUMBER detail statement is permitted per function statement. Refer to the INSERT parameter for additional discussion.

SEQ2=cccccccc

specifies the sequence number of the last logical record to be renumbered or deleted. SEQ2 is required on all DELETE detail statements. If only one record is to be deleted, the SEQ1 and SEQ2 specifications must be identical. SEQ2 is not coded in a NUMBER detail statement that is used with an ADD or REPL function statement.

NEW1=cccccccc

specifies the first sequence number assigned to new or replacement data, or specifies the first sequence number assigned in a renumbering operation. A value specified in NEW1 must be greater than a value specified in SEQ1 (unless SEQ1=ALL is specified, in which case this rule does not apply).

INCR=cccccccc

specifies an increment value used for assigning successive sequence numbers to new or replacement logical records, or specifies an increment value used for renumbering existing logical records.

INSERT=YES

specifies the insertion of a block of logical records. The records, which are data statements containing blank sequence numbers, are numbered and inserted in the output master data set. INSERT is valid only when coded with both a CHANGE function statement and a NUMBER detail statement. SEQ1, NEW1 and INCR are required on the first NUMBER detail statement.

When INSERT=YES is coded:

- The SEQ1 parameter specifies the existing logical record after which the insertion is made. SEQ1=ALL cannot be coded.
- The SEQ2 parameter need not be coded.
- The NEW1 parameter assigns a sequence number to the first logical record to be inserted. If the parameter is alphanumeric, the SEQFLD=(ddl,ddl) parameter should be coded on the function statement.
- The INCR parameter is used to renumber as much as is necessary of the member or data set from the point of the first insertion; the member or data set is renumbered until an existing logical record is found whose sequence number is equal to or greater than the next sequence number to be assigned. If no such logical record is found, the entire member or data set is renumbered.
- Additional NUMBER detail statements, if any, must specify INSERT=YES. If a prior numbering operation renumbers the logical record specified in the SEQ1 parameter of a subsequent NUMBER detail statement, any NEW1 or INCR parameter specifications in the latter NUMBER detail statement are overridden. The prior increment value is used to assign the next successive sequence numbers. If a prior numbering operation does not renumber the logical record specified in the SEQ1 parameter of a subsequent NUMBER detail statement, the latter statement must contain NEW1 and INCR specifications.
- The block of data statements to be inserted must contain blank sequence numbers.
- The insert operation is stopped when a function statement, a detail statement, an end-of-file indication, or a data statement containing a sequence number is encountered.

Detail Restrictions

The SEQ1, SEQ2 and NEW1 parameters (with the exception of SEQ1=ALL) specify eight (maximum) alphanumeric characters. The INCR parameter specifies eight (maximum) numeric characters. Only the significant part of a numeric sequence number need be coded; for example, SEQ1=00000010 can be shortened to SEQ1=10. If, however, the numbers are alphanumeric, the alphabetic characters must be specified; for example, SEQ1=00ABC010 can be shortened to SEQ1=ABC010.

Data Statement

A data statement is used with a function statement, or with a function statement and a detail statement. It contains a logical record used as replacement data for an existing logical record, or new data to be incorporated in the output master data set.

Each data statement contains one logical record, which begins in the first column of the data statement. The length of the logical record is equal to the logical record length (LRECL) specified for the output master data set. Each logical record contains a sequence number to determine where the data is placed in the output master data set (except when INSERT=YES is specified on a NUMBER statement).

When used with a CHANGE function statement, a data statement contains new or replacement data, as follows:

- If the sequence number in the data statement is identical to a sequence number in an existing logical record, the data statement replaces the existing logical record in the output master data set.
- If no corresponding sequence number is found within the existing records, the data statement is inserted in the proper collating sequence within the output master data set. (For proper execution of this function, all records in the old master data set must have a sequence number.)
- If a data statement with a sequence number is used and INSERT=YES was specified, the insert operation is stopped. IEBUPDTE will continue processing if this sequence number is at least equal to the next old master record (record following the referred to sequence record).

When used with an ADD or REPL function statement, a data statement contains new data to be placed in the output master data set.

Sequence numbers within the old master data set are assumed to be in ascending order. No validity checking of sequence numbers is performed for data statements or existing records.

Sequence numbers in data statements must be in the same relative position as sequence numbers in existing logical records. (Sequence numbers include leading zeros and are assumed to be in columns 73 through 80; if the numbers are in columns other than these, the length and relative position must be specified in a SEQFLD parameter within a preceding function statement.)

LABEL Statement

The LABEL statement indicates that the data statements (called label data statements) are to be treated as user labels. These new user labels are placed on the output data set. The next control statement indicates to IEBUPDTE that the last label data statement of the group has been read.

The syntax of the LABEL statement is:

Label	Statement	Parameters
./ [name]	LABEL	

There can be no more than two LABEL statements per execution of IEBUPDTE. There can be no more than eight label data statements following any LABEL statement. The first 4 bytes of each 80-byte label data statement must contain "UHLn" or "UTLn", where *n* is 1 through 8, for input header or input trailer labels respectively, to conform to IBM standards for user labels. Otherwise, data management will overlay the data with the proper four characters.

When IEBUPDTE encounters a LABEL statement, it reads up to eight data statements and saves them for processing by user output label routines. If there are no such routines, the saved records are written by OPEN or CLOSE as user labels on the output data set. If there are user output label processing routines, IEBUPDTE passes a parameter list to the output label routines. The label buffer contains a label data record which the user routine can process before the record is written as a label. If the user routine specifies (via return codes to IEBUPDTE) more entries than there are label data records, the label buffer will contain meaningless information for the remaining entries to the user routine.

The position of the LABEL statement in the SYSIN data set, relative to any function statements, indicates the type of user label that follows the LABEL statement:

- To create output header labels, place the LABEL statement and its associated label data statements before any function statements in the input stream. A function statement, other than LABEL, must follow the last label data statement of the group.
- To create output trailer labels, place the LABEL statement and its associated label data statements after any function statements in the input stream, but before the ENDUP statement. The ENDUP statement is not optional in this case. It must follow the last label data statement of the group if IEBUPDTE is to create output trailer labels.

When UPDATE=INPLACE is specified in a CHANGE statement, user input header labels can be updated by user routines, but input trailer and output labels cannot be updated by user routines. User labels cannot be added or deleted. User input header labels are made available to user routines by the label buffer address in the parameter list. The return codes when CHANGE UPDATE=INPLACE is used differ slightly from the standard return codes.

If you want to examine the replaced labels from the old master data set, you must:

1. Specify an update of the old master by coding the UPDATE=INPLACE parameter in a function statement.
2. Include a LABEL statement in the input data set for either header or trailer labels.
3. Specify a corresponding user label routine.

If these conditions are met, fourth and fifth parameter words will be added to the standard parameter list. The fourth parameter word is not now used; the fifth contains a pointer to the replaced label from the old master. In this case, the number of labels supplied in the SYSIN data set must not exceed the number of labels on the old master data set. If you specify, via return codes, more entries to the user's header label routine than there are labels in the input stream, the first parameter will point to the current header label on the old master data set for the remaining entries. In this case, the fifth parameter is meaningless.

Related reading:

- For more information, see [Appendix C, “Specifying User Exits with Utility Programs,” on page 343](#).
- For more information about the linkage between utility programs and user label processing, see [“Processing User Labels” on page 347](#).
- For information about return codes, see [Appendix C, “Specifying User Exits with Utility Programs,” on page 343](#).

ALIAS Statement

The ALIAS statement is used to create or retain an alias in an output (partitioned or PDSE) directory. The ALIAS statement can be used with any of the function statements. Multiple aliases can be assigned to each member, up to a maximum of 16 aliases.

If an ALIAS statement specifies a name that already exists on the data set, the original TTR (track record) of that directory entry will be destroyed.

If ALIAS statements are used, they must follow the data statements, if any, in the input stream.

The syntax of the ALIAS statement is:

Label	Statement	Parameters
. / [label]	ALIAS	NAME= <i>aliasname</i>

where:

NAME=*aliasname*

specifies a 1- to 8-character alias name.

ENDUP Statement

An ENDUP statement is optional. It is used to indicate the end of SYSIN input to this job step. If there is no other preceding delimiter statement, it serves as an end-of-data indication. The ENDUP statement follows the last group of SYSIN control statements.

When the ENDUP statement is used, it must follow the last label data statement if IEBUPDTE is used to create output trailer labels.

The syntax of the ENDUP statement is:

Label	Statement	Parameters
. / [label]	ENDUP	

IEBUPDTE Examples

These examples illustrate some of the uses of IEBUPDTE. You can use [Table 40 on page 232](#) as a quick-reference guide to IEBUPDTE examples.

Table 40. IEBUPDTE example directory

Operation	Data Set Organization	Device	Comments	Example
ADD and REPL	Partitioned	Disk	A JCL procedure is stored as a new member of a procedure library (PROCLIB). Another JCL procedure is to replace an existing member in PROCLIB.	“Example 1: Place Two Procedures in SYS1.PROCLIB” on page 233
COPY	Sequential	Disk	Sequential data set is copied from one direct access volume to another; user labels can be processed by exit routines.	“Example 10: Copy Sequential Data Set from One Volume to Another” on page 241
CREATE	Partitioned	Disk	Create a new generation.	“Example 11: Create a New Partitioned Data Set” on page 241
CREATE	Sequential	Card Reader and Disk	Sequential data set with user labels is created from card input.	“Example 9: Create a Sequential Data Set from Card Input” on page 240
CREATE a partitioned data set	Partitioned	Disk	Input from control data set and from existing partitioned data set. Output partitioned data set is to contain three members.	“Example 3: Create New Library Using SYS1.MACLIB as a Source” on page 234
CREATE a partitioned library	Partitioned	Disk	Input data is in the control data set. Output partitioned data set is to contain three members.	“Example 2: Create a Three-Member Library” on page 234

Table 40. IEBUPDTE example directory (continued)

Operation	Data Set Organization	Device	Comments	Example
CREATE and DELETE	Partitioned, Sequential	Disk and Tape	Sequential master is created from partitioned disk input. Selected records are to be deleted. Blocked output.	“Example 5: Create New Master Data Set and Delete Selected Records” on page 236
CREATE, DELETE, and UPDATE	Sequential, Partitioned	Tape and Disk	Partitioned data set is created from sequential input. Records are to be deleted and updated. Sequence numbers in columns other than 73 through 80. One member is placed in the output data set.	“Example 6: Create and Update a Library Member” on page 236
INSERT	Partitioned	Disk	Block of logical records is inserted into an existing member.	“Example 7: Insert Records into a Library Member” on page 237
INSERT	Partitioned	Disk	Two blocks of logical records are to be inserted into an existing member. Sequence numbers are alphanumeric.	“Example 8: Renumber and Insert Records into a Library Member” on page 238
UPDATE INPLACE and renumber	Partitioned	Disk	Input data set is considered to be the output data set as well; therefore, no SYSUT2 DD statement is required.	“Example 4: Update a Library Member” on page 235

Examples that use **disk** or **tape** in place of actual device numbers must be changed before use. The actual device numbers depend on how your installation has defined the devices to your system.

Example 1: Place Two Procedures in SYS1.PROCLIB

In this example, two procedures are to be placed in the cataloged procedure library, SYS1.PROCLIB. The example assumes that the two procedures can be accommodated within the space originally allocated to the procedure library.

```
//UPDATE JOB ...
//STEP1 EXEC PGM=IEBUPDTE,PARM=MOD
//SYSPRINT DD SYSOUT=A
//SYSUT1 DD DSN=SYS1.PROCLIB,DISP=OLD
//SYSUT2 DD DSN=SYS1.PROCLIB,DISP=OLD
//SYSIN DD DATA
./ ADD LIST=ALL,NAME=ERASE,LEVEL=01,SOURCE=0
./ NUMBER NEW1=10,INCR=10
//ERASE EXEC PGM=IEBUPDTE
//DD1 DD UNIT=disk,DISP=(OLD,KEEP),VOLUME=SER=111111
//SYSPRINT DD SYSOUT=A
./ REPL LIST=ALL,NAME=LISTPROC
./ NUMBER NEW1=10,INCR=10
//LIST EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=A
//SYSUT1 DD DISP=SHR,DSN=SYS1.PROCLIB(&MEMBER)
//SYSUT2 DD SYSOUT=A,DCB=(RECFM=F,BLKSIZE=80)
//SYSIN DD DATA
./ ENDUP
/*
```

The control statements are as follows:

- SYSUT1 and SYSUT2 DD define the SYS1.PROCLIB data set, which is assumed to be cataloged.
- SYSIN DD defines the control data set, which follows in the input stream. The data set contains the utility control statements and the data to be placed in the procedure library.
- The ADD function statement indicates that records (data statements) in the control data set are to be placed in the output. The newly created procedure, ERASE, is listed in the message data set.

The ADD function will not take place if a member named ERASE already exists in the new master data set referenced by SYSUT2.

- The first NUMBER detail statement indicates that the new and replacement procedures are to be assigned sequence numbers. The first record of each procedure is assigned sequence number 10; the next record is assigned sequence number 20, and so on.
- The ERASE EXEC statement marks the beginning of the first new procedure.
- The REPL function statement indicates that records (data statements) in the control data set are to replace an already existing member. The member is stored in the new master data set referenced by SYSUT2. The REPL function will only take place if a member named LISTPROC already exists in the old master data set referenced by SYSUT1.
- The second NUMBER detail statement is a duplicate of the first.
- The LIST EXEC statement marks the beginning of the second new procedure.
- The ENDUP statement marks the end of the SYSIN DD input data.

Example 2: Create a Three-Member Library

In this example, a three-member partitioned library is created. The input data is contained solely in the control data set.

```
//UPDATE    JOB    ...
//STEP1     EXEC   PGM=IEBUPDTE,PARM=NEW
//SYSPRINT  DD     SYSOUT=A
//SYSUT2    DD     DSNAME=OUTLIB,UNIT=disk,DISP=(NEW,CATLG),
//           VOLUME=SER=111112,SPACE=(TRK,(50,,10)),
//           DCB=(RECFM=F,LRECL=80,BLKSIZE=80)
//SYSIN     DD     DATA
./          ADD     NAME=MEMB1,LEVEL=00,SOURCE=0,LIST=ALL

(Data statements, sequence numbers in columns 73 through 80)

./          ADD     NAME=MEMB2,LEVEL=00,SOURCE=0,LIST=ALL

(Data statements, sequence numbers in columns 73 through 80)

./          ADD     NAME=MEMB3,LEVEL=00,SOURCE=0,LIST=ALL

(Data statements, sequence numbers in columns 73 through 80)

./          ENDUP
/*
```

The control statements are as follows:

- SYSUT2 DD defines the new partitioned master, OUTLIB. Enough space is allocated to allow for subsequent modifications without creating a new master data set.
- SYSIN DD defines the control data set, which follows in the input stream. The data set contains the utility control statements and the data to be placed as three members in the output partitioned data set.
- The ADD function statements indicate that subsequent data statements are to be placed as members in the output partitioned data set. Each ADD function statement specifies a member name for subsequent data and indicates that the member and control statement is listed in the message data set.
- The data statements contain the data to be placed in each member of the output partitioned data set.
- ENDUP signals the end of control data set input.

Because sequence numbers (other than blank numbers) are included within the data statements, no NUMBER detail statements are included in the example.

Example 3: Create New Library Using SYS1.MACLIB as a Source

In this example, a three-member partitioned data set (NEWMCLIB) is created. The data set will contain two members, ATTACH and DETACH, copied from an existing partitioned data set (SYS1.MACLIB), and a new member, EXIT, which is contained in the control data set.

```

//UPDATE   JOB    ...
//STEP1    EXEC   PGM=IEBUPDTE,PARM=MOD
//SYSPRINT DD   SYSOUT=A
//SYSUT1   DD     DSN=SYS1.MACLIB,DISP=SHR
//SYSUT2   DD     DSN=DEV.DRIVER3.NEWMLIB,UNIT=disk,
//           DISP=(NEW,CATLG),SPACE=(TRK,(100,,10)),
//           DCB=(RECFM=F,LRECL=80,BLKSIZE=4000)
//SYSIN    DD     DATA
./         REPRO   NAME=ATTACH,LEVEL=00,SOURCE=1,LIST=ALL
./         REPRO   NAME=DETACH,LEVEL=00,SOURCE=1,LIST=ALL
./         ADD     NAME=EXIT,LEVEL=00,SOURCE=1,LIST=ALL
./         NUMBER  NEW1=10,INCR=100

(Data records for EXIT member)

./         ENDUP
/*

```

The control statements are as follows:

- SYSUT1 DD defines the input partitioned data set SYS1.MACLIB, which is assumed to be cataloged.
- SYSUT2 DD defines the output partitioned data set DEV.DRIVER3.NEWMLIB. Enough space is allocated to allow for subsequent modifications without creating a new master data set.
- SYSIN DD defines the control data set, which follows in the input stream.
- The REPRO function statements identify the existing input members (ATTACH and DETACH) to be copied onto the output data set. These members are also listed in the message data set (because LIST=ALL is specified).
- The ADD function statement indicates that records (subsequent data statements) are to be placed as members in the output partitioned data set, called EXIT. The data statements are to be listed in the message data set.
- The NUMBER detail statement assigns sequence numbers to the data statements. (The data statements contain blank sequence numbers in columns 73 through 80.) The first record of the output member is assigned sequence number 10; subsequent record numbers are increased by 100.
- ENDUP signals the end of SYSIN data.

Note that the three named input members (ATTACH, DETACH, and EXIT) do not have to be specified in the order of their collating sequence in the old master.

Example 4: Update a Library Member

In this example, a member (MODMEMB) is updated within the space it actually occupies. Two existing logical records are replaced, and the entire member is renumbered.

```

//UPDATE   JOB    ...
//STEP1    EXEC   PGM=IEBUPDTE,PARM=MOD
//SYSPRINT DD   SYSOUT=A
//SYSUT1   DD     DSN=PDS,UNIT=disk,DISP=(OLD,KEEP),
//           VOLUME=SER=111112
//SYSIN    DD     *
./         CHANGE  NAME=MODMEMB,LIST=ALL,UPDATE=INPLACE
./         NUMBER  SEQ1=ALL,NEW1=10,INCR=5

(Data statement 1, sequence number 00000020)
(Data statement 2, sequence number 00000035)

/*

```

The control statements are as follows:

- SYSUT1 DD defines the partitioned data set that is updated in place. (Note that the member name need not be specified in the DD statement.)
- SYSIN DD defines the control data set, which follows in the input stream.
- The CHANGE function statement indicates the name of the member to be updated (MODMEMB) and specifies the UPDATE=INPLACE operation. The entire member is listed in the message data set. Note that, as renumbering is being done, and since UPDATE=INPLACE was specified, the listing would have

been provided even if the LIST=ALL parameter had not been specified. See the LIST parameter for more information.

- The NUMBER detail statement indicates that the entire member is to be renumbered, and specifies the first sequence number to be assigned and the increment value (5) for successive sequence numbers.
- The data statements replace existing logical records having sequence numbers of 20 and 35.

Example 5: Create New Master Data Set and Delete Selected Records

In this example, a new master sequential data set is created from partitioned input and selected logical records are deleted.

```
//UPDATE    JOB    ...
//STEP1     EXEC   PGM=IEBUPDTE,PARM=MOD
//SYSPRINT  DD     SYSOUT=A
//SYSUT1    DD     DSN=DCB.PARTDS,DISP=(OLD,KEEP)
//          DD     VOLUME=SER=111112
//SYSUT2    DD     DSN=SEQDS,UNIT=tape,LABEL=(2,SL),
//          DD     DISP=(,KEEP),VOLUME=SER=001234,
//          DD     DCB=(RECFM=FB,LRECL=80,BLKSIZE=2000)
//SYSIN     DD     *
./          CHANGE NEW=PS,NAME=OLDMEMB1

(Data statement 1, sequence number 00000123)

./          DELETE  SEQ1=223,SEQ2=246

(Data statement 2, sequence number 00000224)

/*
```

The control statements are as follows:

- SYSUT1 DD defines the input partitioned data set DCB.PARTDS, which resides on a disk volume.
- SYSUT2 DD defines the output sequential data set, SEQDS. The data set is written as the second data set on a tape volume.
- SYSIN DD defines the control data set, which follows in the input stream.
- CHANGE identifies the input member (OLDMEMB1) and indicates that the output is a sequential data set (NEW=PS).
- The first data statement replaces the logical record whose sequence number is identical to the sequence number in the data statement (00000123). If no such logical record exists, the data statement is incorporated in the proper sequence within the output data set.
- The DELETE detail statement deletes logical records having sequence numbers from 223 through 246, inclusive.
- The second data statement is inserted in the proper sequence in the output data set, because no logical record with the sequence number 224 exists (it was deleted in the previous statement).

Note that only one member can be used as input when converting to sequential organization.

Example 6: Create and Update a Library Member

In this example, a member of a partitioned data set is created from sequential input and existing logical records are updated.

```
//UPDATE    JOB    ...
//STEP1     EXEC   PGM=IEBUPDTE,PARM=MOD
//SYSPRINT  DD     SYSOUT=A
//SYSUT1    DD     DSN=BROWN.OLDSEQDS,UNIT=tape,
//          DD     DISP=(OLD,KEEP),VOLUME=SER=001234
//SYSUT2    DD     DSN=BROWN.NEWPART,UNIT=disk,DISP=(,CATLG),
//          DD     VOLUME=SER=111112,SPACE=(TRK,(10,5,5)),
//          DD     DCB=(RECFM=FB,LRECL=80,BLKSIZE=4080)
//SYSIN     DD     *
./          CHANGE NEW=P0,MEMBER=PARMEM1,LEVEL=01,
./          SEQFLD=605,COLUMN=40,SOURCE=0
```



```

(Data statement 1, sequence number 00020)

./      DELETE      SEQ1=220,SEQ2=250

(Data statement 2, sequence number 00230)
(Data statement 3, sequence number 00260)

./      ALIAS       NAME=MEMB1
/*

```

The control statements are as follows:

- SYSUT1 DD defines the input sequential data set (BROWN.OLDSEQDS). The data set resides on a tape volume.
- SYSUT2 DD defines the output partitioned data set (BROWN.NEWPART). Enough space is allocated to provide for members that may be added in the future.
- SYSIN DD defines the control data set, which follows in the input stream.
- The CHANGE function statement identifies the output member (PARMEM1) and indicates that a conversion from sequential input to partitioned output is made. The SEQFLD parameter indicates that a 5-byte sequence number is located in columns 60 through 64 of each data statement. The COLUMN=40 parameter specifies the starting column of a field (within subsequent data statements) from which replacement information is obtained. SOURCE=0 indicates that the replacement information is provided by you.
- The first data statement is used as replacement data. Columns 40 through 80 of the statement replace columns 40 through 80 of the corresponding logical record. If no such logical record exists, the entire card image is inserted in the output data set member.
- The DELETE detail statement deletes all of the logical records having sequence numbers from 220 through 250.
- The second data statement, whose sequence number falls within the range specified in the DELETE detail statement is incorporated in its entirety in the output data set member.
- The third data statement, which is beyond the range of the DELETE detail statement, is treated in the same manner as the first data statement.
- ALIAS assigns the alias name MEMB1 to the output data set member PARMEM1.

Example 7: Insert Records into a Library Member

In this example, a block of three logical records is inserted into an existing member, and the updated member is placed in the existing partitioned data set.

```

//UPDATE   JOB    ...
//STEP1    EXEC   PGM=IEBUPDTE,PARM=MOD
//SYSPRINT DD     SYSOUT=A
//SYSUT1   DD     DSNAME=PDS,UNIT=disk,DISP=(OLD,KEEP),
//          VOLUME=SER=111112
//SYSUT2   DD     DSNAME=PDS,UNIT=disk,DISP=(OLD,KEEP),
//          VOLUME=SER=111112
//SYSIN    DD     *
./         CHANGE  NAME=RENUM,LIST=ALL,LEVEL=01,SOURCE=0
./         NUMBER  SEQ1=15,NEW1=20,INCR=5,INSERT=YES

(Data statement 1)
(Data statement 2)
(Data statement 3)

/*

```

The control statements are as follows:

- SYSUT1 and SYSUT2 DD define the partitioned data set (PDS).
- SYSIN DD defines the control data set, which follows in the input stream.
- The CHANGE function statement identifies the input member RENUM. The entire member is listed in the message data set.

- The NUMBER detail statement specifies the insert operation and controls the renumbering operation.
- The data statements are the logical records to be inserted. (Sequence numbers are assigned when the data statements are inserted.)

In this example, the existing logical records have sequence numbers 10, 15, 20, 25, and 30. Sequence numbers are assigned by the NUMBER detail statement, as follows:

1. Data statement 1 is assigned sequence number 20 (NEW1=20) and inserted after existing logical record 15 (SEQ1=15).
2. Data statements 2 and 3 are assigned sequence numbers 25 and 30 (INCR=5) and are inserted after data statement 1.
3. Existing logical records 20, 25, and 30 are assigned sequence numbers 35, 40, and 45, respectively.

Table 41 on page 238 shows existing sequence numbers, data statements inserted, and the resultant new sequence numbers.

Table 41. Example of reordered sequence numbers

Sequence Numbers and Data Statements Inserted	New Sequence Numbers
10	10
15	15
Data statement 1	20
Data statement 2	25
Data statement 3	30
20	35
25	40
30	45

Example 8: Renumber and Insert Records into a Library Member

In this example, two blocks (three logical records per block) are inserted into an existing member, and the member is placed in the existing partitioned data set. A portion of the output member is also renumbered.

```
//UPDATE    JOB    ...
//STEP1     EXEC   PGM=IEBUPDTE,PARM=MOD
//SYSPRINT  DD     SYSOUT=A
//SYSUT1    DD     DSNAME=PDS,UNIT=disk,DISP=(OLD,KEEP),
//           VOLUME=SER=111112
//SYSUT2    DD     DSNAME=PDS,UNIT=disk,DISP=(OLD,KEEP),
//           VOLUME=SER=111112
//SYSIN     DD     *
./  CHANGE  NAME=RENUM,LIST=ALL,LEVEL=01,SOURCE=0,SEQFLD=(765,783)
./  NUMBER  SEQ1=AA015,NEW1=AA020,INCR=5,INSERT=YES

(Data statement 1)
(Data statement 2)
(Data statement 3)

./  NUMBER  SEQ1=AA030,INSERT=YES

(Data statement 4)
(Data statement 5)
(Data statement 6)
(Data statement 7, sequence number AA035)

/*
```

The control statements are as follows:

- SYSUT1 and SYSUT2 DD define the partitioned data set PDS.

- SYSIN DD defines the control data set, which follows in the input stream.
- The CHANGE function statement identifies the input member RENUM. The entire member is listed in the message data set.
- The NUMBER detail statements specify the insert operations (INSERT=YES) and control the renumbering operation.
- Data statements 1, 2, 3, and 4, 5, 6 are the blocks of logical records to be inserted. Because they contain blank sequence numbers, sequence numbers are assigned when the data statements are inserted.
- Data statement 7, because it contains a sequence number, stops the insert operation. The sequence number is identical to the number on the next record in the old master data set; consequently, data statement 7 will replace the equally numbered old master record in the output data set.

The existing logical records in this example have sequence numbers AA010, AA015, AA020, AA025, AA030, AA035, AA040, AA045, AA050, BB010, and BB015. The insertion and renumbering operations are performed as follows:

1. Data statement 1 is assigned sequence number AA020 (NEW1=AA020) and inserted after existing logical record AA015 (SEQ1=AA015).
2. Data statements 2 and 3 are assigned sequence numbers AA025 and AA030 (INCR=5) and are inserted after data statement 1.
3. Existing logical records AA020, AA025, and AA030 are assigned sequence numbers AA035, AA040, and AA045, respectively.
4. Data statement 4 is assigned sequence number AA050 and inserted. (The SEQ1=AA030 specification in the second NUMBER statement places this data statement after existing logical record AA030, which has become logical record AA045.)
5. Data statements 5 and 6 are assigned sequence numbers AA055 and AA060 and are inserted after data statement 4.
6. Existing logical record AA035 is replaced by data statement 7, which is assigned sequence number AA065.
7. The remaining logical records in the member are renumbered until logical record BB010 is encountered. Because this record has a sequence number higher than the next number to be assigned, the renumbering operation is ended.

Table 42 on page 239 shows existing sequence numbers, data statements inserted, and the new sequence numbers.

<i>Table 42. Reordered sequence numbers</i>	
Sequence Numbers and Data Statements Inserted	New Sequence Numbers
AA010	AA010
AA015	AA015
Data statement 1	AA020
Data statement 2	AA025
Data statement 3	AA030
AA020	AA035
AA025	AA040
AA030	AA045
Data statement 4	AA050
Data statement 5	AA055

Table 42. Reordered sequence numbers (continued)

Sequence Numbers and Data Statements Inserted	New Sequence Numbers
Data statement 6	AA060
AA035 (Data statement 7)	AA065
AA040	AA070
AA045	AA070
AA050	AA075
BB010	BB010
BB015	BB015

Example 9: Create a Sequential Data Set from Card Input

In this example, IEBUPDTE is used to create a sequential data set from card input. User header and trailer labels, also from the input stream, are placed on this sequential data set.

```
//LABEL      JOB      ...
//CREATION   EXEC     PGM=IEBUPDTE,PARM=NEW
//SYSPRINT   DD       SYSOUT=A
//SYSUT2     DD       DSNAME=LABEL,VOLUME=SER=123456,UNIT=disk,
//              DISP=(NEW,KEEP),LABEL=(,SUL),SPACE=(TRK,(15,3))
//SYSIN      DD       *
./          LABEL

(Header labels)

./          ADD      LIST=ALL,OUTHDR=ROUTINE1,OUTTLR=ROUTINE2

(Data records)

./          LABEL

(Trailer labels)

./          ENDUP
/*
```

The control statements are as follows:

- SYSUT2 DD defines and allocates space for the output sequential data set, called LABEL, which resides on a disk volume.
- SYSIN DD defines the control data set, which follows in the input stream. (This control data set includes the sequential input data set and the user labels, which are on cards.)
- The first LABEL statement identifies the 80-byte card images in the input stream which will become user header labels. (They can be modified by the user's header-label processing routine specified on the ADD function statement.)
- The ADD function statement indicates that the data statements that follow are placed in the output data set. The newly created data set is listed in the message data set. User output header and output trailer routines are to be given control before the writing of header and trailer labels.
- The second LABEL statement identifies the 80-byte card images in the input stream which will become user trailer labels. (They can be modified by the user's trailer-label processing routine specified on the ADD function statement.)
- ENDUP signals the end of the control data set.

Example 10: Copy Sequential Data Set from One Volume to Another

In this example, IEBUPDTE is used to copy a sequential data set from one DASD volume to another. User labels are processed by user exit routines.

```
//LABELS JOB ...
//STEP1 EXEC PGM=IEBUPDTE,PARM=(MOD,,INTLRTN)
//SYSPRINT DD SYSOUT=A
//SYSUT1 DD DSN=OLDMAST,DISP=OLD,LABEL=(,SUL),
//          VOLUME=SER=111111,UNIT=disk
//SYSUT2 DD DSN=NEWMAST,DISP=(NEW,KEEP),LABEL=(,SUL),
//          UNIT=disk,VOLUME=SER=XB182,
//          SPACE=(TRK,(5,10))
//SYSIN DD DSN=INPUT,DISP=OLD,LABEL=(,SUL),
//          VOLUME=SER=222222,UNIT=disk
/*
```

The control statements are as follows:

- SYSUT1 DD defines the input sequential data set, called OLDMAST, which resides on a disk volume.
- SYSUT2 DD defines the output sequential data set, called NEWMAST, which will reside on a disk volume.
- SYSIN DD defines the control data set. The contents of this disk-resident data set in this example are:

```
./ REPRO LIST=ALL,INHDR=INHRTN,INTLR=INTRTN, C
./ OUTHDR=OUTHRTN,OUTTLR=OUTTRN
./ ENDUP
```

- The REPRO function statement indicates that the existing input sequential data set is copied to the output data set. This output data set is listed on the message data set. The user's label processing routines are to be given control when header or trailer labels are encountered on either the input or the output data set.
- ENDUP indicates the end of the control data set.

Example 11: Create a New Partitioned Data Set

In this example, a partitioned generation data set consisting of three members is used as source data in the creation of a new partitioned data set. IEBUPDTE is also used to add a fourth member to the three source members and to number the new member. The resultant data set is cataloged as a new partitioned data set.

```
//NEWGDS JOB ...
//STEP1 EXEC PGM=IEBUPDTE,PARM=MOD
//SYSPRINT DD SYSOUT=A
//SYSUT1 DD DSN=A.B.C,DISP=OLD
//SYSUT2 DD DSN=A.B.C,DISP=(,CATLG),UNIT=disk,
//          VOLUME=SER=111111,SPACE=(TRK,(100,10,10)),
//          DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)
//SYSIN DD DATA
./ REPRO NAME=MEM1,LEVEL=00,SOURCE=0,LIST=ALL
./ REPRO NAME=MEM2,LEVEL=00,SOURCE=0,LIST=ALL
./ REPRO NAME=MEM3,LEVEL=00,SOURCE=0,LIST=ALL
./ ADD NAME=MEM4,LEVEL=00,SOURCE=0,LIST=ALL
./ NUMBER NEW1=10,INCR=5

(Data records comprising MEM4)

./ ENDUP
/*
```

The control statements are as follows:

- SYSUT1 DD defines the partitioned data set, which is used as source data. It can be a PDSE.
- SYSUT2 DD defines the new partitioned data set, which is created from the source partitioned data set and from an additional member included as input and data. It can be a PDSE.
- SYSIN DD defines the control data set, which follows in the input stream.
- The REPRO function statements reproduce the named source members in the output partitioned data set.

- The ADD function statement specifies that the data records following the input stream be included as MEM4.
- The NUMBER detail statement indicates that the new member is to have sequence numbers assigned in columns 73 through 80. The first record is assigned sequence number 10. The sequence number of each successive record is increased by 5.
- ENDUP signals the end of input card data.

Chapter 13. IEHINITT (Initialize Tape) Program

IEHINITT is a system utility used to place standard volume label sets onto any number of magnetic tapes mounted on one or more tape units. They can be ISO/ANSI Version 3 or ISO/ANSI Version 4 volume label sets written in ASCII (American Standard Code for Information Interchange) or IBM standard labels written in EBCDIC.

In this topic, the term "Version 3" is used when referring to ANSI X3.27–1978, ISO 1001–1979 and FIPS 79 standards.

The term "Version 4" is used when referring to ANSI X3.27–1987 level 4 and ISO 1001–1986(E) level 4 standards.

The U.S. government followed Federal Information Processing Standard (FIPS) 79, dated October 17, 1980. It adopted the ISO/ANSI Version 3 standard as a Federal Standard. Later, it withdrew FIPS 79 and did not replace it.

IEHINITT is an APF-authorized program. This means that if another program calls it, that program must also be APF-authorized. To protect system integrity, the calling program must follow the system integrity requirements described in *z/OS MVS Programming: Authorized Assembler Services Guide*.

Because IEHINITT can overwrite previously labeled tapes regardless of expiration date and security protection, IBM recommends that the security administrator use PROGRAM protection with the following sequence of commands:

- RDEFINE PROGRAM IEHINITT ADDMEM('SYS1.LINKLIB'//NOPADCHK) UACC(NONE)
- PERMIT IEHINITT CLASS(PROGRAM) ID(users or groups who should have access) ACCESS(READ)
- SETROPTS WHEN(PROGRAM) REFRESH [Omit REFRESH if you did not have this option active previously]

To further protect against overwriting the wrong tape, IEHINITT asks the operator to verify each tape mount in a non-library environment. Use of IEHINITT in a system-managed tape environment assumes that security controls have been implemented to prevent destruction of production data. SAF/RACF is invoked for authorization processing in a library environment. The level of authority required to proceed with the initialization is UPDATE for CLASS=TAPEVOL.

In addition, installation exits administered via CSVDYNEX, the dynamic exits service, are available. The use of installation exits is completely optional, but if implemented, allows an installation to review all initialization requests and indicate whether a volume should be initialized.

IEHINITT provides volume level security checking using SAF/RACF but does not invoke data set level checking. If you do not use SAF/RACF TAPEVOL profiles, extra precautions may be required when using IEHINITT in a system-managed library.

IEHINITT uses the tape label SVC (SVC number 39) to label tape volumes. The tape label SVC issues a RACROUTE in the TAPEVOL class when it is able to identify the mounted volumes' volser either by reading the tape label or from sense bytes received from the tape drive. UPDATE access to the volume is required to label the volume, or the volume must not be protected; not protected means that either there is no TAPEVOL profile or the TAPEVOL class is not active.

If you plan on using IEHINITT to re-label tape volumes (note that this excludes initializing brand new volumes) and you want to provide protection for labelling and control which users can do this, you should have the TAPEVOL class active and define TAPEVOL profiles. When you use the TAPEAUTHDSN DEVSUPxx option, or use RACF TAPEDSN to protect tape data sets you can define one or more generic TAPEVOL profiles to cover all volumes.

If you use the DFSMSrmm EDGINERS tape labelling and erasing utility you do not need to have the TAPEVOL class active. DFSMSrmm controls who can use the EDGINERS utility and you can only label or erase volumes which are either new to DFSMSrmm or have the INIT or ERASE action pending.

Note: As an alternative to IEHINITT, consider using the EDGINERS utility as described in [“Using DFSMSrmm” on page 245](#). EDGINERS checks security and volume ownership and provides auditing; IEHINITT does not.

Each volume label set created by the program contains:

- A standard volume label with a serial number you specify, owner identification, and a blank security byte. ISO/ANSI

Version 3 and Version 4 labels may contain an access code other than an ASCII space by using the ACCESS keyword. A label conforming to the ISO/ANSI Version 3 standard will be created if a Version 3 label is requested. A label conforming to the Version 4 standard will be constructed if a Version 4 label is requested.

A complete description of IBM standard volume labels and Version 3 and Version 4 volume labels is in [z/OS DFSMS Using Magnetic Tapes](#).

- An 80-byte dummy header label. For IBM standard labels, this record consists of the characters "HDR1" followed by character zeros. For Version 3 or Version 4 labels, this record consists of the characters "HDR1" followed by character zeros in the remaining positions, with the exception of:
 - Position 54, which will contain an ASCII space;
 - A "1" in the file section, file sequence, and generation number fields;
 - A leading space in the creation and expiration date fields;
 - A system code of "IBMZLA", which identifies the operating system creating the label, followed by 14 spaces.
- A tape mark.

When a labeled tape is subsequently used as a receiving volume, the OPEN or EOVS function:

1. Rewrites the volume label.
2. Rewrites the dummy HDR1 record created by IEHINITT with operating system data, device-dependent information, and data set information.
3. Writes HDR2 record, containing data set characteristics.
4. Writes user header labels if the user program provides exits to user label routines.
5. Writes a tape mark.
6. Positions the volume for the user program to write data.

Note for Version 3 and Version 4 Tape Labels:

For Version 3 there is no accessibility code checking done during IEHINITT processing, other than checking for uppercase A through Z in the ACCESS keyword. Therefore, it is possible to create a tape with a volume access code that the receiving operating system will not recognize. In such a situation, the tape would have to be reinitialized to contain an acceptable access code.

The set of valid Version 3 characters is:

```
upper case A--Z, numeric 0--9, and the special characters
| " % & ' ( ) * + , - . / : ; < = > ? space
```

The set of valid Version 4 characters is:

```
upper case A--Z, numeric 0--9, and the special characters
| " % & ' ( ) * + , - . / : ; < = > ? _ space
```

The only difference between the two lists of special characters is the _ (underscore).

If a Version 3 or Version 4 volume is initialized **only** with IEHINITT, the labels produced do not frame an empty (null) data set as required for interchange. In order to produce label symmetry described by the ISO/ANSI standards, at least a minimal Open/Close sequence must be processed. For example, a volume

initialized previously with IEHINITT will result in label symmetry if the data set utility IEBGENER is used before the volume leaves the system for interchange, as follows:

```
//STEP1 EXEC PGM=IEBGENER
//SYSPRINT DD DUMMY
//SYSUT1 DD DUMMY,DCB=(RECFM=F, BLKSIZE=80, LRECL=80)
//SYSUT2 DD DSN=DUMMY, UNIT=(tape), LABEL=(,AL), DISP=OLD,
// DCB=(RECFM=F, BLKSIZE=80, LRECL=80), VOL=SER=volser
//SYSIN DD DUMMY
```

Related reading :

- For more information, see [z/OS DFSMS Installation Exits](#).
- For a information about volume labels, see [z/OS DFSMS Using Magnetic Tapes](#).

Placing a Standard Label Set on Magnetic Tape

IEHINITT can be used to write BCDIC labels on 7-track tape volumes and EBCDIC or ASCII (ISO/ANSI format) labels on 9, 18, 36, 128, and 256 track volumes. Any number of tape cartridges or tape volumes can be labeled in a single execution of IEHINITT.

IEHINITT is supported in all IBM 3494 and 3495 Automated Tape Library Dataserver environments, as well as the 3495-M10 environment. Special considerations apply when using IEHINITT in the 3494 or 3495 Automated Tape Library Dataserver.

Multiple tape volumes initialized via a single INITT command are labeled in sequential order by specifying a serial number to be written on the first tape volume. The serial number must be specified as six numeric characters, and is incremented by 1 for each successive tape volume. If only one tape volume is to be labeled, the specified serial number can be either numeric or alphanumeric.

You can provide additional information, for example:

- owner name
- rewind or unload specifications
- format
- access code

You must supply all tapes to be labeled, and must include with each job request explicit instructions to the operator about where each tape is to be mounted.

IEHINITT writes 7-track tape labels in even parity (translator on, converter off).

Previously labeled tapes can be overwritten with new labels regardless of expiration date and security protection. SAF/RACF is invoked to determine the proper level of access to a tape volume for drives in a Tape Library Dataserver.

If errors are encountered while attempting to label a tape, the tape is left unlabeled. IEHINITT tries to label any tapes remaining to be processed.

Related reading:

- For information about see [“Tape Library Dataserver Considerations” on page 248](#).
- For information about creating routines to write standard or nonstandard labels, see [z/OS DFSMS Using Magnetic Tapes](#).

Using DFSMSrmm

The EDGINERS utility of DFSMSrmm is recommended instead of IEHINITT for labeling tapes that reside both inside and outside IBM TotalStorage Enterprise Automated Tape Library (3495)s for the following reasons:

1. You can label a set of volumes with DFSMSrmm.

2. DFSMSrmm ensures that the data sets on the volume have expired.
3. DFSMSrmm validates that the correct volume is mounted before creating the volume label.
4. DFSMSrmm can track that a volume needs to be labeled and can automate tape labeling using the information in its control data set.
5. DFSMSrmm also provides facilities for erasing the data on a tape when it expires.

Related reading: For more information about the EDGINERS utility, see [z/OS DFSMSrmm Implementation and Customization Guide](#).

Replacing the Key Encrypting Key Structure

You can use the REKEY function with the IEHINITT utility to replace the key encrypting key structure stored on a tape cartridge. This allows you to export a tape cartridge to your business partner without being required to share the private key and to rewrite the full tape. Instead, you can send the corresponding public key that allows your business partner to decrypt the tape on a different tape drive. You can also use the REKEY function to manage the security policy where the key encryption key periodically expires.

When you encrypt a tape cartridge, you also encrypt the data key, used to encrypt and decrypt the data on the tape. You can specify up to two key labels and then encrypt the data key with up to two different public keys. This generates an externally encrypted data key for each key label. The externally encrypted data key is stored in the memory of the tape cartridge in non-user data area. You can re-encrypt the data key using the new key labels, and then use one of them locally, and the other one offsite.

IEHINITT uses SAF/RACF for security checking in a library and non-library environment, at the volume level but not at the data set level. You must have UPDATE level of authority to use the REKEY function for CLAS=TAPEVOL.

Attention: Implement security protection to prevent unauthorized users from accessing or destroying data. The REKEY function might overwrite the existing key labels stored on tapes regardless of the expiration dates of the data set. The installation exits are notified about the changes to the key labels through the dynamic exits routines that do not provide security protection.

Input and Output

IEHINITT uses as input a control data set that contains the utility control statements. IEHINITT produces an output data set that contains:

- Utility program identification
- Initial volume label information for each successfully labeled tape volume
- Contents of utility control statements
- Any error messages

IEHINITT supports the use of uncaptured UCB addresses. Capturing and uncapturing is a process used by the operating system to convert 31 bit, above the 16 MB line addresses into 24 bit, below the line addresses. There are no special considerations that need to be made by the user if devices are allocated via JCL, the normal method of invoking IEHINITT. However, user written applications can dynamically allocate devices without capturing them and pass them to IEHINITT. IEHINITT sets return codes in register 15.

Related reading: For information about the IEHINITT return codes, see [Appendix A, “Invoking Utility Programs from an Application Program,”](#) on page 315.

Control

IEHINITT is controlled by job and utility control statements. The job control statements are used to process or load IEHINITT and to define data sets used and produced by IEHINITT. The utility control statement is used to specify applicable label information.

Job Control Statements

Table 43 on page 247 shows the job control statements for IEHINITT.

Table 43. IEHINITT job control statements

Statement	Use
JOB	Starts the job.
EXEC	Specifies the program name (PGM=IEHINITT) or, if the job control statements reside in a procedure library, the procedure name. The EXEC statement can include additional parameter information.
SYSPRINT DD	Defines a sequential data set for messages.
anyname DD	Defines a tape unit to be used in a labeling operation; more than one tape unit can be identified.
SYSIN DD	Defines the control data set. The control data set normally resides in the input stream; however, it can be defined as a member of a partitioned data set or PDSE or as a sequential data set outside the input stream.

EXEC Statement

The EXEC statement can include PARM information that specifies the number of lines to be printed between headings in the message data set. The EXEC statement can be coded as follows:

Label	Statement	Parameters
// [stepname]	EXEC	PGM=IEHINITT [, PARM= ' LINECNT=nn']

where:

PGM=IEHINITT

specifies that IEHINITT is the program you want to run.

PARM='LINECNT=nn'

specifies the number of lines per page to be printed in the SYSPRINT data set. If PARM is omitted, 60 lines are printed between headings.

If IEHINITT is called from another program, the line count option can be passed in a parameter list that is referred to by the *optionaddr* subparameter of the LINK or ATTACH macro instruction. In addition, a beginning page number can be passed in a 6-byte parameter list that is referred to by the *hdingaddr* subparameter of the LINK or ATTACH macro instruction. For a discussion of linkage conventions, refer to [Appendix A, “Invoking Utility Programs from an Application Program,”](#) on page 315.

SYSPRINT DD Statement

The SYSPRINT data set must have a logical record length of 121. It must consist of fixed-length records with an ISO/ANSI control character in the first byte of each record. Any blocking factor can be specified.

anyname DD Statement

The "anyname" DD statement is entered:

```
//anyname DD DCB=DEN=x,UNIT=(xxxx,n,DEFER)
```

The DEN parameter specifies the density at which the labels are written. The UNIT parameter specifies the device type, number of units to be used for the labeling operation, and deferred mounting. DEFER must be specified to delay mounting a tape until IEHINITT is ready to process it. If DEFER is not used, the volume specified in the JCL will be mounted by allocation. When IEHINITT gets control it will detect this mounted volume and will unconditionally demount the volume. This provides the operator an opportunity to verify that the correct tape is mounted.

The name "anyname" must be identical to a name specified in a utility control statement to relate the specified units to the utility control statement. You also can code DISP=(,PASS) to cause the tape to remain mounted on the drive for use in subsequent job steps.

Related reading:

- For more information about the DEN and UNIT parameters, see [z/OS MVS JCL Reference](#)
- For more information, see [“Example 8: Write on a tape following labeling without demounting and remounting” on page 254.](#)

Tape Library Dataserver Considerations

To run IEHINITT in a library dataserver environment you would use JCL similar to:

```
//anyname DD UNIT=(,DEFER),
//          VOLUME=SER=volser,
//          DISP=(OLD,KEEP)
```

'anyname' is the name on the INITT utility control statement.

'volser' is the volume serial number of a volume which meets all of the following requirements:

- The volume resides in the tape library containing the volumes to be initialized.
- The volume has the PRIVATE use attribute.
- The volume is recorded in the tape recording technology - either 18, 36, 128, or 256-track - that is to be used in labeling the tapes.
- The volume does not have the read compatible special attribute.
- The volume is NOT one of those that will be labeled.

Note: No unit name is required, provided that 'volser' is a library-resident volume.

Additional considerations for running IEHINITT in an Automated Tape Library Dataserver are:

- IEHINITT is not cognizant of library categories and will mount and initialize tapes in a Tape Library Dataserver without regard to whether they reside in a SCRATCH or PRIVATE category.
- Message IEC701D, the normal WTOR permission message issued to the tape/operator's console does not appear on the console when processing on drives in an Automated Tape Dataserver.

SYSIN DD Statement

The SYSIN data set must have a logical record length of 80. Any blocking factor up to a block size of 32720 can be specified.

Utility Control Statements

IEHINITT uses the utility control statements INITT and REKEY. Continuation requirements for utility control statements are described in [“Continuing utility control statements” on page 8.](#)

Any number of INITT and REKEY utility control statements can be included for a given execution of the program. Each INITT statement must be labeled with a ddname that identifies a DD statement in the input stream.

INITT Statement

The syntax of the INITT statement is:

Label	Statement	Parameters
<i>ddname</i>	INITT	SER= <i>serial number</i> [, DISP={REWIND UNLOAD}] [, OWNER=' <i>name</i> '] [, NUMBTAPE={ <i>n</i> 1}] [, LABTYPE=AL] [, VERSION={3 4 }] [, ACCESS= <i>c</i>]

where:

ddname

specifies the name that is identical to the ddname in the name field of the DD statement defining a tape unit(s). This name must begin in column 1 of the record which contains the INITT statement.

SER=*serial number*

specifies the volume serial number of the first or only tape to be labeled. Specify up to six characters. For IBM standard labeled (SL) tapes, the serial number cannot contain blanks, commas, apostrophes, equal signs, or special characters other than periods, hyphens, dollar signs, pound signs, and at signs ('@'). ISO/ANSI labeled tapes (AL) may contain any valid ISO/ANSI *a* type character as described under the OWNER keyword. However, if any nonalphanumeric character (including a period or a hyphen) is present, delimiting apostrophes must be included.

You cannot use a blank as the first character in a volume serial number.

If you set the NUMBTAPE keyword to a value greater than 1, then the volume serial number must be all numeric. If the volume serial number is alphanumeric, the last three digits must be numeric. If the volume serial number is all numeric, it is increased by one for each additional tape (serial number 999999 is increased to 000000). If the volume serial number is alphanumeric, only the last three digits are increased by one for each additional tape (serial number VOL999 is increased to VOL000).

NUMBTAPE={*n* | 1}

specifies the number of tapes to be processed. The value *n* represents a number from 1 to 255. If more than one tape is specified, the volume serial number of the first tape must be all numeric. If the volume serial number is specified as alphanumeric, the last three digits must be numeric.

DISP={REWIND|UNLOAD}

specifies if a tape is to be rewound or rewound and unloaded. Tapes in a Tape Library Dataserver are unconditionally unloaded regardless of the specification for this parameter. These values can be coded:

REWIND

specifies that a tape is to be rewound (but not unloaded) after the label has been written.

UNLOAD

specifies that a tape is to be rewound and unloaded after the label has been written. This is the default.

OWNER='*name*'

specifies the owner's name or similar identification. The information is specified as character constants, and can be up to 10 bytes in length for EBCDIC and BCD volume labels, or up to 14 bytes in length for volume labels written in ASCII. The delimiting apostrophes must be present if

blanks, commas, apostrophes, equal signs, or other special characters (except periods or hyphens) are included. The set of valid ISO/ANSI *a* type characters for ASCII tapes is as follows: upper case A-Z, numeric 0-9, and special characters **!*"%&'()*+,-./:;<=>?**. The set of valid EBCDIC characters is as follows: uppercase A-Z, numeric 0-9, and special characters **¢ . < > (+ ! (X'6A') | (X'4F') ' & ! \$ *) ; ^ - \ / , % _ ? ' : # @ ' = " ~ { } **.

If an apostrophe is included within the OWNER name field, it must be written as two consecutive apostrophes.

The OWNER keyword can be specified for Version 3 or Version 4 tapes. If Version 4 is specified, any ISO/ANSI *a* type character can be used.

NUMTAPE={*n*|1}

specifies the number of tapes to be labeled according to the specifications made in this control statement. The value *n* represents a number from 1 to 255. If more than one tape is specified, the volume serial number of the first tape must be numeric.

LABTYPE=AL

When LABTYPE=AL is specified in the INITT statement, IEHINITT initializes tapes to conform to the Version 3 standard or the Version 4 standard as specified in the VERSION keyword. The format of the VERSION keyword follows.

Default: The tape is written in EBCDIC in IBM standard format for tape cartridges or 9-track tape volumes and in BCDIC for 7-track tape volumes.

VERSION={3|4}

When LABTYPE=AL is specified, the VERSION keyword determines if the format will be Version 3 or Version 4.

3

Initializes the tape to be Version 3

4

Initializes the tape to be Version 4

The volume label (VOL 1) is written the same for Version 3 and Version 4 with one exception. Field *Label Standard Version* will be 3 for Version 3 and 4 for Version 4. There will be no difference in the initialization of the header labels.

Default: If this keyword is specified, that value is used. If not, the installation version level in the DEVSUPxx PARMLIB member is used. If that is not specified, Version 3 is the default.

ACCESS=c

specifies the Version 3 and Version 4 volume accessibility code. Version 3 valid values for *c* are uppercase A through Z only. The default value is a blank character, indicating unlimited access to the volume. For Version 3, you **cannot** specify a blank character for the access code; it must be allowed to default. Version 4 valid values are any ISO/ANSI *a* type characters as follows: upper case A-Z, numeric 0-9, and special characters **!*"%&'()*+,-./:;<=>?**.

The Volume Access installation exit routine must be modified to allow subsequent use of the volume if ACCESS is specified.

You must specify LABTYPE=AL and VERSION if you specify ACCESS.

The ACCESS keyword can be specified for Version 3 or Version 4. If Version 4 is specified, the keyword can be any ISO/ANSI *a* type character.

Related reading: For more information about volume accessibility and ISO/ANSI installation exits, see [*z/OS DFSMS Using Magnetic Tapes*](#).

REKEY Statement

The syntax of the REKEY statement is:

Label	Statement	Parameters
<i>ddname</i>	REKEY	SER= <i>volser</i> [, DISP={REWIND UNLOAD}] [, NUMBTAPE={ <i>n</i> 1}] [, KEYLABL1= <i>keylabel1</i>] [, KEYENCD1={L H}] [, KEYLABL2= <i>keylabel2</i>] [, KEYENCD2={L H}]

where:

KEYLABL1=*keylabel1_name*

specifies the first key label for the key encryption key used by the Encryption Key Manager. To specify the key label value, you can use up to 64 characters with blanks padding the field after the characters (on the right). The characters can be alphanumeric, national, or special characters with some additional characters also allowed. If the label contains blanks or special characters, you must enclose it in single quotation marks. IEHINITT does not validate the character set that you specify for the key label keyword.

KEYLABL2=*keylabel2_name*

specifies the second key label for the key encryption key used by the Encryption Key Manager.

KEYENCD1={L | H}

specifies the encoding mechanism of the first key label.

- **L** = encoded as the specified label
- **H** = encoded as a hash of the public key

KEYENCD2={L | H}

specifies the encoding mechanism of the second key label.

Attention: You must specify at least one key label and the corresponding key encoding mechanism. You can specify the same value for both key labels. If you specify only one key label, the Encryption Key Manager uses the same value for the second key label. You can use the same set of key labels for a list of volumes, or you can specify different key labels and corresponding encoding mechanisms for each volume.

IEHINITT Examples

These examples illustrate some of the uses of IEHINITT. You can use [Table 44 on page 251](#) as a quick-reference guide to IEHINITT examples.

Table 44. IEHINITT example directory

Operation	Comments	Example
LABEL	Three 9-track tapes are to be labeled.	“Example 1: Write EBCDIC Labels on Three Tapes” on page 252
LABEL	A 9-track tape is to be labeled in ISO/ANSI.	“Example 2: Write an ISO/ANSI Label on a Tape” on page 252
LABEL	Two groups of 9-track tape volumes are to be labeled.	“Example 3: Place Two Groups of Serial Numbers on Six Tape Volumes” on page 253
LABEL	9-track tape volumes are to be labeled. Sequence numbers are to be incremented by 10.	“Example 4: Place Serial Number on Eight Tape Volumes” on page 253

Table 44. IEHINITT example directory (continued)

Operation	Comments	Example
LABEL	Three 9-track tape volumes are to be labeled. An alphanumeric label is to be placed on a tape volume; numeric labels are placed on the remaining two tape volumes.	“Example 5: Write EBCDIC Labels in Different Densities” on page 253
LABEL	Two 9-track tape volumes are to be labeled. The first volume is labeled at a density of 6250 bpi; the second at a density of 1600 bpi.	“Example 6: Write Serial Numbers on Tape Volumes at Two Densities” on page 254
LABEL	A 9-track tape volume is labeled in ISO/ANSI format with a nonblank access code.	“Example 7: Write an ISO/ANSI Label with an Access Code” on page 254

Examples that use **tape** in place of actual device numbers must be changed before use. The actual device numbers depend on how your installation has defined the devices to your system.

Example 1: Write EBCDIC Labels on Three Tapes

In this example, serial numbers 001234, 001235 and 001236 are placed on three tape volumes. The labels are written in EBCDIC at 800 bits per inch. Each volume labeled is mounted, when it is required, on a single 9-track tape unit.

```
//LABEL1  JOB  ...
//STEP1   EXEC PGM=IEHINITT
//SYSPRINT DD  SYSOUT=A
//LABEL   DD  DCB=DEN=2,UNIT=(tape,1,DEFER)
//SYSIN    DD  *
LABEL     INITT  SER=001234,NUMBTAPE=3
/*
```

The control statements are as follows:

- LABEL DD defines the tape unit used in the labeling operation.
- SYSIN DD defines the control data set, which follows in the input stream.
- LABEL INITT specifies the number of tapes to be labeled (3), beginning with 001234.

Example 2: Write an ISO/ANSI Label on a Tape

In this example, serial number 001001 is placed on one ISO/ANSI tape volume; the label is written at 800 bits per inch. The volume labeled is mounted, when it is required, on a 9-track tape unit.

```
//LABEL2  JOB  ...
//STEP1   EXEC PGM=IEHINITT
//SYSPRINT DD  SYSOUT=A
//ASCIIILAB DD  DCB=DEN=2,UNIT=(tape,1,DEFER)
//SYSIN    DD  *
ASCIIILAB INITT  SER=001001,OWNER='SAM A. BROWN',LABTYPE=AL
/*
```

The control statements are as follows:

- ASCIIILAB DD defines the tape volume to be used in the labeling operation.
- SYSIN DD defines the control data set, which follows in the input stream.
- ASCIIILAB INITT specifies the serial number, owner ID and label type for the volume. Because the VERSION keyword was not specified, the ISO/ANSI tape will be created based on what is specified in the DEVSUPxx parmlib member or as a version 3 by default.

Example 3: Place Two Groups of Serial Numbers on Six Tape Volumes

In this example, two groups of serial numbers (001234, 001235, 001236, and 001334, 001335, 001336) are placed on six tape volumes. The labels are written in EBCDIC at 800 bits per inch. Each volume labeled is mounted, when it is required, on a single 9-track tape unit.

```
//LABEL3 JOB ...
//STEP1 EXEC PGM=IEHINITT
//SYSPRINT DD SYSOUT=A
//LABEL DD DCB=DEN=2,UNIT=(tape,1,DEFER)
//SYSIN DD *
LABEL INITT SER=001234,NUMBTAPE=3
LABEL INITT SER=001334,NUMBTAPE=3
/*
```

The control statements are as follows:

- LABEL DD defines the tape unit to be used in the labeling operation.
- SYSIN DD defines the control data set, which follows in the input stream.
- LABEL INITT defines the two groups of serial numbers to be put on six tape volumes.

Example 4: Place Serial Number on Eight Tape Volumes

In this example, serial numbers 001234, 001244, 001254, 001264, 001274, and so forth, are placed on eight tape volumes. The labels are written in EBCDIC at 800 bits per inch. Each volume labeled is mounted, when it is required, on one of four 9-track tape units.

```
//LABEL4 JOB ...
//STEP1 EXEC PGM=IEHINITT
//SYSPRINT DD SYSOUT=A
//LABEL DD DCB=DEN=2,UNIT=(tape,4,DEFER)
//SYSIN DD *
LABEL INITT SER=001234
LABEL INITT SER=001244
LABEL INITT SER=001254
LABEL INITT SER=001264
LABEL INITT SER=001274
LABEL INITT SER=001284
LABEL INITT SER=001294
LABEL INITT SER=001304
/*
```

The control statements are as follows:

- LABEL DD defines the tape unit used in the labeling operation.
- SYSIN DD defines the control data set, which follows in the input stream.
- The LABEL INITT statements define the tapes to be labeled by volume serial number.

Example 5: Write EBCDIC Labels in Different Densities

In this example, serial number TAPE1 is placed on a tape volume, and serial numbers 001234 and 001235 are placed on two tape volumes. The labels are written in EBCDIC at 800 and 1600 bits per inch, respectively.

```
//LABEL5 JOB ...
//STEP1 EXEC PGM=IEHINITT
//SYSPRINT DD SYSOUT=A
//LABEL1 DD DCB=DEN=2,UNIT=(tape,1,DEFER)
//LABEL2 DD DCB=DEN=3,UNIT=(tape,1,DEFER)
//SYSIN DD *
LABEL1 INITT SER=TAPE1
LABEL2 INITT SER=001234,NUMBTAPE=2
/*
```

The control statements are as follows:

- LABEL1 DD and LABEL2 DD define two tape volumes to be used in the labeling operation.

- SYSIN DD defines the control data set, which follows in the input stream.
- LABEL1 INITT places the serial number TAPE1 on the tape volume defined in LABEL1 DD. LABEL2 INITT places the serial numbers 001234 and 001235 on the tape volume defined in LABEL2 DD.

Example 6: Write Serial Numbers on Tape Volumes at Two Densities

In this example, the serial number 006250 is written in EBCDIC on a tape volume at a density of 6250 bpi, and the serial number 001600 is written in EBCDIC on a second volume at a density of 1600 bpi.

```
//LABEL6 JOB ...
//STEP1 EXEC PGM=IEHINITT
//SYSPRINT DD SYSOUT=A
//DDFIRST DD DCB=DEN=4,UNIT=(tape,1,DEFER)
//DDSECOND DD DCB=DEN=3,UNIT=(tape,1,DEFER)
//SYSIN DD *
DDFIRST INITT SER=006250
DDSECOND INITT SER=001600
/*
```

The control statements are as follows:

- DDFIRST DD defines the first tape volume to be used.
- DDSECOND DD defines the second tape volume to be used.
- SYSIN DD defines the control data set, which follows in the input stream.
- DDFIRST INITT writes the serial number 006250 on the volume defined in DDFIRST DD. DDSECOND INITT writes the serial number 001600 on the volume defined in DDSECOND DD.

Example 7: Write an ISO/ANSI Label with an Access Code

In this example, a version 4 ISO/ANSI (AL) labeled tape is created with a nonblank access code. The volume serial number is TAPE01.

```
//LABEL7 JOB ...
//STEP1 EXEC PGM=IEHINITT
//SYSPRINT DD SYSOUT=A
//LABEL DD UNIT=(tape,1,DEFER),DCB=DEN=4
//SYSIN DD *
LABEL INITT SER=TAPE01,OWNER=TAPOWNER,LABTYPE=AL,ACCESS=A,VERSION 4
/*
```

The control statements are as follows:

- LABEL DD defines the device on which the tape is mounted.
- SYSIN DD defines the control data set, which follows in the input stream.
- The INITT statement creates a Version 4 ISO/ANSI label for the tape with volume serial number TAPE01, owned by TAPOWNER. The ACCESS code is specified as "A", and the operating system that receives this volume must be able to recognize the "A" in order for the volume to be accepted.

Example 8: Write on a tape following labeling without demounting and remounting

In this example, you can label a tape in one step of a job, and then, without the system demounting and remounting that tape between steps, write to the tape in a subsequent step of the same job. The necessary JCL code follows:

```
//STEP1 EXEC PGM=IEHINITT
//SYSPRINT DD SYSOUT=*
(1) //LABEL DD UNIT=(tape,1,DEFER),DISP=(,PASS)
//SYSIN DD *
(2) LABEL INITT SER=serial,DISP=REWIND
//*
//STEP2 EXEC PGM=user_program
```

```

//INPUT DD DSN=input_dsn,DISP=SHR
//OUTPUT DD DSN=dsname,DISP=(NEW,CATLG),
//          DCB=(dcbinfo),
(3) //          UNIT=tape,VOL=(,RETAIN,SER=serial)
Notes:
(1) Either DISP=(NEW,PASS) or VOL=(,RETAIN) must be specified.
(2) DISP=REWIND must be specified on the INITT statement.
(3) VOL=SER=serial must be specified.
    VOL=REF=*.STEP1.LABEL will not work.

```

Example 9: Rekey one tape volume

In this example, one tape volume SL1000 is to be rekeyed with two key labels.

```

//TAPEJOB JOB ...
//STEP1 EXEC PGM=IEHINITT
//SYSPRINT DD SYSOUT=A
//REKEY1 DD UNIT=(TAPE,1,DEFER)
//SYSIN DD *
REKEY1 REKEY SER=SL1000,
        KEYLABL1=firstkeylabel,KEYENCD1=L,
        KEYLABL2=secondkeylabel,KEYENCD2=H
/*

```

Example 10: Multiple tapes specified for SER keyword

In this example, five tape volumes are to be rekeyed with the same key labels.

```

//TAPEJOB JOB ...
//STEP1 EXEC PGM=IEHINITT
//SYSPRINT DD SYSOUT=A
//REKEY2 DD UNIT=(TAPE,1,DEFER)
//SYSIN DD *
REKEY2 REKEY SER=( AL1000,AL2000,AL3000,AL4000,AL5000),
        KEYLABL1=firstkeylabel,KEYENCD1=L,
        KEYLABL2=secondkeylabel,KEYENCD2=H
/*

```

Example 11: Three tapes with NUMBTAPE specified

In this example, three tape volumes 001200, 001201, and 001202 are to be rekeyed with the same key labels.

```

//TAPEJOB JOB ...
//STEP1 EXEC PGM=IEHINITT
//SYSPRINT DD SYSOUT=A
//REKEY3 DD UNIT=(TAPE,1,DEFER)
//SYSIN DD *
REKEY3 REKEY SER=001200,NUMBTAPE=3,
        KEYLABL1=firstkeylabel,KEYENCD1=L,
        KEYLABL2=secondkeylabel,KEYENCD2=H
/*

```

Example 12: Multiple REKEY Control Statements

In this example, two tape volumes, 005000, and SL8000, are to be rekeyed. Each tape volume is to be rekeyed with different set of key labels and encoding mechanisms.

```

//TAPEJOB JOB ...
//STEP1 EXEC PGM=IEHINITT
//SYSPRINT DD SYSOUT=A
//REKEY4 DD UNIT=(TAPE,1,DEFER)
//REKEY5 DD UNIT=(TAPE)
//SYSIN DD *
REKEY4 REKEY SER=005000,
KEYLABL1=firstkeylabel,KEYENCD1=L,
        KEYLABL2=secondkeylabel,KEYENCD2=H

```

```

REKEY5    REKEY    SER=SL8000,
           KEYLABL1=differentfirstkeylabel,KEYENCD1=L,
           KEYLABL2=differentsecondkeylabel,KEYENCD2=H
/*

```

Example 13: Multiple Control Statements with NUMBTAPE

In this example, three tape volumes 001200, 001201, and 001202 are to be rekeyed with the same key labels specified for the first REKEY control statement with NUMBTAPE specified. The second control statement specifies one tape SL8000 to be rekeyed with different key labels. Note, that KEYLABL2 and its associated KEYENCD2 are omitted, thus the same key label and encoding mechanism values specified for the KEYLABL1 and the KEYENCD1 keywords are used for both key labels.

```

//TAPEJOB   JOB      ...
//STEP1     EXEC     PGM=IEHINITT
//SYSPRINT   DD      SYSOUT=A
//REKEY6     DD      UNIT=(TAPE,1,DEFER)
//REKEY7     DD      UNIT=(TAPE)
//SYSIN      DD      *
REKEY6      REKEY    SER=001200,NUMBTAPE=3,
              KEYLABL1=firstkeylabel,KEYENCD1=L,
              KEYLABL2=secondkeylabel,KEYENCD2=H
REKEY7      REKEY    SER=SL8000,
              KEYLABL1=differentfirstkeylabel,KEYENCD1=L,
/*

```

Example 14: Printout of INITT Statement Specifications and Initial Volume Label Information

In this example, a printout appears of a message data set including the INITT statement and initial volume label information. One INITT statement was used to place serial numbers 001122 and 001123 on two standard label tape volumes. VOL1001122 and VOL1001123 are interpreted as follows:

- VOL1 indicates that an initial volume label was successfully written to a tape volume.
- 001122 and 001123 are the serial numbers that were written onto the volumes.
- A blank space following the serial number represents the Volume Security field, which is not used during OPEN/CLOSE/EOV processing on a standard label tape.

No errors occurred during processing.

```

SYSTEM SUPPORT UTILITIES   IEHINITT

ALL  INITT  SER=001122,NUMBTAPE=2,OWNER='P.T.BROWN',DISP=REWIND

VOL1001122          P.T.BROWN
VOL1001123          P.T.BROWN

```

Chapter 14. IEHLIST (List System Data) Program

IEHLIST is a system utility used to list entries in the directory of one or more partitioned data sets or PDSEs, or entries in an indexed or non-indexed volume table of contents. Any number of listings can be requested in a single execution of the program. For an example of a VTOC listing produced by IEHLIST and a detailed explanation of the fields in the listing, see [Appendix D, “IEHLIST VTOC Listing,”](#) on page 351.

Listing a Partitioned Data Set or PDSE Directory

IEHLIST can list up to 10 partitioned data set or PDSE directories at a time.

The directory of a partitioned data set is composed of variable-length records blocked into 256-byte blocks. Each directory block can contain one or more entries that reflect member or alias names and other attributes of the partitioned members. IEHLIST can list these blocks in edited and unedited format.

The directory of a PDSE, when listed, will have the same format as the directory of a partitioned data set.

Edited Format

The edited format of a partitioned data set directory is meant to be used with module libraries. Most of the information given in an edited listing is meaningful only for load modules.

If you request an edited listing of a partitioned data set or PDSE whose members are not load modules, you will get an edited listing. In that case, the listing will contain information about your data set that will not necessarily be correct. Only request an edited listing of a data set whose members are load modules.

When you request an edited format of a module library, IEHLIST provides the following information:

- Member name
- Entry point
- Relative address of start of member
- Relative address of start of text
- Contiguous virtual storage requirements
- Length of first block of text
- Origin of first block of text
- System status indicators
- Linkage editor attributes
- APF authorization required
- Other information.

Edited entry for a partitioned member

The following example shows an edited entry for a partitioned member, LOADMOD. The entry is shown as it is listed by the IEHLIST program. This figure is only an example of a listing of a partitioned data set or PDSE directory. Your actual edited listing produced by the IEHLIST program may differ.

```

OTHER INFORMATION INDEX
SCATTER FORMAT  SCRT=SCATTER/TRANSLATION TABLE TTR IN HEX, LEN OF SCTR LIST IN DEC, LEN OF
TRANS TABLE IN DEC,      ESDID OF FIRST TEXT RCD IN DEC, ESDID OF CSECT CONTAINING ENTRY POINT IN
DEC
OVERLAY FORMAT  ONLY=NOTE LIST RCD TTR IN HEX, NUMBER OF ENTRIES IN NOTE LIST RCD IN DEC
ALIAS NAMES     ALIAS MEMBER NAMES WILL BE FOLLOWED BY AN ASTERISK IN THE PDS FORMAT LISTING
ATTRIBUTE INDEX
```

IEHLIST

BIT ON	ON	OFF OFF	BIT	ON	OFF	BIT	ON	OFF	BIT
0 EDIT	RENT EDIT	NOT RENT	4	OL	NOT OL	8	NOT DC	DC	12 NOT
1 SYMS	REUS NO SYMS		5	SCTR	BLOCK	9	ZERO ORG	NOT ZERO	13
2 LEVEL	ONLY E LEVEL	NOT ONLY	6	EXEC	NOT EXEC	10	EP ZERO	NOT ZERO	14 F
3 REFR	TEST TEST	NOT REFR	7	1 TXT	MULTI RCD	11	NO RLD	RLD	15
MEMBER AUTH NAME REQ	ENTRY OTHER PT-HEX REQ	ATTR HEX INFORMATION	REL BEGIN	ADDR-HEX 1ST TXT	CONTIG STOR-DEC	LEN 1ST TXT-DEC	1ST ORG	1ST TXT-HEX	SST INFO VS ATTR
LOADMOD NO	000000 SCTR=000000	06E2	000004	00020F	000166248	0927			ABSENT 880000
	00484,01084,32,32								
OF THE 00002 DIRECTORY BLOCKS ALLOCATED TO THIS PDS, 00001 ARE(IS) COMPLETELY UNUSED									

Before printing the directory entries on the first page, an index is printed explaining the asterisk (*), if any, following a member name, the attributes (fields 3 and 10), and other information (field 12). Under OTHER INFORMATION INDEX, scatter and overlay format data is described positionally as it appears in the listing; under the ATTRIBUTE INDEX, the meaning of each attribute bit is explained. There is no index for the VS ATTR field. The data displayed in this field is from the PDS2FTBO field in the PDS directory.

Each directory entry occupies one printed line, except when the member name is an alias and the main member name and associated entry point appear in the user data field. When this occurs, two lines are used and every alias is followed by an asterisk. If the main member is renamed, the old member name will still be in the alias directory entry and consequently printed on the second line.

Unedited (Dump) Format

The unedited formatted listing produced by IEHLIST can be used to list the directories of any type of partitioned data set or PDSE. The directories of partitioned data sets or PDSEs whose members were not produced by the linkage editor are best listed using the unedited format. In an unedited listing, each member is listed separately and in hexadecimal.

Unedited listing of a partitioned data set

The following example shows the format of an unedited listing of a three-member partitioned data set or PDSE directory. The figure displays only an example of an unedited formatted listing produced by IEHLIST program. Your actual unedited listing by IEHLIST program may differ.

MEMBERS	TTRC	VARIABLE	USER DATA	---(USER DATA AND TTRC ARE IN HEX)		
MEMBER1	0009150F	0100000000	89135F0089	135F101300	5C005C0000	D1C1D9C5C4
MEMBER2	000E010F	0100000180	92217F0092	217F163900	1300130000	C9C2D4E4E2
MEMBER3	000D0B0F	0100000000	91194F0091	194F125100	1100110000	D1C1D9C5C4

To correctly interpret user data information, you must know the format of the partitioned entry. The formats of directory entries are discussed in *z/OS DFSMS Using Data Sets*.

Related reading: For a listing of a partitioned data set or PDSE directory, see [“Unedited listing of a partitioned data set”](#) on page 258 or Chapter 11, “IEBPTPCH (Print-Punch) Program,” on page 203.

Listing a Volume Table of Contents

IEHLIST can be used to list, partially or completely, entries in a specified volume table of contents (VTOC), whether indexed or non-indexed. The program lists the contents of selected data set control blocks (DSCBs) in edited or unedited form.

Related reading: For more information on indexed VTOCs, see *z/OS DFSMSdfp Advanced Services*.

Edited Format

Two edited formats are available.

First Edited Format

The first edited format is a comprehensive listing of the DSCBs in the VTOC. It provides the status and attributes of the volume, and describes in depth the data sets residing on the volume. This listing includes:

- Logical record length and block size
- Initial and secondary allocations
- Upper and lower limits of extents
- Alternate track information
- Available space information, including totals of unallocated cylinders, unallocated tracks, and unallocated (Format 0) DSCBs
- Option codes (printed as two hexadecimal digits)
- Record formats
- SMS indicators

The first DSCB on your listing is always a VTOC (Format 4) DSCB. It defines the scope of the VTOC itself; that is, it contains information about the VTOC and the volume rather than the data sets referenced by the VTOC.

Indexed VTOCs

For indexed VTOCs, there are two types of formatted listings. These types are specified using the INDEXDSN parameter.

If INDEXDSN is omitted, the listing contains:

- A statement of the number of levels in the index, if enabled.
- A formatted Format 4 DSCB.
- Formatted data set entries in alphanumeric order (Format 1 and Format 8 DSCB physical-sequential order if the index is disabled).
- Formatted freespace information.
- Totals of unallocated cylinders, unallocated tracks, unallocated (Format 0) DSCBs, and unallocated VIRs.

If INDEXDSN=*name* is specified, the listing contains, in addition to the preceding items:

- Formatted space and allocation information.
- Allocated VTOC index entry records, formatted and listed by level and key sequence within level (in physical-sequential order if the index is disabled).
- If the VTOC index is disabled, a statement is included to this effect.

Related reading: For a sample of the first edited format illustrating how each DSCB will appear in the listing, see [Appendix D, “IEHLIST VTOC Listing,” on page 351](#).

Second Edited Format

The second edited format is an abbreviated description of the data sets. It is provided by default when no format is specifically requested. It provides the following information:

- Data set name
- Creation date (yyyy.ddd)
- Expiration date (yyyy.ddd)

- Password indication
- Organization of the data set
- Extents
- Volume serial number
- SMS indicators

The last line in the listing indicates how much space remains in the VTOC.

For non-indexed VTOCs, data set entries are listed in physical-sequential order. Totals of unallocated cylinders, unallocated tracks, and unallocated (Format 0) DSCBs are also listed.

For indexed VTOCs, this listing contains:

- A statement of the number of levels in the index.
- Data set entries listed in alphanumeric order.
- Totals of unallocated cylinders, unallocated tracks, unallocated (Format 0) DSCBs, and unallocated VIRs.
- SMS indicators.

Unedited (Dump) Format

This option produces a complete hexadecimal listing of the DSCBs in the VTOC. The listing is in an unedited *dump* form, requiring you to know the various formats of applicable DSCBs. The VTOC overlay for IEHLIST listings of VTOCs in dump format is useful in identifying the fields of the DSCBs.

For non-indexed VTOCs, this listing contains:

- DSCBs dumped, in physical-sequential order.
- Totals of unallocated cylinders, unallocated tracks, and unallocated (Format 0) DSCBs.

For indexed VTOCs there are two types of dump listings. These types are specified using the INDEXDSN parameter.

If INDEXDSN is omitted, the listing contains:

- DSCBs dumped in physical-sequential order.
 - If the device has 64K or less tracks, then one token Format 5 DSCB is identified.
 - If the device has more than 64K tracks, then both a token Format 5 DSCB and a token Format 7 DSCB are identified.
- Unformatted free space information.
- Totals of unallocated cylinders, unallocated tracks, unallocated (Format 0) DSCBs, and unallocated VIRs.

If INDEXDSN=*name* is specified, the listing contains, in addition to the preceding items:

- A dump of the space and allocation information.
 - If the VTOC index is disabled, both allocated and unallocated records are dumped in physical-sequential order.
- If the VTOC index is disabled, a statement is included to this effect.

Note: For large volumes, the number of available DSCBs has a maximum value of 65 499 for indexed VTOCs while the summary section shows the correct blank DSCBs.

Related reading: For information about the various formats that data set control blocks can assume, see [*z/OS DFSMSdfp Advanced Services*](#).

Input and Output

IEHLIST uses the following input:

- One or more source data sets that contain the data to be listed. The input data set can be:
 - A VTOC
 - A partitioned data set or PDSE
- A control data set, that contains utility control statements that are used to control the functions of IEHLIST.

IEHLIST produces as output a message data set that contains the result of the IEHLIST operations. The message data set includes the listed data and any error messages.

If IEHLIST is invoked from an application program, you can dynamically allocate the devices and data sets by issuing SVC 99 before calling IEHLIST.

Related reading: For information about IEHLIST return codes, see [Appendix A, “Invoking Utility Programs from an Application Program,”](#) on page 315.

Control

IEHLIST is controlled by job and utility control statements. The job control statements are used to process or load IEHLIST and to define the data sets used and produced by IEHLIST.

Utility control statements are used to control the functions of the program and to define those data sets or volumes to be modified.

Job Control Statements

Table 45 on page 261 shows the job control statements for IEHLIST.

Table 45. IEHLIST job control statements	
Statement	Use
JOB	Starts the job.
EXEC	Specifies the program name (PGM=IEHLIST) or, if the job control statements reside in a procedure library, the procedure name. Additional PARM information can be specified to control the number of lines printed per page.
SYSPRINT DD	Defines a sequential data set for messages.
anyname DD	Defines a permanently mounted or mountable volume.
SYSIN DD	Defines the control data set. The control data set normally follows the job control language in the input stream; however, it can be defined as an unblocked sequential data set or member of a procedure library.

IEHLIST specifications do not allow for serialization of the object being listed. If another program updates a block of the data set just prior to IEHLIST reading the data set, a message (IEH105I or IEH114I) can be issued and the output produced by IEHLIST can be incorrect. If another program deletes a data set while IEHLIST attempts to access it during LISTVTOC processing, message IEH102I can be issued and the output produced by IEHLIST can be incomplete. If this happens, rerun the job.

EXEC Statement

You can control the number of lines IEHLIST will print per page of output using the PARM parameter on the EXEC statement. The EXEC statement can be coded:

Label	Statement	Parameters
// [stepname]	EXEC	PGM=IEHLIST [, PARM= 'LINECNT=xx']

where:

PGM=IEHLIST

specifies that IEHLIST is the program you want to run.

PARM='LINECNT=xx'

specifies the number of lines, xx, to be printed per page. The number specified by xx must be a decimal number from 01 to 99. If LINECNT is not specified, 58 lines are printed per page.

The PARM field cannot contain embedded blanks, zeros, or any other PARM keywords if LINECNT is specified.

SYSPRINT DD Statement

The block size for SYSPRINT must be a multiple of 121. Any blocking factor can be specified for this block size.

anyname DD Statement

A DD statement must be included for each permanently mounted or mountable volume referred to in the job step. These DD statements are used to allocate devices: they are not true data definition statements. Concatenated DD statements are not allowed.

Because IEHLIST modifies the internal control blocks created by device allocation DD statements, these DD statements must not include the DSNNAME parameter. (All data sets are defined explicitly or implicitly by utility control statements.)

The DD statement can be entered:

```
//anyname DD UNIT=xxxx,VOLUME=SER=xxxxxx
,DISP=OLD
```

The UNIT and VOLUME=SER parameters define the device type and volume serial number without requiring that a real data set be allocated on the volume.

SYSIN DD Statement

The block size for SYSIN must be a multiple of 80. Any blocking factor can be specified for this block size. You can concatenate DD statements for SYSIN.

Utility Control Statements

Table 46 on page 262 shows the utility control statements for IEHLIST.

Table 46. IEHLIST utility control statements	
Statement	Use
LISTPDS	Requests a directory listing of one or more partitioned data sets or PDSEs.
LISTVTOC	Requests a listing of all or part of a volume table of contents.

Continuation requirements for utility control statements are described in [“Continuing utility control statements”](#) on page 8.

LISTPDS Statement

The LISTPDS statement is used to request a directory listing of one or more partitioned data sets or PDSEs that reside on the same volume.

The FORMAT option of the LISTPDS statement may be used only on a partitioned data set whose members have been created by the linkage editor. If you try to use FORMAT with a PDSE or partitioned data set whose members are not load modules, the listing will contain undependable information.

The syntax of the LISTPDS statement is:

Label	Statement	Parameters
[<i>label</i>]	LISTPDS	DSNAME=(<i>name</i> [, <i>name</i> [,...]]) [,VOL= <i>device=serial</i>] [, { <u>DUMP</u> <u>FORMAT</u> }]

where:

DSNAME=(*name*[,*name*[,...]])

specifies the fully qualified names of the partitioned data sets or PDSEs whose directories are to be listed. A maximum of 10 names is allowed. If the list consists of only a single name, the parentheses can be omitted.

VOL=*device=serial*

specifies the device type and volume serial number of the volume on which the partitioned data set or PDSE directory resides. If the partitioned data set or PDSE is not on the system residence volume, the VOL parameter is required.

DUMP

specifies that the listing is to be in unedited, hexadecimal form. DUMP is the default.

FORMAT

specifies that the listing is to be edited for each directory entry.

The FORMAT option may be used only on a partitioned data set whose members have been created by the linkage editor. Members that have not been created by the linkage editor cause their directory entries to be listed in unedited (DUMP) format.

Related reading: For more information, see [“Listing a Partitioned Data Set or PDSE Directory”](#) on page 257.

LISTVTOC Statement

The LISTVTOC statement is used to request a partial or complete listing of the entries in a specified volume table of contents.

If you are using IEHLIST to list both the VTOC and the index data set of an indexed VTOC, refer to [“Listing a Volume Table of Contents”](#) on page 258.

The syntax of the LISTVTOC statement is:

Label	Statement	Parameters
[<i>label</i>]	LISTVTOC	[{ <u>DUMP</u> <u>FORMAT</u> [, <u>PDSESPACE</u>] }] [, INDEXDSN=SYS1.VTOCIX.xxxx] [, DATE={ <i>dddy</i> <i>ddyyyy</i> }] [,VOL= <i>device=serial</i>] [, DSNAME=(<i>name</i> [, <i>name</i> [,...]])

where:

DUMP

specifies that the listing is to be in unedited, hexadecimal form. The dump option will show SMS indicators for the volume and DSCBs in the VTOC (format 1, 2, 3, 4, 5, 8 and 9 DSCBs) and the VTOC index in hexadecimal format.

FORMAT [PDSESPACE]

specifies that a comprehensive edited listing is to be generated.

When PDSESPACE is specified, the space allocated, and space used, is displayed. This display occurs only for PDSE data sets, and is in kilobytes (kbytes).

When the space used amount is greater than the space allocated, the PDSE needs to be validated for possible corruption.

If both FORMAT and DUMP are omitted, an abbreviated edited format is generated.

INDEXDSN=SYS1.VTOCIX.xxxx

specifies that index information is to be listed, in addition to the VTOC. The value xxxx is any third level qualifier. DUMP or FORMAT must be specified if INDEXDSN is specified. For more information on indexed VTOCs, refer to [“Listing a Volume Table of Contents”](#) on page 258.

DATE={dddy|dddyyy}

specifies that each entry that expires before this date is to be flagged with an asterisk (*) after the entry name in the listing. This parameter applies only to the abbreviated edited format. The specification of dddy is the same as ddd19yy.

The VTOC expiration date of 'never expire' (1999.365 or 1999.366) is treated in a special way. That is, even if the DATE= parameter specifies a year date after 1999, those expiration dates are not flagged with an asterisk (*). The 'never expire' expiration date will never be flagged with an asterisk with any DATE= specification.

The DATE= parameter specifies a date with ordinary days. Even if a date such as DATE=36599 is specified, it does not mean a 'never expire' date. Instead, it just means a day of 1999/12/31. So, the VTOC expiration date of a date year after 1999 is not flagged with an asterisk.

If you code a date after 1999, then data sets that have the special expiration dates of 36599, 36699, 3651999, 3661999, 99999, or 9991999 are not flagged with an asterisk. Those data sets never expire.

The date that you code is not treated as a 'never expire' date. For example if you code 36599 or 3651999, data sets that expire December 30, 1999 will be flagged and data sets with expiration dates of December 31, 1999 and January 1, 2000 will not be flagged. If you code a ddd value that exceeds the number of days in that year, IEHLIST adds the extra days into the following year. For example, 36699 means January 1, 2000, 3671999 means January 2, 2000, and 9999 means September 25, 2001.

dddy

specifies the day of the year, ddd, and the last two digits of the year, this form designates years before 2000yy.

dddyyy

specifies the day of the year, ddd, and the year from 1900 to 2155, yyyy.

Default: No asterisks appear in the listing.

VOL=device=serial

specifies the device type and volume serial number of the volume on which the VTOC resides.

DSNAME=(name[,name[,...]])

specifies the fully qualified names of the data sets whose entries are to be listed. You can specify a maximum of 10 names, or one pattern with a wildcard such as * or %. If the list consists of only a single name, the parentheses can be omitted.

Note: If the DSNAME specified is a VSAM cluster name, the information returned indicates that the data set exists on the physical device (volume). But since there is no format 1 or 8 DSCB for the VSAM cluster, the output for a VSAM cluster will not be identical to the output for a normal data set.

IEHLIST Examples

These examples illustrate some of the uses of IEHLIST. You can use [Table 47 on page 265](#) as a quick-reference guide to IEHLIST examples.

Table 47. IEHLIST example directory

Operation	Devices	Comments	Example
LISTPDS	Disk and system output device	One PDSE directory and two partitioned data set directories are listed.	“Example 1: List Partitioned Directories Using DUMP and FORMAT” on page 265
LISTVTOC	Disk and system output device	Volume table of contents is listed in edited form; selected data set control blocks are listed in unedited form.	“Example 2: List Non-indexed Volume Table of Contents” on page 265

Examples that use **disk** in place of actual device numbers or names must be changed before use. The actual device numbers or names depend on how your installation has defined the devices to your system.

Example 1: List Partitioned Directories Using DUMP and FORMAT

In this example, the directory of a PDSE is listed. In addition, the directories of two partitioned data sets that reside on the system residence volume are listed.

```
//LISTPDIR JOB ...
//STEP1 EXEC PGM=IEHLIST
//SYSPRINT DD SYSOUT=A
//DD1 DD UNIT=diskB, VOLUME=REF=SYS1.NUCLEUS, DISP=OLD
//DD2 DD UNIT=diskA, VOLUME=SER=222222, DISP=OLD
//SYSIN DD *
LISTPDS DSN=D42.PDSE1, VOL=diskA=222222
LISTPDS DSN=(D55.PART1, D55.PART2), FORMAT
/*
```

The control statements are as follows:

- DD1 DD defines the system residence device.
- DD2 DD defines a device on which a disk volume (222222) is mounted.
- SYSIN DD defines the control data set, which follows in the input stream.
- The first LISTPDS statement indicates that the PDSE directory belonging to data set D42.PDSE1 is to be listed. The listing is in unedited (dump) format. This data set resides on volume 222222.
- The second LISTPDS statement indicates that partitioned data set directories belonging to data sets D55.PART1 and D55.PART2 are to be listed. The listing is in edited format. These data sets exist on the system residence volume.

[“Unedited listing of a partitioned data set” on page 258](#) shows an unedited entry for a partitioned member. [“Edited entry for a partitioned member” on page 257](#) shows an edited entry.

Example 2: List Non-indexed Volume Table of Contents

In this example, a non-indexed volume table of contents is listed in the first edited format. The edited listing is supplemented by an unedited listing of selected data set control blocks.

```
//VTOCLIST JOB ...
//STEP1 EXEC PGM=IEHLIST
//SYSPRINT DD SYSOUT=A
//DD2 DD UNIT=disk, VOLUME=SER=111111, DISP=OLD
//SYSIN DD *
LISTVTOC FORMAT, VOL=disk=111111
LISTVTOC DUMP, VOL=disk=111111, DSN=(SET1, SET2, SET3)
/*
```

The control statements are as follows:

- DD2 DD defines a device containing the specified volume table of contents.
- SYSIN DD defines the control data set, which follows in the input stream.
- The first LISTVTOC statement indicates that the volume table of contents on the specified disk volume is to be listed in edited form.

IEHLIST

- The second LISTVTOC statement indicates that the data set control blocks representing data sets SET1, SET2, and SET3 are to be listed in unedited form.

Chapter 15. IEHMOVE (Move System Data) Program

IEHMOVE is a system utility used to move or copy logical collections of operating system data.

The information given on IEHMOVE is provided for the sake of compatibility only. DFSMSdss should be used instead of IEHMOVE to move or copy data to volumes managed by the Storage Management Subsystem. DFSMSdss or IEBCOPY should be used to process PDSEs. You cannot use IEHMOVE with PDSEs, ISAM or VSAM data sets, or large format sequential data sets. You cannot use IEHMOVE with PDSEs, ISAM or VSAM data sets, large format sequential data sets, or data sets that have format 8 DSCBs. In the latter case, the creator of such a data set specified EATTR=OPT and the data set resides on an Extended Address Volume (EAV).

If you do use IEHMOVE to move or copy data sets to SMS-managed volumes, you must preallocate all the target data sets. If the data set you are copying or moving is cataloged, and you are moving or copying it to an SMS-managed volume, you must rename the data set.

IEHMOVE can be used to move or copy:

- A sequential, partitioned or BDAM data set residing on one to five volumes.
- A group of non-VSAM data sets cataloged in an integrated catalog facility catalog.
- A volume of data sets.
- BDAM data sets with variable-spanned records.

A move operation differs from a copy operation in that a move operation scratches source data if the data set resides on a direct access volume and the expiration date has occurred, while a copy operation leaves source data intact. In addition, for cataloged data sets, a move operation updates the catalog to refer to the moved version (unless otherwise specified), while a copy operation leaves the catalog unchanged.

The scope of a basic move or copy operation can be enlarged by:

- Including or excluding data sets from a move or copy operation
- Merging members from two or more partitioned data sets
- Including or excluding selected members
- Renaming moved or copied members
- Replacing selected members

When moving or copying a data set group or a volume containing password-protected data sets, you must provide the password each time a data set is opened or scratched.

IEHMOVE always moves or copies any user labels associated with an input data set. You cannot use your own label processing routine with IEHMOVE.

A move or copy operation results in: a moved or copied data set; no action; or an unloaded version of the source data set.

Note: If IEHMOVE is unable to successfully move or copy specified data, it tries to reorganize the data and place it on the specified output device. The reorganized data (called an **unloaded data set**) is a sequential data set consisting of 80-byte blocked records that contain the source data and control information for subsequently reconstructing the source data as it originally existed. These results depend upon the compatibility of the source and receiving volumes with respect to:

- Size of the volumes
- Allocation of space on the receiving volume
- Data set organization (sequential, partitioned, or BDAM)
- Movability of the source data set

Related reading:

- For more information about allocating SMS-managed data sets, see *z/OS DFSMS Using Data Sets*.
- For information on the ALLOCATE command, see *z/OS DFSMS Access Method Services Commands*.

Considering Volume Size Compatibility

When using partitioned or sequential data set organization, two volumes are compatible with respect to size if the source record size does not exceed the track size of the receiving volume.

When using BDAM data set organization, two volumes are compatible with respect to size if the track capacity of the source volume does not exceed the receiving track capacity of the receiving volume. BDAM data sets moved or copied to a smaller device type or tape are unloaded. If you wish to load an unloaded data set, it must be loaded to the same device type from which it was originally unloaded.

Table 48 on page 268 shows the results of move and copy operations when the receiving volume is a DASD volume that is compatible in size with the source volume. The organization of the source data set is shown along with the characteristics of the receiving volume.

Table 48. Move and copy operations—DASD receiving volume with size compatible with source volume

Receiving Volume Characteristics	Sequential Data Sets	Partitioned Data Sets	BDAM Data Sets
Space allocated by IEHMOVE (movable data)	Moved or copied	Moved or copied	Moved or copied
Space allocated by IEHMOVE (unmovable data)	Moved or copied	Moved or copied	No action
Space previously allocated, as yet unused	Moved or copied	Moved or copied	No action
Space previously allocated, partially used	No action	Moved or copied (merged)	No action

Table 49 on page 268 shows the results of move and copy operations when the receiving volume is a DASD volume that is not compatible in size with the source volume. The organization of the source data set is shown along with the characteristics of the receiving volume.

Table 49. Move and copy operations—DASD receiving volume with size incompatible with source volume

Receiving Volume Characteristics	Sequential Data Sets	Partitioned Data Sets	BDAM Data Sets
Space allocated by IEHMOVE	Unloaded	Unloaded	Unloaded
Space previously allocated, as yet unused	Unloaded	Unloaded	No action
Space previously allocated, partially used	No action	No action	No action

Table 50 on page 268 shows the results of move and copy operations when the receiving volume is not a DASD volume. The organization of the source data set is shown with the characteristics of the receiving volume.

Table 50. Move and copy operations—Non-DASD receiving volume

Receiving Volume Characteristics	Sequential Data Sets	Partitioned Data Sets	BDAM Data Sets
Movable data	Moved or copied	Unloaded	Unloaded
Unmovable data	Unloaded	Unloaded	No action

Allocating Space for a Moved or Copied Data Set

Space can be allocated for a data set on a receiving volume either by you (through the use of DD statements) or by IEHMOVE in the IEHMOVE job step.

If the source data is unmovable (that is, if it contains location-dependent code), you should allocate space on the receiving volume using absolute track allocation to ensure that the data set is placed in the same relative location on the receiving volume as it was on the source volume. Unmovable data can be moved or copied if space is allocated by IEHMOVE, but the data may not be in the same location on the receiving volume as it was on the source volume.

When data sets are to be moved or copied between unlike DASD devices, a secondary allocation should be made to ensure that ample space is available on the receiving volume.

Space for a new data set should not be allocated by you when a BDAM data set is to be moved or copied, not unloaded, because IEHMOVE cannot determine if the new data set is empty.

If IEHMOVE performs the space allocation for a new data set, the space requirement information of the old data set (if available) is used. This space requirement information is obtained from the DSCB of the source data set, if it is on a DASD volume, or from the control information in the case of an unloaded data set.

If space requirement information is available, IEHMOVE uses this information to derive an allocation of space for the receiving volume, taking into account the differences in device characteristics, such as track capacity and overhead factors. However, when data sets with variable or undefined record formats are being moved or copied between unlike DASD devices, no assumption can be made about the space that each individual record needs on the receiving device.

In general, when variable or undefined record formats are to be moved or copied, IEHMOVE tries to allocate sufficient space. This can cause too much space to be allocated under these circumstances:

- When moving or copying from a device with a relatively large block overhead to a device with a smaller block overhead, the blocks being small in relation to the block size.
- When moving or copying from a device with a relatively small block overhead to a device with a larger block overhead, the blocks being large in relation to the block size.

BDAM data sets with variable or undefined record formats always have the same amount of space allocated by IEHMOVE. This practice preserves any relative track addressing system that may exist within the data sets.

If a sequential data set, which is not an unloaded data set, on a non-DASD volume is to be moved or copied to a DASD volume, and space attributes are not available through a previous allocation, IEHMOVE makes a default space allocation. The default allocation consists of a primary allocation of 72,500 bytes of DASD storage (data and gaps) and up to 15 secondary allocations of 36,250 bytes each.

Space cannot be previously allocated for a partitioned data set that is to be unloaded unless the SPACE parameter in the DD statement making the allocation implies sequential organization. BDAM data sets should not be previously allocated because IEHMOVE cannot determine if they are empty or not.

If a move or copy operation is unsuccessful, the source data remains intact.

If a move or copy operation is unsuccessful and space was allocated by IEHMOVE, all data associated with that operation is scratched from the receiving DASD volume. If the receiving volume was tape, it will contain a partial data set.

If a move or copy operation is unsuccessful and space was previously allocated, no data is scratched from the receiving volume. If, for example, IEHMOVE moved 104 members of a 105-member partitioned data set and encountered an input/output error while moving the 105th member:

- The entire partitioned data set is scratched from the receiving volume if space was allocated by IEHMOVE.
- No data is scratched from the receiving volume if space was previously allocated. In this case, after determining the nature of the error, you need move only the 105th member into the receiving partitioned data set.

If a data set that has only user trailer labels is to be moved from a tape volume to a DASD volume, space must be previously allocated on the DASD volume to ensure that a track is reserved to receive the user labels.

Reblocking Data Sets

Data sets with fixed or variable records can be reblocked to a different block size by previously allocating the desired block size on the receiving volume. No reblocking can be performed when loading or unloading. Also, no reblocking can be performed on data sets with variable spanned or variable blocked spanned records.

When moving or copying data sets with undefined record format and reblocking to a smaller block size (that is, transferring records to a device with a track capacity *smaller* than the track capacity of the original device), you must make the block size for the receiving volume equal to or larger than the size of the largest record in the data set being moved or copied.

When copying data sets with undefined record format to a device with a *larger* track capacity, IEHMOVE will not reblock the output data set to a larger block size. IEHMOVE simply copies the source data set to the target data set.

However, if the target data set is preallocated with a larger block size than the source data set, the data set becomes unusable because the source block size is used during the copy.

Blocked format data sets that do not contain user data TTRNs or keys can be reblocked or unblocked by including the proper keyword subparameters in the DCB operand of the DD statement used to previously allocate space for the data set. The new blocking factor must be a multiple of the logical record length originally assigned to the data set.

Related reading: For information about user data TTRNs, see [z/OS DFSMS Using Data Sets](#).

Using IEHMOVE with RACF®

If the Resource Access Control Facility (RACF*), a component of the Security Server for OS/390, is active, these considerations apply:

- You must have valid RACF authorization to access any RACF-defined data sets with IEHMOVE. ALTER authorization is required to access the source data set for a MOVE function, as the source data set is scratched. When moving a volume or group of data sets, you must have adequate access authorization to all of the RACF-protected data sets on the volume or in the group.
- If you have the RACF ADSP attribute and IEHMOVE is to allocate space for the receiving data set, that data set will be automatically defined to RACF. If the data set does not have your userid as the first level qualifier, at least one of these conditions must be met:
 1. You specify MOVE or COPY with RENAME so that the first level qualifier is the correct userid,
 2. The data set being moved or copied is a group data set and you are connected to the group with CREATE authority, or
 3. You have the OPERATION attribute.
- If COPYAUTH is specified and the input data set is RACF-protected (whether or not you have the ADSP attribute) and the output data set is not preallocated, then the receiving data set of a MOVE or COPY operation is given a copy of the input data set's RACF protection access list during allocation, governed by the same restrictions for defining a data set for a user with the ADSP attribute. You must have ALTER access authorization to the input data set to either MOVE or COPY using COPYAUTH.
- The temporary work files are allocated with the nonstandard names of **SYSUT1.T<time>, **SYSUT2.T<time>, and **SYSUT3.T<time>. These names must be included as valid data set names in the RACF Naming Convention Table if that option is being used.

Moving or Copying a Data Set

IEHMOVE can be used to move or copy sequential, partitioned, and BDAM data sets, as follows:

- A sequential data set can be:
 1. Moved from one DASD volume or non-DASD volume to another (or to the same volume provided that it is a DASD volume), or
 2. Copied from one volume to another (or to the same volume provided that the data set name is changed and the receiving volume is a DASD volume).
- A partitioned data set can be:
 1. Moved from one DASD volume to another (or to the same volume), or
 2. Copied from one DASD volume to another (or to the same volume provided that the data set name is changed).
- A BDAM data set can be moved or copied from one DASD volume to another provided that the receiving device type is the same device type or larger, and that the record size does not exceed 32K bytes.

Sequential Data Sets

Table 51 on page 271 shows basic and optional move and copy operations for sequential data sets.

Table 51. Moving and copying sequential data sets

Operation	Basic Actions	Optional Actions
Move Sequential	Move the data set. For DASD, scratch the source data. For non-cataloged data sets, update the appropriate catalog to refer to the moved data set.	Prevent automatic cataloging of the moved data set. Rename the moved data set.
Copy Sequential	Copy the data set. The source data set is not scratched. The catalog is not updated to refer to the copied data set.	Delete the catalog entry for the source data set. Catalog the copied data set on the receiving volume. Rename the copied data set.

When moving or copying sequential data sets on a direct access device, IEHMOVE execution time can be reduced by using multiple BSAM buffers for input and output.

Related reading: For information on how to specify the number of buffers to be used by IEHMOVE, see “EXEC Statement” on page 277.

Partitioned Data Sets

Table 52 on page 271 shows basic and optional move and copy operations for partitioned data sets.

Table 52. Moving and copying partitioned data sets

Operation	Basic Actions	Optional Actions
Move Partitioned	Move the data set. Scratch the source data. For cataloged data sets, update the appropriate catalog to refer to the moved data set.	Prevent automatic cataloging of the moved data set. Rename the moved data set. Reallocate directory space, if the space was allocated by IEHMOVE during the move. Merge two or more data sets. Move only selected members. Replace members. Unload the data set.

Table 52. Moving and copying partitioned data sets (continued)

Operation	Basic Actions	Optional Actions
Copy Partitioned	Copy the data set. The source data is not scratched. The catalog is not updated to refer to the copied data set.	Delete the catalog entry for the source data set. Catalog the copied data set. Rename the copied data set. Reallocate directory space, unless the space previously allocated is partially used. Merge two or more data sets. Copy only selected members. Replace members. Unload the data set.

Figure 41 on page 272 shows a copied partitioned data set. IEHMOVE moves or copies partitioned members in the order in which they appear in the partitioned directory. That is, moved or copied members are placed in collating sequence on the receiving volume. The IEBCOPY utility program can be used to copy data sets whose members are not to be collated.

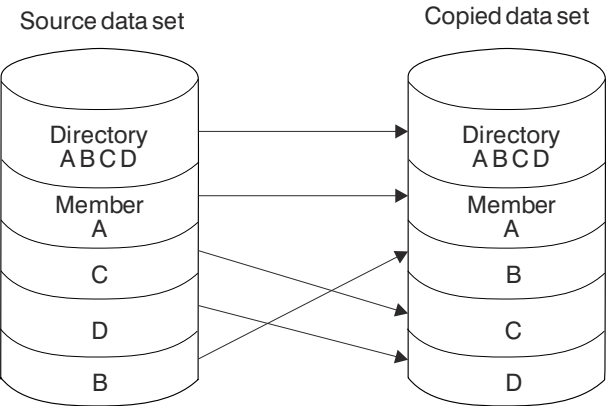


Figure 41. Partitioned data set before and after an IEHMOVE copy operation

Members that are merged into an existing data set are placed, in collating sequence, after the last member in the existing data set. If the target data set contains a member with the same name as the data set to be moved, the member will not be moved or copied unless the REPLACE statement is coded.

Figure 42 on page 273 shows members from one data set merged into an existing data set. Members B and F are copied in collating sequence.

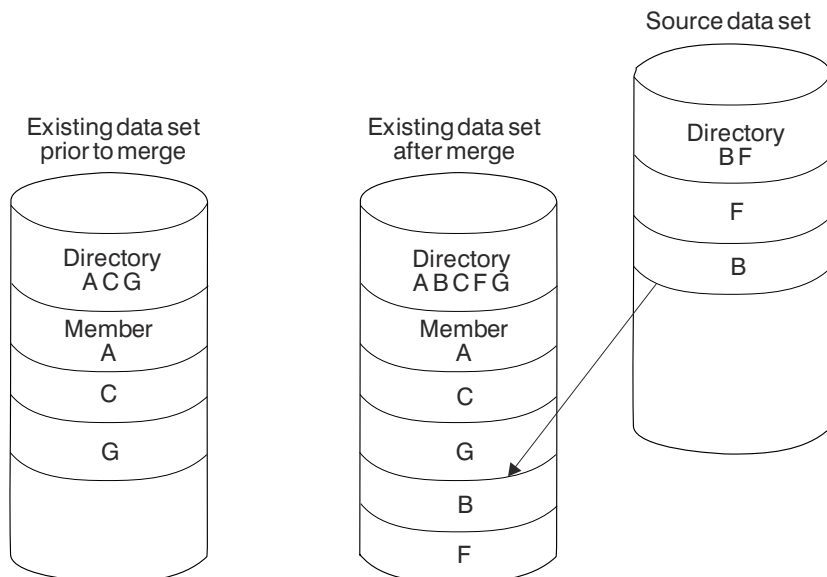


Figure 42. Merging two data sets using IEHMOVE

Figure 43 on page 273 shows how members from two data sets are merged into an existing data set. Members from additional data sets can be merged in a like manner. Members F, B, D and E from the source data sets are copied in collating sequence.

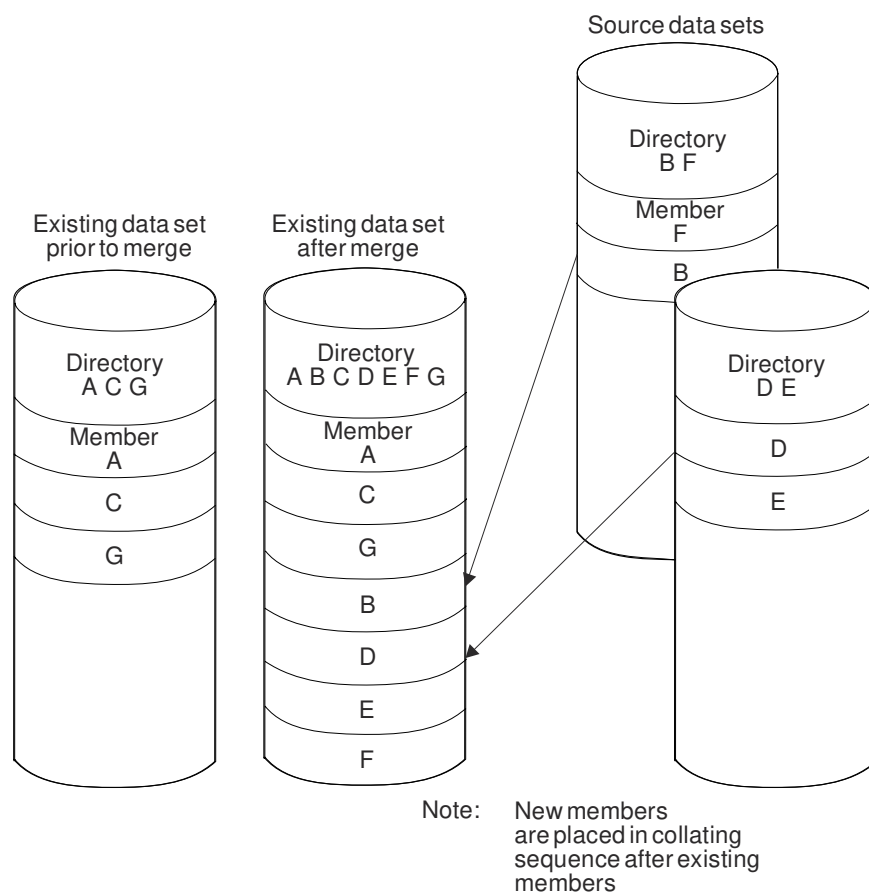


Figure 43. Merging three data sets using IEHMOVE

Related reading: For more information, see [Chapter 4, “IEBCOPY \(Library Copy\) Program,”](#) on page 41.

BDAM Data Sets

When moving or copying a BDAM data set from one device to another device of the same type, relative track and relative block integrity are maintained.

When moving or copying a BDAM data set to a larger device, relative track integrity is maintained for data sets with variable or undefined record formats; relative block integrity is maintained for data sets with fixed record formats.

When moving or copying a BDAM data set to a smaller device or a tape, the data set is unloaded. An unloaded data set is loaded only when it is moved or copied to the same device type from which it was unloaded.

BDAM data sets with variable-spanned records can be copied from one DASD volume to a compatible DASD volume provided that the record size does not exceed 32K bytes.

Because a BDAM data set can reside on one to five volumes (all of which must be mounted during any move or copy operation), it is possible for the data set to span volumes. However, single variable-spanned records are contained on one volume.

Relative track integrity is preserved in a move or copy operation for spanned records. Moved or copied BDAM data sets occupy the same relative number of tracks that they occupied on the source device.

If a BDAM data set is unloaded (moved or copied to a smaller device or tape), it must be loaded back to the same device type from which it was originally unloaded.

When moving or copying variable-spanned records to a larger device, record segments are combined and respanned if necessary. Because the remaining track space is available for new records, variable-spanned records are unloaded before being moved or copied back to a smaller device.

If you wish to create a BDAM data set without using data management BDAM macros, all data management specifications must be followed. Special attention must be given to data management specifications for R0 track capacity record content, segment descriptor words, and the BFTEK=R parameter.

When moving or copying a multivolume data set, the secondary allocation for BDAM data sets should be at least two tracks.

Related reading:

- For information about volume compatibility, see [“Considering Volume Size Compatibility” on page 268](#).
- For information about using data management specifications, see [z/OS DFSMS Using Data Sets](#).
- For information about "WRITE" macro, see [z/OS DFSMS Macro Instructions for Data Sets](#).

Multivolume Data Sets

IEHMOVE can be used to move or copy multivolume data sets. To move or copy a multivolume data set, specify the complete volume list in the VOL=SER parameter on the DD statement. A maximum of 5 volumes can be specified. The same number of volumes must be specified for the output data set as existed for the input data set. If the user wishes to consolidate a multivolume data set so that the data set will reside on fewer volumes, the output data set must be allocated on the target volume(s) by the user before moving or copying the data set. To move or copy a data set that resides on more than one tape volume, specify the volume serial numbers of all the tape volumes and the sequence numbers of the data set on the tape volumes in the utility control statement. (You can specify the sequence number even if the data set to be moved or copied is the only data set on a volume.) To move or copy a data set to more than one tape volume, specify the volume serial numbers of all the receiving volumes in the utility control statement.

Unloaded Data Sets

If IEHMOVE is unable to successfully move or copy specified data, it tries to reorganize the data and place it on the specified output device. The reorganized data (called an *unloaded data set*) is a sequential

data set consisting of 80-byte blocked records that contain the source data and control information for subsequently reconstructing the source data as it originally existed.

When an unloaded data set is moved or copied (via IEHMOVE) to a device that will support the data in its true form, the data is automatically reconstructed. For example, if you try to move a partitioned data set to a tape volume, the data is unloaded to that volume. You can re-create the data set merely by moving the unloaded data set to a DASD volume.

Unmovable Data Sets

A data set with the unmovable attribute can be moved or copied from one DASD volume to another or to the same volume provided that space has been previously allocated on the receiving volume. Change the name of the data set if move or copy is to be done to the same volume.

Moving or Copying a Group of Cataloged Data Sets

IEHMOVE can be used to move or copy a group of partitioned, sequential or BDAM data sets (a "DSGROUP") that are cataloged in integrated catalog facility and whose names are qualified by one or more identical names. For example, a group of data sets qualified by the name A.B can include data sets named A.B.D and A.B.E, but could not include data sets named A.C.D or A.D.F.

You cannot use IEHMOVE to move or copy a DSGROUP to a volume managed by the Storage Management Subsystem.

If a group of data sets is moved or copied to magnetic tape, the data sets must be retrieved one by one by data set name and file-sequence number, or by file-sequence number for unlabeled or nonstandard labeled tapes.

Access method services can be used to determine the structure of integrated catalog facility catalogs.

Table 53 on page 275 shows basic and optional move and copy operations for a group of partitioned, sequential or BDAM cataloged data sets.

Table 53. Moving and copying a group of cataloged data sets

Operation	Basic Actions	Optional Actions
Move a group of cataloged data sets	Move the data set group (excluding password-protected data sets) to the specified volumes. Scratch the source data sets (BDAM only). Merging is not done.	Prevent updating of the appropriate catalog. Include password-protected data sets in the operation. Unload data sets. If a data set group is cataloged, you may include or exclude other data sets during the operation.
Copy a group of cataloged data sets	Copy the data set group (excluding password-protected data sets). Source data sets are not scratched. Merging is not done.	Include password-protected data sets in the operation. Delete catalog entries for the source data sets. Catalog the copied data sets on the receiving volumes. Unload a data set or sets. If a data set group is cataloged, you may include or exclude other data sets during the operation.

Related reading: For more information, see [z/OS DFSMS Access Method Services Commands](#).

Moving or Copying a Volume of Data Sets

IEHMOVE can be used to move or copy the data sets of an entire DASD volume to another volume or volumes. A move operation differs from a copy operation in that the move operation scratches source data sets, while the copy operation does not. For both operations, any cataloged entries associated with the source data sets remain unchanged.

If the source volume contains a SYSCTLG data set, that data set is the last to be moved or copied onto the receiving volume.

If a volume of data sets is moved or copied to tape, sequential data sets are moved; partitioned and BDAM data sets are unloaded. The data sets must be retrieved one by one by data set name and file-sequence number, or by file-sequence number for unlabeled or nonstandard labeled tapes.

When copying a volume of data sets, you have the option of cataloging all source data sets in a SYSCTLG data set on a receiving volume. However, if a SYSCTLG data set exists on the source volume, error messages indicating that an inconsistent index structure exists are generated when the source SYSCTLG entries are merged into the SYSCTLG data set on the receiving volume.

The move-volume feature does not merge partitioned data sets. If a data set on the volume to be moved has a name identical to a data set name on the receiving volume, the data set is not moved or merged onto the receiving volume.

The copy-volume feature **does** merge partitioned data sets. If a data set on the volume to be copied has a name identical to a data set name on the receiving volume, the data set is copied and merged onto the receiving volume.

Table 54 on page 276 shows basic and optional move and copy operations for a volume of data sets.

Table 54. Moving and copying a volume of data sets

Operation	Basic Actions	Optional Actions
Move a volume of data sets	Move all data sets not protected by a password to the specified DASD volumes. Scratch the source data sets for DASD volumes.	Include password-protected data sets in the operation. Unload the data sets.
COPY a volume of data sets	Copy all data sets not protected by a password to the specified DASD volume. The source data sets are not scratched.	Include password-protected data sets in the operation. Catalog all copied data sets. Unload the data sets.

Input and Output

IEHMOVE uses the following input:

- One or more partitioned, sequential or BDAM data sets, which contain the data to be moved, copied, or merged into an output data set.
- A control data set, which contains utility control statements that are used to control the functions of the program.
- A work data set, which is a work area used by IEHMOVE.

IEHMOVE does not support VIO (virtual input/output) data sets.

IEHMOVE produces the following output:

- An output data set, which is the result of the move, copy, or merge operation.
- A message data set, that contains informational messages (for example, the names of moved or copied data sets) and error messages, if applicable.

If IEHMOVE is invoked from an application program, you can dynamically allocate the devices and data sets by issuing SVC 99 before calling IEHMOVE.

Related reading: For information about IEHMOVE return codes, see [Appendix A, “Invoking Utility Programs from an Application Program,”](#) on page 315.

Control

IEHMOVE is controlled by job and utility control statements. The job control statements are used to process or load the program, define the devices and volumes used and produced by IEHMOVE, and prevent data sets from being deleted inadvertently.

IEHMOVE is an APF-authorized program. This means that if another program calls it, that program must also be APF-authorized. To protect system integrity, the calling program must follow the system integrity requirements described in [z/OS MVS Programming: Assembler Services Guide](#).

Utility control statements are used to control the functions of the program and to define those data sets or volumes that are to be used.

Job Control Statements

Table 55 on page 277 shows the job control statements for IEHMOVE.

Table 55. IEHMOVE job control statements

Statement	Use
JOB	Starts the job.
EXEC	Specifies the program name (PGM=IEHMOVE) or, if the job control statements reside in a procedure library, the procedure name. This statement can also include optional parameter information.
SYSPRINT DD	Defines a sequential data set for messages. The data set can be written onto a system output device, a magnetic tape volume, or a direct access volume.
SYSUT1 DD	Defines a volume on which three work data sets required by IEHMOVE are allocated.
anyname DD	Defines a permanently mounted or mountable DASD volume. At least one permanently mounted volume must be identified.
tape DD	Defines a mountable tape device.
SYSIN DD	Defines the control data set. The data set, which contains utility control statements, usually follows the job control statements in the input stream; however, it can be defined either as a sequential data set or as a member of a procedure library.

Since SYSUT2 and SYSUT3 are reserved for IEHMOVE, it is not recommended to use them in an IEHMOVE job step.

EXEC Statement

The EXEC statement for IEHMOVE can contain parameter information that is used by the program to allocate additional work space or control line density on output listings. You can also code the REGION subparameter to control the region size that IEHMOVE operates in when you are moving or copying sequential data sets.

The syntax of the EXEC statement is:

Label	Statement	Parameters
// [stepname]	EXEC	PGM=IEHMOVE [, PARM=' [POWER= <i>n</i>] [, LINECNT= <i>xx</i>] '] [, REGION={ <i>nK</i> <i>nM</i> }]

where:

PGM=IEHMOVE

specifies that you want to run IEHMOVE.

PARM='[POWER=*n*] [,LINECNT=*xx*]'

specifies optional parameter information to be passed to IEHMOVE.

POWER=*n*

specifies that you want to increase the size of the space allocated to the work areas that IEHMOVE will use. You should use this parameter when moving or copying partitioned data sets that have more than 750 members. *N* may be from 1 to 999. To calculate it, divide the number of members by 750 and round the result up to be an integer. Code that value or a larger value. For example, code:

- POWER=2 when 750 to 1500 members are to be moved or copied.
- POWER=3 when 1501 to 2250 members are to be moved or copied.
- POWER=4 when 2251 to 3000 members are to be moved or copied.

The default value for *N* is 1. Note that if you code a very large value, it probably will fail due to running out of storage.

LINECNT=*xx*

specifies how many lines per page will be printed in the listing of the SYSPRINT data set. *Xx* can be a two-digit number from 04 through 99.

REGION={*nK* | *nM*}

specifies the region size you want IEHMOVE to run in when you are moving or copying *sequential* data sets. You can use this parameter to enhance IEHMOVE performance, but it is not a required parameter for moving or copying sequential data sets.

The minimum number of buffers required for enhanced IEHMOVE copy performance is 4: two for input and two for output. The size of an input buffer is computed as:

```
(INPUT BLOCKSIZE + KEY LENGTH) + 24
```

The size of an output buffer is computed as:

```
(OUTPUT BLOCKSIZE + KEY LENGTH) + 40
```

The maximum number of input buffers used by IEHMOVE is two times the number of buffers that will fit in the input track size. The maximum number of output buffers used by IEHMOVE is two times the number of buffers that will fit in the output track size.

If space for the minimum of four buffers is not available, a single buffer is used and message IEH476I is issued.

You can code this parameter in the JOB statement rather than the EXEC statement, if you prefer.

Message IEH477I, describing the number and size of your buffers, will be issued each time multiple BSAM buffers are used. If you do not specify your region size to achieve the maximum number of buffers, the last line of the message will indicate the amount by which the value of the REGION parameter should be increased in order to obtain the maximum number of buffers.

The execution time of an IEHMOVE move or copy operation will vary with the number of buffers available, the size of the data sets, and the block size.

Related reading:

- For more information about coding PARM keyword values, see [z/OS MVS JCL Reference](#).
- For information about coding the REGION parameter, see [z/OS MVS JCL Reference](#).

SYSPRINT DD Statement

The block size for the SYSPRINT data set must be a multiple of 121. Any blocking factor can be specified.

SYSUT1 DD Statement

The SYSUT1 DD statement defines a DASD volume that IEHMOVE uses for its work areas. The SYSUT1 DD statement must be coded:

```
//SYSUT1 DD UNIT=xxxx, VOL=SER=xxxxxx, DISP=OLD
```

The UNIT and VOLUME parameters define the device type and volume serial number. The DISP=OLD specification prevents the inadvertent deletion of a data set. The SYSUT1 DD statement cannot define an SMS-managed volume.

At least three utility work areas of 13, 13, and 26 contiguous tracks, respectively, must be available for work space on the volume defined by the SYSUT1 DD statement. (This figure is based on a 3380 being the work volume. If a direct access device other than a 3380 is used, an equivalent amount of space must be available.)

IEHMOVE automatically calculates and allocates the amount of space needed for the work areas. No SPACE parameter, therefore, should be coded in the SYSUT1 DD statement. However, you can increase the size of the work areas by coding the POWER value in the PARM parameter of the EXEC statement.

Prior space allocations can be made by specifying a dummy execution of the IEHPROGM utility program before the execution of IEHMOVE.

Note: IEHMOVE uses nonstandard data set names to allocate its work data sets. The names start with one or more asterisks. These work data sets are deleted at completion of the requested functions.

However, if IEHMOVE does not end normally (abend, system malfunction, and so forth), these work data sets remain on the DASD volume and cannot be deleted with any IBM utility. You can delete them by executing an IEFBR14 job and specifying their data set names in single quotes with DISP=(OLD,DELETE).

anyname DD Statement

A DD statement must be included for each permanently mounted or mountable volume referred to in the job step. These DD statements are used to allocate devices.

The DD statement should be coded:

```
//anyname1 DD UNIT=xxxx, VOL=SER=xxxxxx,  
DISP=OLD
```

The UNIT and VOLUME parameters define the device type and volume serial number. The DISP=OLD specification prevents the inadvertent creation of a data set.

You can also code the DSN parameter to identify a volume, if the data set you name resides on that volume. When unloading a data set from one DASD volume to another, this parameter is required for the data set to be unloaded. An unloaded data set on a DASD volume can only be loaded to the same device type from which it was unloaded.

For mountable volumes, when the number of volumes to be processed is greater than the number of devices defined by DD statements, there must be an indication (in the applicable DD statements) that multiple volumes are to be processed. This indication can be in the form of deferred mounting, as follows:

```
//anyname2 DD UNIT=(xxxx, ,DEFER), VOL=(PRIVATE, ...),  
// DISP=(...,KEEP)
```

Here, the PRIVATE indication in the VOL parameter is optional. Unit affinity cannot be used on DD statements defining mountable devices.

tape DD Statement

The tape DD statement can be coded:

```
//tape DD DSN=xxxxxxx,UNIT=xxxx,VOLUME=SER=xxxxxx,
//        DISP=(...,KEEP),LABEL=(...,...),DCB=(TRTCH=C,DEN=x)
```

When unloading a data set from one DASD volume to another, the data set name (DSN=) must be coded on the DD statement for the data set to be unloaded. An unloaded data set on a DASD volume can only be loaded to the same device type from which it was unloaded.

A utility control statement parameter refers to the tape DD statement for label and mode information.

The date on which a data set is moved or copied to a magnetic tape volume is automatically recorded in the HDR1 record of a standard tape label if a TODD parameter is specified in a utility control statement. An expiration date can be specified by including the EXPDT or RETPD subparameters of the LABEL keyword in the DD statement referred to by a TODD parameter.

A sequence number, for a data set on a tape volume, or a specific device number (for example, unit address 190), must be specified on a utility control statement instead of a reference to a DD statement. To move or copy a data set from or to a tape volume containing more than one data set, specify the sequence number of the data set in the utility control statement. To move or copy a data set from or to a specific device, specify the unit address (rather than a group name or device type) in the utility control statement. To copy to a unit record or unlabeled tape volume, specify any standard name or number in the utility control statement.

The tape DD statement can be used to communicate DCB attributes of data sets residing on tape volumes that do not have standard labels to IEHMOVE. If no DCB attributes are specified, an undefined record format and a block size of 2560 are assumed. However, in order to recognize unloaded data sets on an unlabeled tape volume, the DCB attributes must be specified as follows:

```
DCB=(RECFM=FB,LRECL=80,BLKSIZE=800) .
```

In no case can the block size exceed 32760 bytes.

SYSIN DD Statement

The block size for the SYSIN data set must be a multiple of 80. Any blocking factor up to a block size of 32760 bytes can be specified.

Utility Control Statements

IEHMOVE is controlled by the utility control statements shown in [Table 56 on page 280](#).

Table 56. IEHMOVE utility control statements

Statement	Use
MOVE DSNNAME	Moves a data set.
COPY DSNNAME	Copies a data set.
MOVE DSGROUP	Moves a group of cataloged partitioned, sequential or BDAM cataloged data sets.
COPY DSGROUP	Copies a group of cataloged partitioned, sequential or BDAM cataloged data sets.
MOVE PDS	Moves a partitioned data set.
COPY PDS	Copies a partitioned data set.
MOVE VOLUME	Moves a volume of data sets.
COPY VOLUME	Copies a volume of data sets.

In addition, there are four *subordinate* control statements that can be used to modify the effect of a MOVE DSGROUP, COPY DSGROUP, MOVE PDS, COPY PDS, MOVE CATALOG, or COPY CATALOG operation. The subordinate control statements are:

- **INCLUDE** statement, which is used to enlarge the scope of a MOVE PDS, or COPY PDS statement by including a member or data set not explicitly included by the statement it modifies.
- **EXCLUDE** statement, which is used with a MOVE PDS, COPY PDS, statement to exclude a member from a move or copy operation.
- **REPLACE** statement, which is used with a MOVE PDS or COPY PDS statement to exclude a member from a move or copy operation and to replace it with a member from another partitioned data set.
- **SELECT** statement, which is used with MOVE PDS or COPY PDS statements to select members to be moved or copied and, optionally, to rename the specified members.

Continuation requirements for utility control statements are described in [“Continuing utility control statements”](#) on page 8.

MOVE DSNNAME and COPY DSNNAME Statements

The MOVE DSNNAME statement is used to move a data set. The source data set is scratched. If the data set is cataloged, the catalog is automatically updated unless UNCATLG and FROM are specified.

The COPY DSNNAME statement is used to copy a data set. The source data set, if cataloged, remains cataloged unless UNCATLG or CATLG is specified without the RENAME and FROM parameters. The source data set is not scratched in a copy.

The syntax of the MOVE DSNNAME and COPY DSNNAME statements is:

Label	Statement	Parameters
[<i>label</i>]	{MOVE COPY}	DSNAME= <i>name</i> , TO=device={ <i>serial</i> (<i>list</i>)} [, {FROM=device={ <i>serial</i> (<i>list</i>)} [, UNCATLG] [, CATLG] [, RENAME= <i>name</i>] [, FROMDD= <i>ddname</i>] [, TODD= <i>ddname</i>] [, UNLOAD] [, COPYAUTH]
Note: CATLG may only be coded with COPY DSNNAME		

where:

DSNAME=*name*

specifies the fully qualified name of the data set to be moved or copied.

TO=device={*serial* | (*list*)}

specifies the device type and volume serial number of the volume or volumes to which the specified data set is to be moved or copied. If the data set resides on more than one volume, code the list of volume serial numbers in parentheses, separating the numbers with commas.

FROM=device={*serial* | (*list*)}

specifies the device number or device type and serial number of the volume on which the data set resides. If the data set resides on more than one volume, enclose the list of serial numbers within parentheses, separating the numbers with commas.

If the data set is cataloged, do not code "FROM".

If you want to specify a specific device rather than device type, code the device number in the *device* subparameter.

When FROM is used with MOVE DSNAME, the catalog will not be updated.

If the data set resides on a tape device, the serial number must be enclosed in parentheses, and the data set sequence number must be included as follows: "(*serial,sequence number*)".

If FROM is omitted, the data set is assumed to be cataloged in the integrated catalog facility catalog.

UNCATLG

specifies that the catalog entry pertaining to the source data set is to be removed. This parameter should be used only if the source data set is cataloged. If the volume is identified by FROM, UNCATLG is ignored. Alias entries in catalogs for the source data set is lost and can be replaced with access method services if the data set is later cataloged. For more information, see the DELETE command in *z/OS DFSMS Access Method Services Commands*. For a MOVE operation, UNCATLG inhibits cataloging of the output data set.

CATLG

specifies that the copied data sets are to be cataloged as described as follows:

1. The cataloging is done in the integrated catalog facility catalog.
2. If the RENAME and FROM parameters are omitted, the entries for the source data sets are deleted from the appropriate catalog to permit the copied data sets to be recataloged.

For proper results, this control statement option must be used instead of specifying the "CATLG" option in the "DISP" parameter in the DD statements.

CATLG may only be coded with COPY DSNAME.

RENAME=name

specifies that the data set is to be renamed, and indicates the new name.

FROMDD=ddname

specifies the name of the DD statement from which DCB and LABEL information, except data set sequence number, can be obtained for input data sets on tape volumes. The FROMDD operand can be omitted, if the data set has standard labels and resides on a 9-track tape volume.

TODD=ddname

specifies the name of a DD statement from which DCB (except RECFM, BLKSIZE and LRECL) and LABEL (except data set sequence number) information for an output data set on a tape volume can be obtained.

The DD statement describes the mode and label information to be used when creating the output data set on tape volumes. Record format, blocksize and logical record length information, if coded, is ignored.

When UNLOAD is specified, it describes the mode and label information to be used when unloading the data set. Record format, blocksize and logical record length information, if coded, must specify (RECFM=FB, BLKSIZE=800, LRECL=80).

TODD must be specified in the control statement when an expiration data (EXPDT) or retention period (RETPD) is to be created or changed.

The TODD parameter can be omitted for 9-track tapes with standard labels and default density for the unit type specified.

UNLOAD

specifies that the data set is to be unloaded to the receiving volumes.

COPYAUTH

specifies that the receiving data set is to be given the same access list as the input data set, if the input data set is RACF protected and the output data set is not preallocated.

MOVE DSGROUP and COPY DSGROUP Statements

The MOVE DSGROUP statement is used to move groups of data sets whose names are partially qualified by one or more identical names. The data sets may be cataloged on several catalogs. Source data sets are scratched. Data set groups to be moved must reside on DASD volumes. Only data sets that could be moved by MOVE DSNAMES or MOVE PDS can be moved by MOVE DSGROUP. Alias entries in catalogs for the data sets are lost and can be replaced with access method services.

The COPY DSGROUP statement is used to copy groups of data sets whose names are partially qualified by one or more identical names. The data sets may be cataloged on several catalogs. Only data sets that can be copied with COPY DSNAMES or COPY PDS can be copied with COPY DSGROUP. Data set groups to be copied must reside on DASD volumes.

Related reading: For more information, see [z/OS DFSMS Access Method Services Commands](#).

MOVE DSGROUP operations cause the catalog to be updated automatically unless UNCATLG is specified. COPY DSGROUP operations leave the source data sets cataloged unless UNCATLG or CATLG is specified without the RENAME and FROM parameters.

The syntax of the MOVE and COPY DSGROUP statements is:

Label	Statement	Parameters
[<i>label</i>]	{MOVE COPY}	DSGROUP [= <i>name</i>] [, TO=device={ <i>serial</i> (<i>list</i>) } [, PASSWORD] [, UNCATLG] [, CATLG] [, TODD= <i>ddname</i>] [, UNLOAD] [, COPYAUTH]
Note: CATLG may only be coded with COPY DSGROUP		

where:

DSGROUP=[*name*]

specifies the cataloged data sets to be moved or copied. If *name* is a fully qualified data set name, only that data set is not moved or copied. If *name* is one or more qualifiers, but not fully qualified, all data sets whose names are qualified by *name* are moved or copied. If *name* is omitted, all data sets whose names are found in the searched catalog are moved or copied.

TO=device={*serial*|(list)}

specifies the device type and volume serial number of the volume or volumes to which the specified group of data sets is to be moved or copied. If the group of data sets is on more than one volume, code the list of serial numbers in parentheses, separating the numbers with commas.

PASSWORD

specifies that password protected data sets are included in the operation. This is not VSAM password protection, but the data set password scheme. This is described in the [z/OS DFSMSdfp Advanced Services](#). If PASSWORD is omitted, only data sets that are not protected are copied or moved.

UNCATLG

specifies that the catalog entries pertaining to the source data sets are to be removed. This parameter should be used only if the source data set is cataloged. If the volume is identified by FROM, UNCATLG is ignored. Alias entries in integrated catalog facility for the source data sets are lost and can be replaced with access method services if the data sets are later cataloged. For more information, see [z/OS DFSMS Access Method Services Commands](#). For a MOVE operation, UNCATLG inhibits cataloging of the output data sets.

CATLG

specifies that the cataloging is done in the integrated catalog facility catalog.

For proper results, this control statement option must be used instead of specifying the "CATLG" option in the "DISP" parameter in the DD statements.

CATLG may only be coded with COPY DSGROUP.

TODD=ddname

specifies the name of a DD statement from which DCB (except RECFM, BLKSIZE and LRECL) and LABEL (except data set sequence number) information for output data sets on tape volumes can be obtained.

The DD statement describes the mode and label information to be used when creating output data sets on tape volumes. Record format, blocksize and logical record length information, if coded, is ignored.

When UNLOAD is specified, it describes the mode and label information to be used when creating unloaded versions of data sets on tape volumes. Record format, blocksize and logical record length information, if coded, must specify (RECFM=FB, BLKSIZE=800, LRECL=80).

TODD must be specified in the control statement when an expiration data (EXPDT) or retention period (RETPD) is to be created or changed.

The TODD parameter can be omitted for 9-track tapes with standard labels and default density for the unit type specified.

UNLOAD

specifies that the data sets are to be unloaded to the receiving volumes.

COPYAUTH

specifies that the receiving data set is to be given the same access list as the input data set, if the input data set is RACF protected and the output data set is not preallocated.

MOVE PDS and COPY PDS Statements

The MOVE PDS statement is used to move partitioned data sets. When used in conjunction with INCLUDE, EXCLUDE, REPLACE, or SELECT statements, the MOVE PDS statement can be used to merge selected members of several partitioned data sets or to replace members. The source data set is scratched.

The COPY PDS statement is used to copy partitioned data sets. When used in conjunction with INCLUDE, EXCLUDE, REPLACE, or SELECT statements, the COPY PDS statement can be used to merge selected members of several partitioned data sets or to replace members.

If IEHMOVE is used to allocate space for an output partitioned data set, the MOVE PDS or COPY PDS statements can be used to expand a partitioned directory.

If the receiving volume contains a partitioned data set with the same name, the two data sets are merged.

MOVE PDS causes the appropriate catalog to be updated automatically unless UNCATLG and FROM are specified. COPY PDS leaves the source data set cataloged unless UNCATLG or CATLG is specified without the RENAME and FROM parameters.

The syntax of the MOVE PDS and COPY PDS statements is:

Label	Statement	Parameters
[label]	{MOVE COPY}	PDS=name , TO=device={serial list} [, {FROM=device=serial}] [, EXPAND=nn] [, UNCATLG]

Label	Statement	Parameters
		[, CATLG] [, RENAME= <i>name</i>] [, FROMDD= <i>ddname</i>] [, TODD= <i>ddname</i>] [, UNLOAD] [, COPYAUTH]
Note: CATLG may only be coded with COPY PDS		

where:

PDS=*name*

specifies the fully qualified name (that is, the name with all its qualifiers, if any) of the partitioned data set to be moved or copied.

TO=*device*=*{serial|list}*

specifies the device type and volume serial number of the volume to which the partitioned data set is to be moved or copied. If you are unloading the partitioned data set to multiple tape volumes, code the list of serial numbers in parentheses, separating the numbers with commas.

FROM=*device*=*serial*

specifies the device type and serial number of the volume on which the data set resides.

If the data set is cataloged, do not code "FROM".

If you want to specify a specific device rather than device type, code the device number in the *device* subparameter.

When FROM is used with MOVE PDS, the catalog will not be updated.

If FROM is omitted, the data set is assumed to be cataloged in the integrated catalog facility catalog.

EXPAND=*nn*

specifies the decimal number (up to 99) of 256-byte records to be added to the directory of the specified partitioned data set. For COPY, EXPAND cannot be specified if space is previously allocated. For MOVE, EXPAND will be ignored if space is previously allocated.

UNCATLG

specifies that the catalog entry pertaining to the source partitioned data set is to be removed. This parameter should be used only if the source data set is cataloged. If the volume is identified by FROM, UNCATLG is ignored. Alias entries in catalogs for the source data set is lost and can be replaced with access method services if the data set is later cataloged. For more information, see [z/OS DFSMS Access Method Services Commands](#). For a MOVE operation, UNCATLG inhibits cataloging of the output data set.

CATLG

specifies that the copied data sets are to be cataloged as follows.

1. The cataloging is done in the integrated catalog facility catalog.
2. If the RENAME and FROM parameters are omitted, the entries for the source data sets are deleted from the appropriate catalog to permit the copied data sets to be recataloged.

For proper results, this control statement option must be used instead of specifying the "CATLG" option in the "DISP" parameter in the DD statements.

CATLG may only be coded with COPY PDS.

RENAME=*name*

specifies that the data set is to be renamed, and indicates the new name.

FROMDD=ddname

specifies the name of the DD statement from which DCB and LABEL information, except data set sequence number, can be obtained for input data sets on tape volumes. The tape data set must be an unloaded version of a partitioned data set. The FROMDD operand can be omitted if the data set has standard labels and resides on a 9-track tape volume.

TODD=ddname

specifies the name of a DD statement from which DCB (except RECFM, BLKSIZE and LRECL) and LABEL (except data set sequence number) information for the output data set can be obtained, when the data set is being unloaded to tape. Record format, blocksize and logical record length information, if coded, must specify (RECFM=FB, BLKSIZE=800, LRECL=80).

TODD must be specified in the control statement when an expiration data (EXPDT) or retention period (RETPD) is to be created or changed.

The TODD parameter can be omitted for 9-track tapes with standard labels and default density for the unit type specified.

UNLOAD

specifies that the data set is to be unloaded to the receiving volumes.

COPYAUTH

specifies that the receiving data set is to be given the same access list as the input data set, if the input data set is RACF protected and the output data set is not preallocated.

MOVE VOLUME and COPY VOLUME Statements

The MOVE VOLUME statement is used to move all the data sets residing on a specified volume. The COPY VOLUME statement is used to copy all the data sets residing on a specified volume.

Any catalog entries associated with the data sets remain unchanged. Data sets to be moved or copied must reside on DASD volumes.

The syntax of the MOVE VOLUME and COPY VOLUME statements is:

Label	Statement	Parameters
[<i>label</i>]	{MOVE COPY}	VOLUME= <i>device=serial</i> , TO= <i>device=list</i> [, PASSWORD] [, CATLG] [, TODD= <i>ddname</i>] [, UNLOAD] [, COPYAUTH]
Note: CATLG may only be coded with COPY VOLUME		

where:

VOLUME=device=serial

specifies the device type and volume serial number of the source volume.

TO=device=serial

specifies the device type and volume serial number of the volume to which the volume of data sets is to be moved or copied.

PASSWORD

specifies that password protected data sets are included in the operation. This is not VSAM password protection, but the OS password scheme. If PASSWORD is omitted, only data sets that are not protected are copied or moved.

CATLG

specifies that the copied data sets are to be cataloged.

For proper results, this control statement option must be used instead of specifying the "CATLG" option in the "DISP" parameter in the DD statements.

CATLG may only be coded with COPY VOLUME.

TODD=ddname

specifies the name of a DD statement from which DCB (except RECFM, BLKSIZE and LRECL) and LABEL (except data set sequence number) information for output data sets on tape volumes can be obtained.

The DD statement describes the mode and label information to be used when creating output data sets on tape volumes. Record format, blocksize and logical record length information, if coded, is ignored.

When UNLOAD is specified, it describes the mode and label information to be used when creating unloaded versions of data sets on tape volumes. Record format, blocksize and logical record length information, if coded, must specify (RECFM=FB, BLKSIZE=800, LRECL=80).

TODD must be specified in the control statement when an expiration data (EXPDT) or retention period (RETPD) is to be created or changed.

The TODD parameter can be omitted for 9-track tapes with standard labels and default density for the unit type specified.

UNLOAD

specifies that the data sets are to be unloaded to the receiving volumes.

COPYAUTH

specifies that the receiving data set is to be given the same access list as the input data set, if the input data set is RACF protected and the output data set is not preallocated.

INCLUDE Statement

The INCLUDE statement is used to enlarge the scope of MOVE DSGROUP, COPY DSGROUP, MOVE PDS, or COPY PDS statements by including a member or a data set not explicitly defined in those statements. The INCLUDE statement follows the MOVE or COPY statement whose function it modifies. The record characteristics of the included partitioned data sets must be compatible with those of the other partitioned data sets being moved or copied.

Any number of INCLUDE statements can modify a MOVE or COPY statement. For a partitioned data set, the INCLUDE statement is invalid when data is unloaded or when unloaded data is moved or copied. For DSGROUP operations, INCLUDE is invalid.

The syntax of the INCLUDE statement is:

Label	Statement	Parameters
[<i>label</i>]	INCLUDE	DSNAME= <i>name</i> [, MEMBER= <i>membername</i>] [, {FROM= <i>device</i> = <i>{serial}</i> (<i>list</i>) }]

where:

DSNAME=*name*

specifies the fully qualified name of a data set. If used in conjunction with MOVE or COPY DSGROUP, the named data set is included in the group. If used in conjunction with MOVE or COPY PDS, either the named partitioned data set or a member of it (if the MEMBER parameter is specified) is included in the operation.

MEMBER=membername

specifies the name of one member in the partitioned data set named in the DSNAME parameter. The named member is merged with the partitioned data set being moved or copied. Neither the partitioned data set containing the named member nor the member is scratched.

FROM=device={serial| (list)}

specifies the device type and serial number of the volume on which the data sets to be included reside. If the data sets reside on more than one volume, enclose the list of serial numbers within parentheses, separating the numbers with commas.

If the data set is cataloged, do not code "FROM".

If you want to specify a specific device rather than device type, code the device number in the *device* subparameter.

If the data set resides on a tape device, the serial number must be enclosed in parentheses, and the data set sequence number must be included as follows: "(*serial,sequence number*)".

EXCLUDE Statement

The EXCLUDE statement is used to restrict the scope of MOVE DSGROUP, COPY DSGROUP, MOVE PDS, COPY PDS, MOVE CATALOG, or COPY CATALOG statements by excluding a specific portion of data defined in those statements.

Partitioned data set members excluded from a MOVE PDS operation cannot be recovered (the source data set is scratched). Any number of EXCLUDE statements can modify a MOVE PDS or COPY PDS statement.

Source data sets excluded from a MOVE DSGROUP or MOVE CATALOG operation remain available. Only one EXCLUDE statement can modify a MOVE DSGROUP, COPY DSGROUP. The EXCLUDE statement is invalid when data is unloaded or when unloaded data is moved or copied.

The syntax of the EXCLUDE statement is:

Label	Statement	Parameters
[<i>label</i>]	EXCLUDE	{DSGROUP= <i>name</i> MEMBER= <i>membername</i> }

where:

DSGROUP=name

Specifies the cataloged data sets or the catalog entries to be excluded in when moving a data set group or catalog. If used in conjunction with MOVE DSGROUP or COPY DSGROUP, all cataloged data sets whose names are qualified by *name* are excluded from the operation. If used in conjunction with MOVE CATALOG or COPY CATALOG, all catalog entries whose names are qualified by *name* are excluded from the operation.

MEMBER=membername

specifies the name of a member to be excluded from a MOVE or COPY PDS operation.

SELECT Statement

The SELECT statement is used with the MOVE PDS or COPY PDS statement to select members to be moved or copied, and to optionally rename these members. The SELECT statement cannot be used with either the EXCLUDE or REPLACE statement to modify the same MOVE PDS or COPY PDS statement. The SELECT statement is invalid when data is unloaded or when unloaded data is moved or copied. Because the source data set is scratched, members not selected in a MOVE PDS operation cannot be recovered.

The syntax of the SELECT statement is:

Label	Statement	Parameters
[<i>label</i>]	SELECT	MEMBER={(<i>name1</i> [, <i>name2</i>] [, ...]) ((<i>name1,newname1</i>) [, (<i>name2,newname2</i>)] [, ...]) }

where:

MEMBER={(*name1* [,*name2*][,...])| ((*name1,newname1*) [(*name2,newname2*)[,...])}

specifies the names of the members to be moved or copied by a MOVE or COPY PDS operation, and, optionally, new names to be assigned to the members.

REPLACE Statement

The REPLACE statement is used with a MOVE PDS or COPY PDS statement to exclude a member from the operation and replace it with a member from another partitioned data set. The new member must have the same name as the old member and must possess compatible record characteristics. Any number of REPLACE statements can modify a MOVE PDS or COPY PDS statement. The REPLACE statement is invalid when data is unloaded or when unloaded data is moved or copied.

The syntax of the REPLACE statement is:

Label	Statement	Parameters
[<i>label</i>]	REPLACE	DSNAME= <i>name</i> , MEMBER= <i>name</i> [, {FROM= <i>device=serial</i>]

where:

DSNAME=*name*

specifies the fully qualified name of the partitioned data set that contains the replacement member.

MEMBER=*membername*

specifies the name of one member in the partitioned data set named in the DSNAME parameter. The member replaces an identically named member in the partitioned data set being moved or copied. Neither the partitioned data set containing the named member nor the member is scratched.

FROM=*device=serial*

specifies the device type and serial number of the volume on which the data set which contains the replacement member resides.

If the data set is cataloged, do not code "FROM".

If you want to specify a specific device rather than device type, code the device number in the *device* subparameter.

IEHMOVE Examples

These examples illustrate some of the uses of IEHMOVE. You can use [Table 57 on page 289](#) as a quick-reference guide to IEHMOVE examples.

Table 57. IEHMOVE example directory

Operation	Data Set Organization	Device	Comments	Example
MOVE	Data Set Group	Disk	Data set group is moved.	“Example 1: Move Sequential Data Sets from Disk Volume to Separate Volumes” on page 290
MOVE	Partitioned	Disk	A partitioned data set is moved; a member from another partitioned data set is merged with it.	“Example 2: Move Partitioned Data Set to Disk Volume and Merge” on page 290
MOVE	Partitioned	Disk	A data set is moved to a volume on which space was previously allocated.	“Example 4: Move Partitioned Data Set to Allocated Space” on page 291
MOVE	Partitioned	Disk	Three data sets are moved and unloaded to a volume on which space was previously allocated.	“Example 5: Move and Unload Partitioned Data Sets Volume” on page 292

Table 57. IEHMOVE example directory (continued)

Operation	Data Set Organization	Device	Comments	Example
MOVE	Sequential	Disk	Source volume is demounted after job completion. Two mountable disks.	“Example 1: Move Sequential Data Sets from Disk Volume to Separate Volumes” on page 290
MOVE	Sequential	Disk and Tape	A sequential data set is unloaded to an unlabeled 9-track tape volume.	“Example 6: Unload Sequential Data Set onto Unlabeled Tape Volume” on page 293
MOVE	Sequential	Disk and Tape	Unloaded data sets are loaded from a single volume.	“Example 7: Load Unloaded Sequential Data Sets from Labeled Tape” on page 293
MOVE	Volume	Disk	A volume of data sets is moved to a disk volume.	“Example 3: Move Volume of Data Sets to Disk Volume” on page 291

Examples that use **disk** or **tape** in place of actual device names or numbers must be changed before use. The actual device names or numbers depend on how your installation has defined the devices to your system.

Example 1: Move Sequential Data Sets from Disk Volume to Separate Volumes

In this example, three sequential data sets (SEQSET1, SEQSET2, and SEQSET3) are moved from a disk volume to three separate disk volumes. Each of the three receiving volumes is mounted when it is required by IEHMOVE. The source data sets are not cataloged. Space is allocated by IEHMOVE.

```
//MOVEDS JOB ...
//STEP1 EXEC PGM=IEHMOVE
//SYSPRINT DD SYSOUT=A
//SYSUT1 DD UNIT=disk,VOLUME=SER=333333,DISP=OLD
//DD1 DD UNIT=(disk,,DEFER),DISP=OLD,
// VOLUME=(PRIVATE,,SER=(222222))
//DD2 DD UNIT=(disk,,DEFER),DISP=OLD,
// VOLUME=(PRIVATE,,SER=(222333))
//DD3 DD UNIT=(disk,,DEFER),DISP=OLD,
// VOLUME=(PRIVATE,,SER=(222444))
//DD4 DD VOLUME=(PRIVATE,RETAIN,SER=(444444)),
// UNIT=disk,DISP=OLD
//SYSIN DD *
MOVE DSNAME=SEQSET1,TO=disk=222222,FROM=disk=444444
MOVE DSNAME=SEQSET2,TO=disk=222333,FROM=disk=444444
MOVE DSNAME=SEQSET3,TO=disk=222444,FROM=disk=444444
/*
```

The control statements are discussed as follows:

- SYSUT1 DD defines the disk device that is to contain the work data set.
- DD1, DD2, and DD3 DD define the receiving volumes.
- DD4 DD defines a device on which the source volume is mounted. Because the RETAIN subparameter is included, the volume remains mounted until the job has completed.
- SYSIN DD defines the control data set, which follows in the input stream.
- MOVE moves the source data sets to volumes 222222, 222333, and 222444, respectively. The source data sets are scratched.

Example 2: Move Partitioned Data Set to Disk Volume and Merge

In this example, a partitioned data set (PARTSET1) is moved to a disk volume. In addition, a member (PARMEM3) from another partitioned data set (PARTSET2) is merged with the source members on the receiving volume. The source partitioned data set (PARTSET1) is scratched. Space is allocated by IEHMOVE.

```
//MOVEPDS JOB ...
//STEP1 EXEC PGM=IEHMOVE
//SYSPRINT DD SYSOUT=A
//SYSUT1 DD UNIT=disk,VOLUME=SER=333000,DISP=OLD
//DD1 DD UNIT=disk,VOLUME=SER=111111,DISP=OLD
//DD2 DD UNIT=disk,VOLUME=SER=222111,DISP=OLD
//DD3 DD UNIT=disk,VOLUME=SER=222222,DISP=OLD
//DD4 DD UNIT=disk,VOLUME=SER=222333,DISP=OLD
//SYSIN DD *
      MOVE PDS=PARTSET1,T0=disk=222333,FROM=disk=222111
      INCLUDE DSNAME=PARTSET2, MEMBER=PARMEM3, FROM=disk=222222
/*
```

The control statements are discussed as follows:

- SYSUT1 DD defines the disk volume that is to contain the work data set.
- DD1 DD defines the system residence device.
- The DD2, DD3, and DD4 DD statements define devices that are to contain the two source volumes and the receiving volume.
- SYSIN DD defines the control data set, which follows in the input stream.
- MOVE defines the source partitioned data set, the volume that contains it, and its receiving volume.
- INCLUDE includes a member from a second partitioned data set in the operation.

Example 3: Move Volume of Data Sets to Disk Volume

In this example, a volume of data sets is moved to a disk volume. All data sets that are successfully moved are scratched from the source volume; however, any catalog entries pertaining to those data sets are not changed. Space is allocated by IEHMOVE.

```
//MOVEVOL JOB ...
//STEP1 EXEC PGM=IEHMOVE
//SYSPRINT DD SYSOUT=A
//SYSUT1 DD UNIT=disk,VOLUME=SER=222222,DISP=OLD
//DD1 DD UNIT=disk,VOLUME=SER=111111,DISP=OLD
//DD2 DD UNIT=disk,VOLUME=SER=222222,DISP=OLD
//DD3 DD UNIT=disk,VOLUME=SER=333333,DISP=OLD
//SYSIN DD *
      MOVE VOLUME=disk=333333,T0=disk=222222,PASSWORD
/*
```

The control statements are discussed as follows:

- SYSUT1 DD defines the device that is to contain the work data set.
- DD1 DD defines the system residence device.
- DD2 DD defines the device on which the receiving volume is mounted.
- DD3 DD defines a device on which the source volume is mounted.
- SYSIN DD defines the control data set, which follows in the input stream.
- MOVE specifies a move operation for a volume of data sets and defines the source and receiving volumes. This statement also indicates that password-protected data sets are included in the operation.

Example 4: Move Partitioned Data Set to Allocated Space

In this example, a partitioned data set is moved to a disk volume on which space has been previously allocated for the data set. The source data set is scratched.

```
//MOVEPDS JOB ...
//STEP1 EXEC PGM=IEHMOVE
//SYSPRINT DD SYSOUT=A
//SYSUT1 DD UNIT=disk,VOLUME=SER=222222,DISP=OLD
//DD1 DD UNIT=disk,VOLUME=SER=111111,DISP=OLD
//DD2 DD UNIT=disk,VOLUME=SER=222222,DISP=OLD
//DD3 DD UNIT=disk,VOLUME=SER=333333,DISP=OLD
//SYSIN DD *
```

```

/*      MOVE    PDS=PDSSET1,T0=disk=222222,FROM=disk=333333

```

The control statements are discussed as follows:

- SYSUT1 DD defines the device that is to contain the work data set.
- DD1 DD defines the system residence device.
- DD2 DD defines the device on which the receiving volume is to be mounted.
- DD3 DD defines a device on which the source volume is mounted.
- SYSIN DD defines the control data set, which follows in the input stream.
- MOVE specifies a move operation for the partitioned data set PDSSET1 and defines the source and receiving volumes.

Example 5: Move and Unload Partitioned Data Sets Volume

In this example, three partitioned data sets are moved from three separate source volumes to a disk volume. The source data set PDSSET3 is unloaded. (The record size exceeds the track capacity of the receiving volume.)

```

//MOVEPDS  JOB    ...
//STEP1    EXEC  PGM=IEHMOVE
//SYSPRINT DD  SYSOUT=A
//SYSUT1   DD  UNIT=disk,VOLUME=SER=222222,DISP=OLD
//DD1      DD  UNIT=disk,VOLUME=SER=111111,DISP=OLD
//DD2      DD  UNIT=(disk,,DEFER),DISP=OLD,
//          VOLUME=(PRIVATE,,SER=(333333))
//DD3      DD  UNIT=disk,VOLUME=SER=222222,DISP=OLD
//SYSIN    DD  *
      MOVE  PDS=PDSSET1,T0=disk=222222,FROM=disk=333333
      MOVE  PDS=PDSSET2,T0=disk=222222,FROM=disk=222222
      MOVE  PDS=PDSSET3,T0=disk=222222,FROM=disk=444444,UNLOAD
/*

```

PDSSET1, PDSSET2, and PDSSET3 are already allocated on the receiving volume. PDSSET3 is allocated as a sequential data set; PDSSET1 and PDSSET2 are allocated as partitioned data sets. Since PDSSET3 is moved to a sequential data set, it is unloaded.

For a discussion of estimating space allocations, see [z/OS DFSMS Using Data Sets](#).

The DCB attributes of PDSSET3 are:

```
DCB=(RECFM=U,BLKSIZE=5000)
```

The unloaded attributes are:

```
DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)
```

The control statements are discussed as follows:

- SYSUT1 DD defines the device that is to contain the work data set.
- DD1 DD defines the system residence device.
- DD2 DD defines a device on which the source volumes are mounted as they are required.
- DD3 DD defines a device on which the receiving volume is mounted.
- SYSIN DD defines the control data set, which follows in the input stream.
- MOVE specifies move operations for the partitioned data sets and defines the source and receiving volumes for each data set.

Example 6: Unload Sequential Data Set onto Unlabeled Tape Volume

In this example, a sequential data set is unloaded onto a 9-track, unlabeled tape volume (800 bits per inch).

```
//UNLOAD    JOB    ...
//STEP1     EXEC   PGM=IEHMOVE
//SYSPRINT  DD     SYSOUT=A
//SYSUT1    DD     UNIT=disk,VOLUME=SER=222222,DISP=OLD
//DD1       DD     UNIT=disk,VOLUME=SER=111111,DISP=OLD
//DD2       DD     UNIT=disk,VOLUME=SER=222222,DISP=OLD
//TAPEOUT   DD     UNIT=tape,VOLUME=SER=SCRCH2,DISP=OLD,
//           DCB=(DEN=2,RECFM=FB,LRECL=80,BLKSIZE=800),
//           LABEL=(,NL)
//SYSIN     DD     *
              MOVE  DSN=SEQSET1,TO=tape=SCRCH2,
FROM=disk=222222,TODD=TAPEOUT
/*
```

The control statements are discussed as follows:

- SYSUT1 DD defines the device that is to contain the work data set.
- DD1 DD defines the system residence device.
- DD2 DD defines a device on which the source volume is mounted.
- TAPEOUT DD defines a device on which the receiving tape volume is mounted. This statement also provides label and mode information.
- SYSIN DD defines the control data set, which follows in the input stream.
- MOVE moves the sequential data set SEQSET1 from a disk volume to the receiving tape volume. The data set is unloaded. The TODD parameter in this statement refers to the TAPEOUT DD statement for label and mode information.

Example 7: Load Unloaded Sequential Data Sets from Labeled Tape

In this example, three unloaded sequential data sets are loaded from a labeled, 7-track tape volume (556 bits per inch) to a disk volume. Space is allocated by IEHMOVE. The example assumes that the disk volume is capable of supporting the data sets in their original forms.

```
//LOAD      JOB    ...
//STEP1     EXEC   PGM=IEHMOVE
//SYSPRINT  DD     SYSOUT=A
//SYSUT1    DD     UNIT=disk,VOLUME=SER=222222,DISP=OLD
//DD1       DD     UNIT=disk,VOLUME=SER=111111,DISP=OLD
//DD2       DD     UNIT=disk,VOLUME=SER=222222,DISP=OLD
//TAPESETS  DD     UNIT=tape,VOLUME=SER=001234,DISP=OLD,
//           LABEL=(1,SL),DCB=(DEN=1,TRTCH=C)
//SYSIN     DD     *
              MOVE  DSN=UNLDSET1,TO=disk=222222,
FROM=tape=(001234,1),FROMDD=TAPESETS
              MOVE  DSN=UNLDSET2,TO=disk=222222,
FROM=tape=(001234,2),FROMDD=TAPESETS
              MOVE  DSN=UNLDSET3,TO=disk=222222,
FROM=tape=(001234,3),FROMDD=TAPESETS
/*
```

The control statements are discussed as follows:

- SYSUT1 DD defines the device that is to contain the work data set.
- DD1 DD defines the system residence device.
- DD2 DD defines a device on which the receiving volume is mounted.
- TAPESETS DD defines a device on which the source tape volume is mounted. DCB information is provided in this statement.
- SYSIN DD defines the control data set, which follows in the input stream.
- MOVE moves the unloaded data sets to the receiving volume.

To move a data set from a tape volume that contains more than one data set, you must specify the sequence number of the data set in the *list* field of the FROM parameter on the utility control statement.

Example 8: Move Cataloged Data Set Group

In this example, the cataloged data set group A.B.C, which comprises data set A.B.C.X, A.B.C.Y, and A.B.C.Z, is moved from two disk volumes onto a third volume. Space is allocated by IEHMOVE. The catalog is updated to refer to the receiving volume. The source data sets are scratched.

```
//MOVEDSG JOB ...
//STEP1 EXEC PGM=IEHMOVE
//SYSPRINT DD SYSOUT=A
//SYSUT1 DD UNIT=disk,VOLUME=SER=222222,DISP=OLD
//DD1 DD UNIT=disk,VOLUME=SER=111111,DISP=OLD
//DD2 DD UNIT=disk,VOLUME=SER=222222,DISP=OLD
//DD3 DD UNIT=disk,VOLUME=SER=333333,DISP=OLD
//DD4 DD UNIT=disk,VOLUME=SER=444444,DISP=OLD
//SYSIN DD *
MOVE DSGROUP=A.B.C,T0=disk=222222
/*
```

The control statements are discussed as follows:

- SYSUT1 DD defines the device that is to contain the work data set.
- DD1 DD defines the system residence device.
- DD2 DD defines a device on which the receiving volume is mounted.
- DD3 DD defines a device on which one of the source volumes is mounted.
- DD4 DD defines a device on which one of the source volumes is mounted.
- SYSIN DD defines the control data set, which follows in the input stream.
- MOVE moves the specified data sets to volume 222222.

This example can be used to produce the same result without the use of the DD4 DD statement, using one less mountable disk device. With DD3 and DD4, both of the source volumes are mounted at the start of the job. With DD3 only, the 333333 volume is mounted at the start of the job. After the 333333 volume is processed, the utility requests that the operator mount the 444444 volume. In this case, the DD3 statement is coded:

```
//DD3 DD UNIT=(disk,,DEFER),DISP=OLD,
// VOLUME=(PRIVATE,,SER=(333333))
```

Chapter 16. IEHPROGM (Program Maintenance) Program

IEHPROGM is a system utility that is used to modify system control data and to maintain data sets at an organizational level. IEHPROGM should only be used by those programmers who are locally authorized to do so.

IEHPROGM does not support dynamic UCBs while processing data sets that are password-protected.

You can use IEHPROGM to perform the following tasks:

- Scratch (delete) a data set or a member of a partitioned data set.
- Rename a data set or a member of a partitioned data set.
- Maintain data set passwords.

You must have RACF authority in order to use IEHPROGM.

IDCAMS is recommended for use with SMS managed data sets.

You can write an assembler program to perform any of the IEHPROGM functions.

Related reading:

- For information on IDCAMS, see *z/OS DFSMS Access Method Services Commands*.
- For more information, see *z/OS DFSMSdfp Advanced Services* and *z/OS DFSMS Using Data Sets*.
- For information on RACF requirements for the Storage Management Subsystem, see *z/OS DFSMSdfp Storage Administration*.

Scratching or Renaming a Data Set or Member

IEHPROGM can be used to scratch the following data sets from a DASD volume or volumes:

- Sequential, BDAM, or partitioned data sets or PDSE. They can be data sets that are named by the operating system.
- Members of a partitioned data set.
- A temporary VSAM data set.

A data set is considered scratched when its data set control block is removed from the volume table of contents (VTOC) of the volume on which it resides; its space is made available for reallocation.

A member is considered scratched when its name is removed from the directory of the partitioned data set in which it is contained.

For partitioned data sets that are not PDSEs, the space occupied by a scratched member is not available for reallocation until the partitioned data set is scratched or compressed. (When scratching a member of a partitioned data set, all aliases of that member should also be removed from the directory.)

On SCRATCH requests, the presence of the PURGE or NOPURGE keyword may be ignored for SMS managed data sets. The use of the PURGE and NOPURGE keywords is unchanged for non-SMS managed data sets.

- When OVRD_EXPDT(NO) is specified in the IGDSMSxx member of SYS1.PARMLIB or the OVRD_EXPDT keyword is not specified, the PURGE and NOPURGE keywords are honored.
- When OVRD_EXPDT(YES) is specified in the IGDSMSxx member of SYS1.PARMLIB, the PURGE and NOPURGE keywords are **not** honored. The data set is **always** deleted, whether or not it has expired. This is true only if the data set is a DASD data set and SMS managed.

When scratching or renaming a data set managed by the Storage Management Subsystem (SMS), the device type and volumes list on the VOL parameter must reflect the volume actually allocated to the data

set. This is a restriction for both SMS and non-SMS managed data sets. However, when you specify a volume when allocating an SMS-managed data set, SMS will not automatically allocate the data set on that volume.

When scratching an SMS-managed data set, IEHPROGM will uncatalog that data set.

You should use IDCAMS DELETE VR to delete uncataloged data sets on SMS managed volumes. If you attempt to scratch and uncataloged data set on an SMS-managed volume, IEHPROGM will ONLY scratch (an uncatalog) a cataloged version of the data set, if one exists. When the specified volume in IEHPROGM is found to be SMS managed, a Catalog locate is used to identify a volume containing the data. The Catalog locate may return a different volume than specified in IEHPROGM resulting in the wrong data set to be scratched.

When scratching or renaming a data set the device type and volumes list on the VOL parameter must reflect the volume actually allocated to the data set.

IEHPROGM will **not** scratch the data set containing the index for an indexed VTOC.

IEHPROGM can be used to rename a data set or member that resides on a DASD volume. In addition, the program can be used to change any member aliases.

When renaming an SMS-managed data set, IEHPROGM will uncatalog the data set and then catalog the data set under its new name in the appropriate catalog. If uncataloging cannot be done, because of an alias, IEHPROGM will not rename the data set.

Temporary VSAM data sets can be scratched using SCRATCH VTOC,SYS.

If RACF is active, ALTER authorization is required to scratch a RACF-defined data set, or rename a data set, and UPDATE authorization is required to scratch or rename a member of a partitioned data set.

Note: RACF, an IBM security package, will not allow you to rename a data set that is covered only by a generic profile to a name that will not be covered by a generic profile because this would allow you to unprotect the data set.

Maintaining Data Set Passwords

IEHPROGM can be used to maintain non-VSAM password entries in the PASSWORD data set and to alter the protection status of DASD data sets in the data set control block (DSCB). This topic also explains why data set passwords provide poor security and why IBM recommends z/OS Security Server (RACF).

A data set can have one of three types of password protection, as indicated in the DSCB for DASD data sets and in the tape label for tape data sets.

The possible types of data set password protection are:

- No protection, which means that no passwords are required to read or write the data set.
- Read/write protection, which means that a password is required to read or write the data set.
- Read-without-password protection, which means that a password is required only to write the data set; the data set can be read without a password.

If a system data set is password protected and a problem occurs on the data set, maintenance personnel must be provided with the password in order to access the data set and resolve the problem.

A data set can have one or more passwords assigned to it; each password has an entry in the PASSWORD data set. A password assigned to a data set can allow read and write access, or only read access to the data set.

Figure 44 on page 297 shows the relationship between the protection status of data set ABC and the type of access allowed by the passwords assigned to the data set. Passwords ABLE and BAKER are assigned to data set ABC. If no password protection is set in the DSCB or tape label, data set ABC can be read or written without a password. If read/write protection is set in the DSCB or tape label, data set ABC can be read with either password ABLE or BAKER and can be written with password ABLE. If read-without-password protection is set in the DSCB or tape label, data set ABC can be read without a password and can be written with password ABLE; password BAKER is never needed.

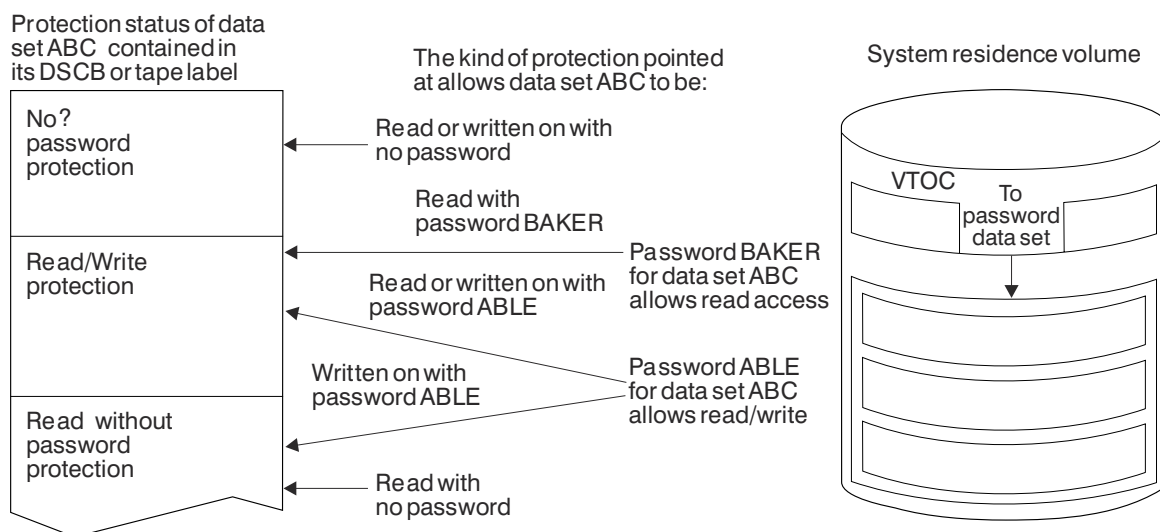


Figure 44. Relationship between the protection status of a data set and its passwords

Before IEHPROGM is used to maintain data set passwords, the PASSWORD data set must reside on the system residence volume. IEHPROGM can then be used to:

- Add an entry to the PASSWORD data set.
- Replace an entry in the PASSWORD data set.
- Delete an entry from the PASSWORD data set.
- Provide a list of information from an entry in the PASSWORD data set.

Each entry in the PASSWORD data set contains the name of the protected data set, the password, the protection mode of the password, an access counter, and 77 bytes of optional user data. The protection mode of the password defines the type of access allowed by the password and whether the password is a control password or secondary password. The initial password, added to the PASSWORD data set for a particular data set, is marked in the entry as the control password for that data set. The second and subsequent passwords added for the same data set are marked as secondary passwords.

For DASD data sets, IEHPROGM updates the protection status in the DSCB when a control password entry is added, replaced, or deleted. This permits setting and resetting the protection status of an existing DASD data set at the same time its passwords are added, replaced, or deleted. IEHPROGM automatically alters the protection status of a data set in the DSCB if the following conditions are met:

- The control password for the data set is being added, replaced, or deleted.
- The data set is online.
- The volume on which the data set resides is specified on the utility control statement, or the data set is cataloged.
- The data set is not allocated within the IEHPROGM job.

For tape data sets, IEHPROGM cannot update the protection status in the tape label when a password entry is added, replaced, or deleted. Protection status in a tape label must be set with JCL.

Passwords to be added, replaced, deleted, or listed can be specified on utility control statements or can be entered by the console operator. IEHPROGM issues a message to the console operator when a password on a utility control statement is either missing or invalid. The message contains the job name, step name, and utility control statement name and identifies the particular password that is missing or invalid. Two invalid passwords are allowed per password entry on each utility control statement before the request is ignored; a total of five invalid passwords is allowed for the password entries on all the utility control statements in a job step before the step is canceled.

Related reading:

- For a complete description of data set passwords and the PASSWORD data set, see [z/OS DFSMSdfp Advanced Services](#).
- For the contents of the DSCB, see [z/OS DFSMSdfp Advanced Services](#).
- For a description of tape labels, see [z/OS DFSMS Using Magnetic Tapes](#).

Adding Data Set Passwords

When a password is added for a data set, an entry is created in the PASSWORD data set with the specified data set name, password name, protection mode of the password (read/write or read only), and the optional 77 characters of user-supplied data. The access counter in the entry is set to zero.

The control password for a data set must always be specified to add, replace, or delete secondary passwords. The control password should not be specified, however, to list information from a secondary password entry.

Secondary passwords can be assigned to a data set to restrict some users to reading the data set or to record the number of times certain users access the data set. The access counter in each password entry provides a count of the number of times the password was used to successfully open the data set.

If a control password for an online DASD data set is added, the protection status of the data set (read/write or read-without-password) is set in the DSCB.

Replacing Data Set Passwords

Any of the following information may be replaced in a password entry: the password, protection mode (read/write or read only) of the password, and the 77 characters of user data. The protection status of a data set can be changed by replacing the control entry for the data set.

If the control entry of an online DASD data set is replaced, the DSCB is also reset to indicate any change in the protection status of the data set. Therefore, you should ensure that the volume is online when changing the protection status of a DASD data set.

Deleting Data Set Passwords

When a control password entry is deleted from the PASSWORD data set, all secondary password entries for that data set are also deleted. However, when a secondary entry is deleted, no other password entries are deleted.

If the control password entry is deleted for an online DASD data set, the protection status of the data set in the DSCB is also changed to indicate no protection. When deleting a control password for a DASD data set, the user should ensure that the volume is online. If the volume is not online, the password entry is removed, but data set protection is still indicated in the DSCB; the data set cannot be accessed unless another password is added for that data set.

If the control password entry is deleted for a tape data set, the tape volume cannot be accessed unless another password is added for that data set.

The delete function should be used to delete all the password entries for a scratched data set to make the space available for new entries.

Listing Password Entries

A list of information from any entry in the PASSWORD data set can be obtained in the SYSPRINT data set by providing the password for that entry. The list includes: the number of times the password has been used to successfully open the data set; the type of password (control password or secondary password) and type of access allowed by the password (read/write or read-only); and the user data in the entry. The following example shows a sample list of information printed from a password entry.

```
DECIMAL ACCESS COUNT= 000025
PROTECT MODE BYTE= SECONDARY, READ ONLY
USER DATA FIELD= ASSIGNED TO J. BROWN
```

Input and Output

IEHPROGM uses the following input:

- One or more data sets containing system control data to be modified.
- A control data set that contains utility control statements used to control the functions of the program.

IEHPROGM produces the following output:

- A modified object data set or volumes.
- A message data set that contains error messages and information from the PASSWORD data set.

If IEHPROGM is invoked from an application program, you can dynamically allocate the devices and data sets by issuing SVC 99 before calling IEHPROGM.

Related reading: For information about IEHPROGM return codes, see [Appendix A, “Invoking Utility Programs from an Application Program,”](#) on page 315.

Control

IEHPROGM is controlled by job and utility control statements.

You can use job control statements to perform these tasks:

- Process or load the program.
- Define the control data set.
- Define volumes or devices to be used during the course of program execution.
- Prevent data sets from being deleted inadvertently.
- Prevent volumes from being demounted before they have been completely processed by the program.
- Suppress the listing of utility control statements.

Utility control statements are used to control the functions of the program and to define those data sets or volumes that are to be modified.

Job Control Statements

Table 58 on page 299 shows the job control statements for IEHPROGM.

Table 58. IEHPROGM job control statements

Statement	Use
JOB	Starts the job.
EXEC	Specifies the program name (PGM=IEHPROGM) or, if the job control statements reside in a procedure library, the procedure name. Additional PARM information can be specified to control the number of lines per page on the output listing and to suppress printing of utility control statements. See “EXEC Statement” on page 299.
SYSPRINT DD	Defines a sequential data set for messages.
anyname DD	Defines a permanently mounted or mountable volume.
SYSIN DD	Defines the control data set. The control data set normally follows the job control statements in the input stream; however, it can be defined as a member of a procedure library.

EXEC Statement

The syntax of the EXEC statement is:

Label	Statement	Parameters
// [stepname]	EXEC	PGM=IEHPROGM [, PARM=[LINECNT=xx] [, <u>PRINT</u> NOPRINT]]

where:

PGM=IEHPROGM

specifies that you want to run IEHPROGM.

PARM=[LINECNT=xx] [, PRINT | NOPRINT]

specifies what should be contained in the SYSPRINT data set, and how the printed output of SYSPRINT should be formatted. If more than one subparameter is coded with PARM, the subparameters must be enclosed in parentheses or single quotes.

LINECNT=xx

specifies the number of lines per page to be printed in the listing of the SYSPRINT data set. The value xx can be from 01 to 99. The default is 45.

PRINT | NOPRINT

specifies whether utility control statements are to be included in the SYSPRINT listing. Suppressing printing of utility control statements assures that passwords assigned to data sets remain confidential. However, suppressing printing may make it difficult to interpret error messages, because the relevant utility control statement is not printed before the message.

The default is to print all control statements.

SYSPRINT DD Statement

The block size for the SYSPRINT data set must be a multiple of 121. Any blocking factor can be specified.

anyname DD Statement

A DD statement must be included for each permanently mounted or mountable volume referred to in the job step. These DD statements are used as device allocation statements, rather than as true data definition statements.

This DD statement can be entered:

```
//anyname DD UNIT=xxxx, VOLUME=SER=xxxxxx,
DISP=OLD
```

The UNIT and VOLUME parameters define the device type and volume serial number. The DISP=OLD specification prevents the inadvertent creation of a data set.

Because IEHPROGM modifies the internal control blocks created by device allocation DD statements, the DSNAME parameter, if supplied, will be ignored by IEHPROGM. (All data sets are defined explicitly or implicitly by utility control statements.)

Note: Unpredictable results may occur in multitasking environments where dynamic allocation/deallocation of devices, by other tasks, causes changes in the TIOT during IEHPROGM execution.

To specify deferred mounting with mountable volumes, code:

```
//anyname DD VOLUME=(PRIVATE, SER=xxxxxx),
// UNIT=(xxxx, DEFER), DISP=OLD
```

Unit affinity cannot be used on DD statements defining mountable devices.

SYSIN DD Statement

The block size for the SYSIN data set must be a multiple of 80. Any blocking factor can be specified.

Utility Control Statements

Table 59 on page 301 shows the utility control statements for IEHPROGM.

Table 59. IEHPROGM utility control statements

Statement	Use
SCRATCH	Scratches a data set or a member from a DASD volume.
RENAME	Changes the name or alias of a data set or member residing on a DASD volume.
CATLG	Generates an entry in the index of a catalog.
UNCATLG	Removes an entry from the catalog.
ADD	Adds a password entry in the PASSWORD data set.
REPLACE	Replaces information in a password entry.
DELETEP	Deletes an entry in the PASSWORD data set.
LIST	Formats and lists information from a password entry.

Continuation requirements for utility control statements are described in [“Continuing utility control statements”](#) on page 8.

SCRATCH Statement

The SCRATCH statement is used to scratch a data set or member from a DASD volume. A data set or member is scratched only from the volumes designated in the SCRATCH statement. This function does not delete catalog entries for scratched data sets.

A SCRATCH operation will not be processed if the data set or volume is being used by a program executing concurrently. "DISP=OLD" on the DD statement only prevents the inadvertent deletion of a data set. It does not ensure exclusive use of the data set during execution of the job step. When scratching a member of a partitioned data set, it is your responsibility to ensure that the data set is not currently in use.

For multivolume data sets, all volumes specified must be online.

When scratching a data set managed by the Storage Management Subsystem (SMS), care must be taken to ensure that the device type and volumes list on the VOL parameter reflects the volume actually allocated to the data set. When you specify a volume when creating an SMS-managed data set, SMS will not automatically allocate the data set on that volume.

When scratching an SMS-managed data set, IEHPROGM will uncatalog that data set.

The syntax of the SCRATCH statement is shown as follows.

Label	Statement	Parameters
[label]	SCRATCH	{VTOC DSNAME=name} , VOL=device=(list) [, PURGE] [, MEMBER=name] [, SYS]

where:

VTOC

specifies that all data sets on the designated volume be scratched, except for

- a data set that is protected by a password
- a data set whose expiration date has not passed
- a data set that contains the index for an indexed VTOC

Password-protected data sets are scratched if the correct password is provided.

The effect of VTOC is modified when it is used with PURGE or SYS.

DSNAME=name

specifies the fully qualified name of the data set to be scratched or the partitioned data set that contains the member to be scratched. The name must not exceed 44 characters, including delimiters.

VOL=device=(list)

specifies the device type and serial numbers of the volumes, limited to 50, that contain the data sets. If only one serial number is listed, it need not be enclosed in parentheses. Multiple serial numbers should be separated with commas.

If VTOC or MEMBER is specified, VOL cannot specify more than one volume. Caution should be used when specifying VTOC if VOL specifies the system residence volume.

PURGE/NOPURGE

specifies whether data sets designated by DSNAME or VTOC are scratched.

The presence of the PURGE or NOPURGE keyword may be ignored for SMS managed data sets. When OVRD_EXPDT is specified in the IGDSMSxx member of SYS1.PARMLIB, the PURGE and NOPURGE keywords are **not** honored. The data set is **always** deleted, whether or not it has expired. This is true only if the data set is a DASD data set and SMS managed. The use of the PURGE and NOPURGE keywords is unchanged for non-SMS managed data sets.

MEMBER=name

specifies a member name or alias of a member (in the named data set) to be removed from the directory of a partitioned data set. This name is not validity-checked because all members must be accessible, whether the name is valid or not.

Default: The entire data set or volume of data sets specified by name is scratched.

SYS

limits the action of SCRATCH VTOC so that only temporary data sets are erased. This means data sets whose names were coded beginning with an ampersand on JCL. An example is

```
DSN=&LIB
```

Temporary data sets have names beginning with "AAAAAAAA.AAAAAAAAA.AAAAAAAAA.AAAAAAAAA." or "SYSnnnnn.T" with "F", "V", or "A" in position 19. These are names assigned to the data sets by the operating system.

If the name of the data set is in this form, it is likely to be a temporary data set which was not erased at normal step or job termination; nnnnn is the date the data set was created in yyddd format.

The SYS parameter is valid only when VTOC is specified.

RENAME Statement

The RENAME statement is used to change the true name or alias of a data set or member residing on a DASD volume. The name is changed only on the designated volumes. The rename operation does not update the catalog.

A RENAME operation will not be processed if the data set or volume is being used by a program executing concurrently. When renaming a member of a partitioned data set, it is your responsibility to ensure that the data set is not currently in use.

For multivolume data sets, all volumes specified must be online.

If you do not code the MEMBER parameter, the entire data set is renamed.

When renaming a data set managed by the Storage Management Subsystem (SMS), care must be taken to ensure that the device type and volumes list on the VOL parameter reflects the volume actually allocated

to the data set. When you specify a volume when allocating an SMS-managed data set, SMS will not automatically allocate the data set on that volume.

When renaming SMS-managed data sets, IEHPROGM will uncatalog the data set and recatalog the data set under the new name. If recataloging is necessary, but cannot be done (because of an alias), IEHPROGM will not rename the data set.

The syntax of the RENAME statement is:

Label	Statement	Parameters
[<i>label</i>]	RENAME	DSNAME= <i>name</i> , VOL= <i>device</i> =(<i>list</i>) , NEWNAME= <i>name</i> [, MEMBER= <i>name</i>]

where:

DSNAME=*name*

specifies the fully qualified name of the data set to be renamed or the partitioned data set that contains the member to be renamed. The name must not exceed 44 characters, including delimiters.

VOL=*device*=(*list*)

specifies the device type and serial numbers of the volumes, limited to 50, that contain the data sets. If only one serial number is listed, it need not be enclosed in parentheses. Multiple serial numbers should be separated with commas.

If MEMBER is specified, VOL cannot specify more than one volume.

NEWNAME=*name*

specifies the new fully qualified name or alias name for the data set or the new member.

MEMBER=*name*

specifies a member name or alias of a member (in the named data set) to be renamed. This name is not validity-checked because all members must be accessible, whether the name is valid or not.

Default: The entire data set or volume of data sets specified by name is changed.

CATLG and UNCATLG Statements

The CATLG statement is used to generate a non-VSAM entry in a catalog.

The UNCATLG statement is used to remove a non-VSAM entry from the catalog.

You cannot use IEHPROGM to catalog or uncatalog data sets (except by renaming or scratching them) which are SMS-managed. IEHPROGM can only be used to catalog or uncatalog non-VSAM data sets which are not SMS-managed. To catalog or uncatalog SMS managed data sets, see [z/OS DFSMS Managing Catalogs](#) and [z/OS DFSMS Access Method Services Commands](#).

The syntax of the CATLG and UNCATLG statements is:

Label	Statement	Parameters
[<i>label</i>]	{CATLG UNCATLG}	DSNAME= <i>name</i> , VOL= <i>device</i> = <i>{ (list) </i> <i>(serial,seqno [, . . .]) }</i>
Note: VOL can only be coded with CATLG		

where:

DSNAME=name

specifies the fully qualified name of the data set to be cataloged or uncataloged. The name must not exceed 44 characters, including delimiters.

VOL=device=({(list)| (serial,seqno[,...])}

specifies the device type, serial numbers, and data set sequence numbers (for tape volumes) of the volumes (up to 50) that contain the data sets to be cataloged in the catalog.

Always use generic device names (for instance, 3390) for *device*.

The volume serial numbers must appear in the same order in which they were originally encountered (in DD statements within the input stream) when the data set was created. Multiple serial numbers should be separated with commas.

Seqno is valid only for data sets which reside on tape. This value is meaningful for only the first volume serial. The maximum value that you can code in z/OS V1R5 or later is 65535. If you expect to use this catalog entry on an earlier level of the system than that, do not code a value that exceeds 9999. For more information on file sequence numbers, see [z/OS DFSMS Using Magnetic Tapes](#).

VOL can only be coded with CATLG.

ADD (Add a Password) and REPLACE (Replace a Password) Statements

The ADD statement is used to add a password entry in the PASSWORD data set. When the control entry for an online DASD data set is added, the indicated protection status of the data set is set in the DSCB; when a secondary entry is added, the protection status in the DSCB is not changed.

The REPLACE statement is used to replace any or all of the following information in a password entry: the password name, protection mode (read/write or read only) of the password, and user data. When the control entry for an online DASD data set is replaced, the protection status of the data set is changed in the DSCB if necessary; when a secondary entry is replaced, the protection status in the DSCB is not changed.

The syntax of the ADD and REPLACE statements is:

Label	Statement	Parameters
[label]	{ADD REPLACE}	DSNAME= <i>name</i> [, PASSWORD1= <i>current-password</i>] [, PASSWORD2= <i>new-password</i>] [, CPASSWORD= <i>control-password</i>] [, TYPE= <i>code</i>] [, VOL= <i>device</i> =({(list)}) [, DATA= ' <i>user-data</i> ']
Note: PASSWORD1 can only be coded with REPLACE		

where:

DSNAME=name

specifies the fully qualified name of the data set whose password entry is to be added or changed. The name must not exceed 44 characters, including delimiters.

PASSWORD1=current-password

specifies the password in the entry to be changed. If PASSWORD1 is not coded, the operator is prompted for the current password. PASSWORD1 can only be coded with REPLACE.

PASSWORD2=new-password

specifies the new password to be added or assigned to the entry. If the password is not to be changed, the current password must also be specified as the new password. The password can consist of 1 to 8 alphanumeric characters. If PASSWORD2 is not coded, the operator is prompted for a new password.

CPASSWORD=control-password

specifies the control password for the data set.

For ADD, CPASSWORD must be specified unless this is the first password assigned to the data set, in which case PASSWORD2 specifies the password to be added.

For REPLACE, CPASSWORD must be specified unless the control entry is being changed, in which case PASSWORD1 specifies the control password.

TYPE=code

specifies the protection code of the password and, if a control password entry is to be changed for or assigned to a BDAM online data set, specifies the protection status of the data set. The values that can be specified for *code* are:

- 1**
specifies that the password is to allow both read and write access to the data set; if a control password is being assigned or changed, read/write protection is set in the DSCB.
- 2**
specifies that the password is to allow only read access to the data set; if control password is being assigned or changed, read/write protection is set in the DSCB.
- 3**
specifies that the password is to allow both read and write access to the data set; if a control password is being assigned or changed, read-without-password protection is set in the DSCB.

Default: For ADD, if this parameter is omitted, the new password is assigned the same protection code as the control password for the data set. If a control password is being "added", TYPE=3 is the default. For REPLACE, the protection is not changed.

VOL=device=(list)

specifies the device type and serial numbers of the volumes, limited to 50, that contain the data sets. If only one serial number is listed, it need not be enclosed in parentheses. Multiple serial numbers should be separated with commas.

If omitted, the protection status in the DSCB is not set or changed, unless the data set is cataloged and online. This parameter is not necessary for secondary password entries, or if the desired protection status in the DSCB is already set or is not to be changed.

DATA='user-data'

specifies the user data to be placed in the password entry. The user data has a maximum length of 77 bytes and must be enclosed in apostrophes. Any other apostrophes contained within the user data must be entered as two single apostrophes.

If DATA is omitted from an ADD operation, 77 blanks are used. If DATA is omitted from a REPLACE operation, current user data is not changed.

DELETEP (Delete a Password) Statement

The DELETEP statement is used to delete an entry in the PASSWORD data set. If a control entry is deleted, all the secondary entries for that data set are also deleted. If a secondary entry is deleted, only that entry is deleted. When the control entry for an online DASD data set is deleted, the protection status in the DSCB is set to indicate that the data set is no longer protected.

The syntax of the DELETEP statement is:

Label	Statement	Parameters
[label]	DELETEP	DSNAME=name [, PASSWORD1=current-password] [, CPASSWORD=control-password] [, VOL=device=(list)]

where:

DSNAME=name

specifies the fully qualified name of the data set whose password entry is to be deleted. The name must not exceed 44 characters, including delimiters.

PASSWORD1=current-password

specifies the password in the entry to be deleted. If PASSWORD1 is not coded, the operator is prompted for the current password.

CPASSWORD=control-password

CPASSWORD must be specified unless the control entry is being deleted, in which case PASSWORD1 specifies the control password.

VOL=device=(list)

specifies the device type and serial numbers of the volumes, limited to 50, that contain the data sets. If only one serial number is listed, it need not be enclosed in parentheses. Multiple serial numbers should be separated with commas.

If omitted, the protection status in the DSCB is not changed, unless the data set is cataloged and online. This parameter is not necessary for secondary password entries, or if the desired protection status in the DSCB is already set.

LIST (List Information from a Password) Statement

The LIST statement is used to format and print information from a password entry.

The syntax of the LIST statement is:

Label	Statement	Parameters
[label]	LIST	DSNAME=name , PASSWORD1=current-password

where:

DSNAME=name

specifies the fully qualified name of the data set whose password entry is to be listed. The name must not exceed 44 characters, including delimiters.

PASSWORD1=current-password

specifies the password in the entry to be listed. If PASSWORD1 is not coded, the operator is prompted for the current password.

IEHPROGM Examples

The following examples illustrate some of the uses of IEHPROGM. Table 60 on page 306 can be used as a quick-reference guide to IEHPROGM examples. The numbers in the "Example" column point to the examples that follow.

Table 60. IEHPROGM example directory

Operation	Mount Volumes	Comments	Example
LIST, REPLACE	Disk	A password entry is listed. Protection mode and status are changed, and user data is added.	6
RENAME	Disk	A member of a partitioned data set is renamed.	7
RENAME, DELETEP, ADD	Disk	A data set is renamed. The old passwords are deleted and new passwords are assigned.	5
RENAME, UNCATLG, CATLG	Disk	A data set is renamed on two mountable devices; the old data set name is removed. The data set is cataloged under its new name.	3
SCRATCH	Disk	The data sets' DSCB is scratched.	1
SCRATCH, UNCATLG	Disk	Two data sets are scratched and their entries removed from the catalog.	2

Table 60. IEHPROGM example directory (continued)

Operation	Mount Volumes	Comments	Example
UNCATLG	Disk	Index structures for three generation data sets are deleted from the catalog.	4

Examples that use **disk** or **tape** in place of actual device names or numbers must be changed before use. The actual device names or numbers depend on how your installation has defined the devices to your system.

For information about generation data sets and groups, see [Processing Generation Data Groups in z/OS DFSMS Using Data Sets](#).

Example 1: Scratch Temporary System Data Sets

In this example, all temporary system data sets are scratched from the volume table of contents.

```
//SCRVTOC JOB ...
//STEP1 EXEC PGM=IEHPROGM
//SYSPRINT DD SYSOUT=A
//DD2 DD UNIT=disk,VOLUME=SER=222222,DISP=OLD
//SYSIN DD *
SCRATCH VTOC,VOL=disk=222222,SYS
/*
```

The control statements are discussed as follows:

- The DD2 statement defines a volume. Because the system residence volume is not referred to, a DD statement is needed to define it.
- The SCRATCH statement, with SYS specified, indicates that all temporary system data sets whose expiration dates have expired are scratched from the specified volume.

Example 2: Scratch and Uncatalog Two Data Sets

In this example, two data sets are scratched: SET1 and A.B.C.D.E are scratched from volume 222222. Both data sets are uncataloged.

```
//SCRDSETS JOB ...
//STEP1 EXEC PGM=IEHPROGM
//SYSPRINT DD SYSOUT=A
//DD1 DD UNIT=disk,VOLUME=SER=111111,DISP=OLD
//DD2 DD UNIT=disk,DISP=OLD,VOLUME=SER=222222
//SYSIN DD *
SCRATCH DSNAME=SET1,VOL=disk=222222
UNCATLG DSNAME=SET1
SCRATCH DSNAME=A.B.C.D.E,VOL=disk=222222
UNCATLG DSNAME=A.B.C.D.E
/*
```

The utility control statements are discussed as follows:

- The first SCRATCH statement specifies that SET1, which resides on volume 222222, is scratched.
- The first UNCATLG statement specifies that SET1 is uncataloged.
- The second SCRATCH statement specifies that A.B.C.D.E, which resides on volume 222222, is scratched.
- The second UNCATLG statement specifies that A.B.C.D.E is uncataloged.

Example 3: Rename a Multi-Volume Data Set Catalog

In this example, the name of a data set is changed on two mountable volumes. The old data set name is removed and the data set is cataloged under its new data set name.

```
//RENAMEDS JOB ...
//STEP1 EXEC PGM=IEHPROGM
//SYSPRINT DD SYSOUT=A
//DD1 DD VOLUME=SER=111111,UNIT=disk,DISP=OLD
//DD2 DD UNIT=(disk,,DEFER),DISP=OLD,
// VOLUME=(PRIVATE,SER=(222222,333333))
//SYSIN DD *
RENAME DSN= A.B.C,NEWNAME=NEWSET,VOL=disk=(222222,333333)
UNCATLG DSN= A.B.C
CATLG DSN= NEWSET,VOL=disk=(222222,333333)
/*
```

The control statements are discussed as follows:

- RENAME specifies that data set A.B.C, which resides on volumes 222222 and 333333, is to be renamed NEWSET.
- UNCATLG specifies that data set A.B.C is uncataloged.
- CATLG specifies that NEWSET, which resides on volumes 222222 and 333333, is cataloged.

Example 4: Uncatalog Three Data Sets

In this example, three data sets, A.B.C.D.E.F.SET1, A.B.C.G.H.SET2, and A.B.I.J.K.SET3, are uncataloged.

```
//DLTSTRUC JOB ...
//STEP1 EXEC PGM=IEHPROGM
//SYSPRINT DD SYSOUT=A
//DD1 DD UNIT=disk,VOLUME=SER=111111,DISP=OLD
//SYSIN DD *
UNCATLG DSN= A.B.C.D.E.F.SET1
UNCATLG DSN= A.B.C.G.H.SET2
UNCATLG DSN= A.B.I.J.K.SET3
/*
```

The control statements are discussed as follows:

- The UNCATLG statements specify that data sets A.B.C.D.E.F.SET1, A.B.C.G.H.SET2, and A.B.I.J.K.SET3 are uncataloged.

Example 5: Rename a Data Set and Define New Passwords

In this example, a data set is renamed. The data set passwords assigned to the old data set name are deleted. Then two passwords are assigned to the new data set name. If the data set is not cataloged, a message is issued indicating that the LOCATE macro instruction ended unsuccessfully.

```
//ADDPASS JOB ... 72
//STEP1 EXEC PGM=IEHPROGM,PARM='NOPRINT'
//SYSPRINT DD SYSOUT=A
//DD1 DD VOLUME=SER=222222,DISP=OLD,
// UNIT=disk
//SYSIN DD *
RENAME DSN= OLD,VOL=disk=222222,NEWNAME=NEW
DELETEP DSN= OLD,PASSWORD1=KEY
ADD DSN= NEW,PASSWORD2=KEY,TYPE=1, X
DATA='SECONDARY IS READ'
ADD DSN= NEW,PASSWORD2=READ,CPASSWORD=KEY,TYPE=2, X
DATA='ASSIGNED TO J. DOE'
/*
```

The utility control statements are discussed as follows:

- RENAME specifies that the data set called OLD is renamed NEW. The operator is required to supply a password to rename the old data set.
- DELETEP specifies that the entry for the password KEY is deleted. Because KEY is a control password in this example, all the password entries for the data set name are deleted. The VOL parameter is not needed because the protection status of the data set as set in the DSCB is not to be changed; read/write protection is presently set in the DSCB, and read/write protection is desired when the passwords are reassigned under the new data set name.

- The ADD statements specify that entries are added for passwords KEY and READ. KEY becomes the control password and allows both read and write access to the data set. READ becomes a secondary password and allows only read access to the data set. The VOL parameter is not needed, because the protection status of the data set is still set in the DSCB.

Example 6: List and Replace Password Information

In this example, information from a password entry is listed. Then the protection mode of the password, the protection status of the data set, and the user data are changed.

```

//REPLPASS JOB          ...                               72
//STEP1    EXEC PGM=IEHPROGM,PARM='NOPRINT'
//SYSPRINT DD SYSOUT=A
//DD1      DD UNIT=disk,VOLUME=SER=111111,DISP=OLD
//DD2      DD VOLUME=(PRIVATE,SER=(222222,333333)),
//          UNIT=(disk,,DEFER),DISP=OLD
//SYSIN    DD *
          LIST  DSNAME=A.B.C,PASWORD1=ABLE
          REPLACE DSNAME=A.B.C,PASWORD1=ABLE,PASWORD2=ABLE,TYPE=3,      X
              VOL=disk=(222222,333333),                                X
              DATA='NO SECONDARIES; ASSIGNED TO DEPT 31'
/*

```

The utility control statements are discussed as follows:

- LIST specifies that the access counter, protection mode, and user data from the entry for password ABLE are listed. Listing the entry permits the content of the access counter to be recorded before the counter is reset to zero by the REPLACE statement.
- REPLACE specifies that the protection mode of password ABLE is to be changed to allow both read and write access and that the protection status of the data set is changed to write-only protection. The VOL parameter is required because the protection status of the data set is changed and the data set, in this example, is not cataloged. Because this is a control password, the CPASWORD parameter is not required.

Example 7: Rename a Partitioned Data Set Member

In this example, a member of a partitioned data set is renamed.

```

//REN      JOB          ...
//STEP1    EXEC PGM=IEHPROGM
//SYSPRINT DD SYSOUT=A
//DD1      DD VOL=SER=222222,DISP=OLD,UNIT=disk
//SYSIN    DD *
          RENAME VOL=disk=222222,DSNAME=DATASET,NEWNAME=BC,MEMBER=ABC
/*

```

The control statements are discussed as follows:

- DD1 DD defines a permanently mounted volume.
- SYSIN DD defines the input data set, which follows in the input stream.
- RENAME specifies that member ABC in the partitioned data set DATASET, which resides on a disk volume, is renamed BC.

Chapter 17. IFHSTATR (List ESV Data) program

IFHSTATR is a system utility that formats and prints information from Type 21 SMF (system management facilities) records. These records provide error statistics by volume (ESV) data.

Refer to Macro IGESMF21 and *z/OS MVS System Management Facilities (SMF)* for the contents of type 21 SMF records.

Assessing the quality of tapes in a library

The statistics gathered by SMF in ESV records can be very useful for assessing the quality of a tape library. IFHSTATR prints ESV records in date/time sequence. You may find it useful to sort ESV records into volume serial number sequence, device address sequence, or into error occurrence sequence to help analyze the condition of the library.

The IFHSTATR report helps to identify deteriorating media (tapes); occasionally, poor performance from a particular tape drive can also be identified. The TAPE UNIT SERIAL may be used to identify the tape drive that wrote the tape.

An ESV record is written to the SMF data set:

1. When a volume is demounted
2. When a volume is demounted via DDR
3. When a tape drive is off-line
4. When a HALT EOD command is issued
5. When EREP is run

Because an ESV record may be written at other than demount time, more than one record may be written during the time a volume is mounted. Therefore, the number of records for a volume should not be used to determine the number of mounts or uses of a volume.

Input and output

IFHSTATR uses, as input, ESV records that contain error and usage information about magnetic tape volumes. If no ESV records are found, a message is written to the output data set. If the ESV record is not 62 or 80 bytes long, an INVALID TYPE 21 RECORD message is printed.

Run IFASMFDP to convert SYS1.MANX or SYS1.MANY from VSAM to physical sequential prior to using IFHSTATR to retrieve data from the ESV records.

IFHSTATR produces an output data set which contains information selected from ESV records. The output takes the form of 121-byte unblocked records, with an American National Standards Institute (ANSI) control character in the first byte of each record.

The input data set cannot have a block size that exceeds 32760 bytes. The tapes represented by the SMF records can have block sizes that exceed 32760 bytes.

If the block size field or the usage SIO count field in the Type 21 record exceeds 99,999, IFHSTATR scales the output. For example, if the field was greater than 99,999 but less than 1,000,000 it will be scaled to multiples of 1,000 and the letter "K" will be appended. If the field is greater than 999,999 it will be scaled to multiples of 1,000,000 and the letter "M" will be appended. Note that in this case the meanings of "K" and "M" differ from the meanings of "K" and "M" in the BLKSIZE value on the DD statement. On the DD statement, they mean multiples of 1024 and 1048576 respectively.

Related reading: For information about IFASMFDP, see *z/OS MVS System Management Facilities (SMF)*.

Figure 45 on page 312 shows a sample of printed output from IFHSTATR. The fields in the printed output are explained in the legend that follows.

MAGNETIC TAPE ERROR STATISTICS BY VOLUME 99172																			
VOLUME SERIAL	DATE	TIME OF DAY	DEV ADR	T/U SER	MODE	BLOCK SIZE	TAPE FMT	TEMP READ	TEMP READB	TEMP WRITE	PRM RD	PRM WRT	NOISE BLOCK	ERASE GAPS	CLEAN ACTS	USAGE SIO	MBYTES READ	MBYTES WRITTEN	
INVALID TYPE 21 RECORD																			
T34201	99172	08:04:22	0180	00000	OUT	N/A	N/A	1	N/A	2	3	4	5	6	7	8	N/A	N/A	
T34202	99172	12:01:59	0281	56789	OUT	80	1600	1	N/A	2	3	4	5	6	7	8	N/A	N/A	
T34200	99172	12:02:18	028C	67890	RB	32768	6250	255	N/A	255	255	255	255	65535	65535	65535	N/A	N/A	
T34200	99172	12:03:21	0480	78901	RB	80	N/A	1	2	3	4	5	N/A	6	7	8	9	10	
T34201	99172	12:04:21	0480	89012	RF	65535	N/A	65535	65535	65535	255	255	N/A	65535	65535	65535	65535	65535	
T35901	99172	14:35:41	09A0	72310	RF	2147M	N/A	0	0	0	0	1	N/A	0	0	999K	0	0	
T35904	11213	14:47:24	0FA4	06AE2	RF	N/A	N/A	0	0	0	0	0	N/A	0	0	3590	2147483392	2147483647	

* T342000 IS A 3420 WITH SMALL NUMBER OF ERRORS WITH BLOCKSIZE/DENSITY NOT AVAILABLE																			
* T342001 IS A 3420 WITH SMALL NUMBER OF ERRORS																			
* T342002 IS A 3420 WITH MAXIMUM NUMBER OF ERRORS																			
* T348000 IS A 3480 WITH SMALL NUMBER OF ERRORS																			
* T348001 IS A 3480 WITH MAXIMUM NUMBER OF ERRORS																			
* T359001 IS A 3590 WITH SIO COUNT GREATER THAN 99,999 BUT LESS THAN 1,000,000 AND A BLOCKSIZE GREATER THAN 999,999																			
* T359004 IS A 3590 WITH MAXIMUM NUMBER OF MEGABYTES WRITTEN																			

DA5U1044

Figure 45. Sample output from IFHSTATR

Legend

TIME OF DAY	The time the ESV record was written.
DEV ADR	The device address of the tape drive on which the tape was mounted
T/U SER	Serial number of the tape drive that wrote the tape, which is obtained from the tape label for input tapes if available.
MODE ¹	The OPEN flag bits for the data set being accessed. <ul style="list-style-type: none"> • OUT = OPENED for OUTPUT • RF = OPENED for INPUT forward • RB = OPENED for INPUT read backward
BLOCKSIZE ¹	The block size in the last data set accessed.
TAPE FORMAT ¹	The recording format of the tape.
TEMP READ	Number of read data checks that were successfully retried.
TEMP READB ²	Number of read data checks on read backward commands that were successfully retried.
TEMP WRITE	Number of write data checks that were successfully retried.
PERM RD	Number of read data checks that were not successfully retried.
PERM WRT	Number of write data checks that were not successfully retried.
NOISE BLOCK	(NRZI only) Number of read data checks that had the number of bytes read less than 12.
ERASE GAPS	Number of times an erase gap command was issued during error recovery. An erase gap command is issued before a retry of a write data check.
CLEAN ACTS	Number of times that, during read data check recovery, the tape was moved over the cleaner blade. This will normally be done after every fourth retry of the original read command.
USAGE SIO	Number of channel programs completed (channel programs started by ERP are not counted). Because a channel program has any number of CCWs, this may not be the count of the reads or writes.
MBYTES READ ²	Megabytes read.

Legend

MBYTES WRITTEN² Megabytes written.

Notes:

¹ Data originates in the DCB or DCBE and may not be available.

² Buffered tape units only

Control

IFHSTATR is controlled by job control statements. Utility control statements are not used.

Table 61 on page 313 shows the job control statements for IFHSTATR.

Table 61. IFHSTATR job control statements

Statement	Use
JOB	Starts the job.
EXEC	Specifies the program name (PGM=IFHSTATR).
SYSUT1 DD	Defines the input data set and the device on which it resides. The DSNAME and DISP parameters must be included. You need LABEL and DCB parameters if the device is a tape device without IBM standard labels. You need the UNIT and VOLUME parameters if the data set is not cataloged.
SYSUT2 DD	Defines the sequential data set on which the output is written.

IFHSTATR example

In this example, IFHSTATR is used to print out Type 21 SMF records.

```
//REPORT   JOB    ...
//STEP1    EXEC   PGM=IFHSTATR
//SYSUT1   DD     UNIT=3480,DSNAME=SYS1.MAN,LABEL=(,SL),
//          VOL=SER=valid,DISP=OLD
//SYSUT2   DD     SYSOUT=A
/*
```

The output data set can reside on any output device supported by BSAM.

Note: The input LRECL and BLKSIZE parameters are not specified by IFHSTATR. This information is taken from the DCB parameter on the SYSUT1 DD statement or from the tape label.

Appendix A. Invoking Utility Programs from an Application Program

This appendix documents Programming Interface and Associated Guidance Information provided by DFSMS.

This appendix is intended to help you invoke a utility program from an application program.

You can start a utility program through an application program by using the LINK macro instructions. (ATTACH may also be used, but additional parameters are needed.)

You must supply the information that is usually specified in the PARM parameter of the EXEC statement, and any nonstandard ddnames that define the data sets that you want the utility to use.

Note: All parameters must reside below the line (that is, have 24 bit addresses).

When invoking IEBCOMPR, IEBCOPY, IEBDG, IEBGENER, IEBPTCH, IEHLIST, IEHMOVE, or IEHPROGM from an application program or the TSO CALL command, you must dynamically allocate the device by issuing SVC 99 before calling the utility or you must use the JCL or TSO ALLOCATE equivalent.

IEHINITT, IEHMOVE, and IEHPROGM are APF (authorized program facility) programs. When executing an authorized program, the calling program must also be authorized. If you are using TSO, the TSO service routine IKJEFTSR may be used by an unauthorized program to invoke an authorized program such as IEHINITT.

The syntax of the LINK macro instruction is:

Label	Statement	Parameters
[<i>label</i>]	LINK	EP= <i>progrname</i> , PARAM=(<i>optionaddr</i> [, <i>ddnameaddr</i> [, <i>hdingaddr</i> [, <i>uexitparmaddr</i>]]) , VL=1

where:

EP=*progrname*

specifies the name of the utility program.

PARAM=(*optionaddr* [, *ddnameaddr* [, *hdingaddr* [, *uexitparmaddr*]])

specifies, as a sublist, address parameters to be passed from the application program to the utility program. All parameters and the parameter list itself must be in 24-bit addressable storage. Refer to the IEBCPARM macro's CPARM_PARMLIST DSECT for a map of the specifiable parameter options. These values can be coded:

optionaddr

specifies the address of an option list which contains options usually specified in the PARM parameter of the EXEC statement. This must be present for all utility programs. Refer to the IEBCPARM macro's CPARM_OPTIONS_PREFIX DSECT for a map of the caller's option list.

ddnameaddr

specifies the address of a list of alternate ddnames for the data sets used during utility program processing. If standard ddnames are used and this is not the last parameter in the list, it should point to a halfword of zeros. Refer to the IEBCPARM macro's CPARM_DDNAME_PREFIX DSECT for a map of the ddname array. If you do not wish to specify this parameter or the later parameters, you may omit it.

hdingaddr

specifies the address of a list which contains an EBCDIC beginning page number for the SYSPRINT data set. If *hdingaddr* is omitted, the page number defaults to 1. Refer to the IEBCPARM macro's

CPARM_PAGENUM_PREFIX DSECT for a map of the caller's page number option. If you do not wish to specify this parameter or the later parameter, you may omit it.

uexitparmaddr

specifies the address of the IEBCOPY user-exit parameter list. Refer to the IEBCPARM mapping macro CPARM_EXIT_DSECT for a description of the IEBCOPY user exit parameters. If *uexitparmaddr* is omitted, the IEBCOPY user exits will not be invoked. *Uexitparmaddr* is only supported in IEBCOPY. If you do not wish to specify this parameter, you may omit it.

VL=1

specifies that the sign bit of the last fullword of the address parameter list is to be set to 1.

Related reading:

- For more information about ATTACH, see [z/OS MVS Programming: Assembler Services Reference ABE-HSP](#).
- For more information about LINK parameters, see [z/OS MVS Programming: Assembler Services Reference ABE-HSP](#).
- For more information about program authorization, see [z/OS MVS Programming: Authorized Assembler Services Guide](#).
- For information about TSO, see [z/OS TSO/E Programming Services](#).
- For information about building parameter lists, see [“Building Parameter Lists” on page 325](#).

IEBCOPY User-Exit Parameter

Refer to the IEBCPARM mapping macro CPARM_EXIT_DSECT for a description of the IEBCOPY user exit parameters.

The IEBCPARM mapping macro maps the four word parameter list that is passed by the caller of IEBCOPY. The IEBCPARM mapping macro is created and deleted by the program that invokes IEBCOPY.

Table 62. IEBCPARM mapping macro			
Offset - Decimal (Hex)	Name	Value	Description
00 (00)	CPARM_PARMLIST	DSECT	
00 (00)	CPARM_PARMLIST_OPTIONS	DS A(CPARM_OPTIONS_SIZE)	Options list
04 (04)	CPARM_PARMLIST_DDNames	DS A(CPARM_DDNames_PREFIX)	DDNames list
08 (08)	CPARM_PARMLIST_PAGENUM	DS A(CPARM_PAGENUM_PREFIX)	PAGE NUMBER list
12 (0C)	CPARM_PARMLIST_EXITS	DS A(CPARM_EXIT_DSECT)	User exits parameter list
16 (10)	CPARM_PARMLIST_END	DS 0A	End of supported parameter-list
	CPARM_PARMLIST_LENGTH	EQU *-CPARM_PARMLIST	Parameter-list length
	CPARM_PARMLIST_LASTFLAG	EQU B'10000000'	LAST PARAMETER indicator
00 (00)	CPARM_OPTIONS_PREFIX	DSECT	Full-word boundary
Description: Maps the caller's options list. The CPARM_OPTIONS_SIZE address is contained in the first word in the IEBCOPY invocation parameter list.			

Table 62. IEBCPARM mapping macro (continued)			
Offset - Decimal (Hex)	Name	Value	Description
00 (00)	CPARM_OPTIONS_SIZE	DC Y(L'CPARM_OPTIONS_TEXT)	OS EXEC PARM string length
02 (02)	CPARM_OPTIONS_TEXT	DC CL'100'	OS EXEC PARM character string
00 (00)	CPARM_DDNames_PREFIX	DSECT	Full-word boundary
<i>Description:</i> Maps the caller's DDnames list. The CPARM_DDNames_PREFIX address is contained in the second word in the IEBCOPY invocation parameter list.			
00 (00)	CPARM_DDNames_SIZE	DC Y(L'CPARM_DDNames_LENGTH-2)	DDNames array length
02 (02)	CPARM_DDNames_1	DC XL8'00'	Reserved
10 (0A)	CPARM_DDNames_2	DC XL8'00'	Reserved
18 (12)	CPARM_DDNames_3	DC XL8'00'	Reserved
26 (1A)	CPARM_DDNames_4	DC XL8'00'	Reserved
34 (22)	CPARM_DDNames_IN	DC CL8'SYSIN'	
42 (2A)	CPARM_DDNames_PRINT	DC CL8'SYSPRINT'	
50 (32)	CPARM_DDNames_7	DC XL8'00'	Reserved
58 (3A)	CPARM_DDNames_UT1	DC CL8'SYSUT1'	
66 (42)	CPARM_DDNames_UT2	DC CL8'SYSUT2'	
74 (4A)	CPARM_DDNames_UT3	DC CL8'SYSUT3'	
82 (52)	CPARM_DDNames_UT4	DC CL8'SYSUT4'	
90 (5A)	CPARM_DDNames_12	DC XL8'00'	Reserved
98 (62)	CPARM_DDNames_13	DC XL8'00'	Reserved
106 (6A)	CPARM_DDNames_14	DC XL8'00'	Reserved
114 (72)	CPARM_DDNames_15	DC XL8'00'	Reserved
122 (7A)	CPARM_DDNames_16	DC XL8'00'	Reserved
	CPARM_DDNames_LENGTH	EQU *-CPARM_DDNames_SIZE	
00 (00)	CPARM_PAGENUM_PREFIX	DSECT	Full-word boundary
<i>Description:</i> Maps the caller's page number list. The CPARM_PAGENUM_PREFIX address is contained in the third word in the IEBCOPY invocation parameter list.			
00 (00)	CPARM_PAGENUM_SIZE	DC Y(L'CPARM_PAGENUM_TEXT)	PAGENUM string length

Table 62. IEBCPARM mapping macro (continued)			
Offset - Decimal (Hex)	Name	Value	Description
02 (02)	CPARM_PAGENUM_TEXT	DC CL4'0001'	PAGENUM character string
00 (00)	CPARM_EXIT_DSECT	DSECT	Double-word boundary
<i>Description:</i> Maps the caller's user exit parameter list. The CPARM_EXIT_DSECT address is contained in the fourth word in the IEBCOPY invocation parameter list.			
00 (00)	CPARM_EXIT_LENGTH	DS H	Number of bytes that follow this field.
02 (02)	CPARM_EXIT_PARMS	DS 0H	
02 (02)		DC XL6	Filler for DW alignment, must be zero
08 (08)	CPARM_EXIT_USR	DS CL8	This field allows the invoking program to be re- entrant. On all calls, IEBCOPY will copy this field to CPLST_PARMLST_U SER
16 (10)	CPARM_EXIT_CONTROLSTMT_ADDR	DS AD(0)	Double word that contains a 31-bit address of the invoking program's control statement user exit or zero. The high order 33 bits must be zero. The exit will be called in 31-bit addressing mode.
20 (14)	CPARM_EXIT_CONTROLSTMT_ADD	EQU CPARM_EXIT_MEMBER_ADDR+4,4	31-bit address

Table 62. IEBCPARM mapping macro (continued)			
Offset - Decimal (Hex)	Name	Value	Description
24 (18)	CPARM_EXIT_MEMBER_ADDR	DS AD(0)	Double word that contains a 31-bit address of the invoking program's member-selection user exit or zero. The high order 33 bits must be zero. The exit will be called in 31-bit addressing mode.
28 (1C)	CPARM_EXIT_MEMBER_ADD	EQU CPARM_EXIT_MEMBER_ADDR+4,4	31-bit address
32 (20)	CPARM_EXIT_PARMS_LENGTH	EQU *-CPARM_EXIT_PARMS	Length of PARM LIST

If the address for the control statement exit is zero, IEBCOPY will read control statements from SYSIN or a substituted DD name as documented elsewhere. If the address for the control statement exit is non-zero, then IEBCOPY will not open SYSIN or a DD name substituted for it.

Invocation of IEBCOPY

The invoker of IEBCOPY can specify the exits by using the fourth parameter on the invocation.

The addresses passed to IEBCOPY in R1 and R13 are always interpreted as 24-bit addresses regardless of the AMODE at entry to IEBCOPY. The user exit addresses (passed via the fourth parameter) are interpreted as 31-bit addresses regardless of the AMODE at entry to IEBCOPY. Those exits will always be called in AMODE(31) regardless of the AMODE at entry to IEBCOPY.

The fourth word in the IEBCOPY invocation parameter list will have a 31-bit address of the user exit parameter list. This parameter list will be mapped by the IEBCPARM mapping macro. The fourth word must have the high order bit on indicating the end of the list.

Compatibility Warning: There might be existing invocations of IEBCOPY that pass three parameters but fail to set the high order bit on the last word. The z/OS V2R1 version of IEBCOPY would examine the non-existent fourth pointer and get unpredictable results.

IEBCOPY User Exit Routines

IEBCOPY user exit routines enable a program to supply IEBCOPY control statements instead of IEBCOPY reading control statements from SYSIN. (Note: you cannot specify these user exit routines when you invoke IEBCOPY via JCL.) This makes it possible, for instance, to control the command input stream by providing an exit routine for SYSIN. Also, the user exit can pass information back to IEBCOPY to be added to the SYSPRINT output stream.

When IEBCOPY calls a user exit, standard linkage must be used and standard register convention must be followed; that is, register 1 points to the argument list and it might be above the 16 MB line, register 13 points to a standard 72-byte save area that resides above the line, register 14 contains the return address, and register 15 contains the entry point address. On exit from the user exit routine, CPLST_PARM_LIST_RC contains the exit routine's return code. The list of return codes that are expected from the IEBCOPY user exits is contained in the IEBCPLST mapping macro.

Parameter List for Calls to the Exits

IEBCOPY supplies a parameter list for each call to the invoking program's exits. The parameter list used when calling the exit is mapped by the IEBCPLST mapping macro.

Macro name:

IEBCPLST

Description:

Maps the IEBCOPY parameter list for calls to the user exits

Function:

The IEBCPLST structure is used by IEBCOPY to pass information to the "control statement" exit and the "member-selection" exit.

Created by:

IEBCOPY

Notes:

1. IEBCPLST is created by the IEBCOPY utility when the utility is invoked with a parameter list indicating that user exits are to be employed. You cannot use user exits when you invoke IEBCOPY with JCL or with the TSO CALL command
2. The storage acquired for IEBCPLST will be readable in the user's storage protection key.

Offset - Decimal, Hex	Name	Type	Description
00 00	CPLST_PARMLIST	DSECT	
00 00	CPLST_PARMLIST_HEAD	EQU CPLST_PARMLIST,CPLST_PARMLIST_HEAD_LENGTH	
00 00	CPLST_PARMLIST_HDR	DS 0CL16	Header set by IEBCOPY
00 00	CPLST_PARMLIST_ID	DS CL8	"IEBCPLST" eyecatcher
08 08	CPLST_PARMLIST_LEN	DS F	Length of PARMLIST including the member name array
12 0C	CPLST_PARMLIST_VERSION	DS AL1	Version number = 1
13 0D		DS CL3	Reserved for alignment, must be zero
16 10	CPLST_PARMLIST_DSNUMBER	DS F	Input DS number for current IEBCOPY operation (Origin 1).
20 14	CPLST_PARMLIST_ENTRY_ADD	DS F	A(CPLST_CONTROL_AREA) address of the data record control area or the member entry array.
24 18	CPLST_PARMLIST_COUNT	DS F	Binary 1 - number of member entries, Binary 1 - for data record control set by IEBCOPY
28 1C	CPLST_PARMLIST_RC	DS F	Return code - set by the exit.
32 20		DS XL4	Reserved, must be zero.
36 24	CPLST_PARMLIST_FLAGS	DS 0XL4	Informational flags set by IEBCOPY and passed to the exit. Undefined flags must be zero.
36 24	CPLST_PARMLIST_CONTROL_FLAGS	DS XL1	General exit control flags.

Offset - Decimal, Hex	Name	Type	Description
	CPLST_CONTROL_INIT	EQU B'1000000 0'	Open input statement file or starting a new member list.
	CPLST_CONTROL_DATA	EQU B'0100000 0'	Input statement / print record exit.
	CPLST_CONTROL_MEMBER	EQU B'0010000 0'	Member-Selection exit
	CPLST_CONTROL_MAXRC	EQU B'0001000 0'	IEBCOPY return-code and reason-code have been updated by exit
37 25	CPLST_PARMLIST_INPUTDS_FLAGS	DS XL1	Input data set flags
	CPLST_INPUT_SEQ	EQU B'1000000 0'	Input data set is sequential
	CPLST_INPUT_PDSE	EQU B'0100000 0'	Input data set is a PDSE
	CPLST_INPUT_PDS	EQU B'0010000 0'	Input data set is a PDS
	CPLST_INPUT_SMDE	EQU B'0001000 0'	1 = input directory is an SMDE, 0 = input directory is a PDS
38 26	CPLST_PARMLIST_OUTPUTDS_FLAGS	DS XL1	Output data set flags
	CPLST_OUTPUT_SEQ	EQU B'1000000 0'	Output data set is sequential
	CPLST_OUTPUT_PDSE	EQU B'0100000 0'	Output data set is a PDSE
	CPLST_OUTPUT_PDS	EQU B'0010000 0'	Output data set is a PDS
39 27		DS XL1	Reserved, must be zero
40 28	CPLST_PARMLIST_USR	DS XL8	User data - initially copied from invocation parameter CPARM_EXIT_USR
48 30	CPLST_PARMLIST_MAXRETC	DS F	IEBCOPY return code - set by IEBCOPY and may be updated by the exit. Valid codes are: 0, 4, 8.
52 34	CPLST_PARMLIST_MAXRSNC	DS XL8	Reason code associated with IEBCOPY return code - set by IEBCOPY and may be updated by the exit.

Offset - Decimal, Hex	Name	Type	Description
60 3C		DS XL4	Reserved
	CPLST_PARMLIST_HEAD_LENGTH	EQU	*-CPLST_PARMLIST
64 40	CPLST_CONTROL_AREA	DSECT	Exit control area
The Control Statement exit control area contains the parameters passed to the "control statement" user exit for either a SYSIN control statement or a SYSPRINT report record.			
ORG CPLST_CONTROL_AREA Start of exit control area			
00 00	CPLST_RECORD_CONTROL	DS 0D	record data control parameters
	CPLST_CONTROL_OFLAGS	DS 0XL4	Flags set by input control exit. Undefined flags must be zero
00 00	CPLST_CONTROL_OFLAG1	DC XL1'00'	Exit information flag One
	CPLST_CONTROL_IN	EQU B'1000000 0'	Input control statement record
	CPLST_CONTROL_PRINT	EQU B'0100000 0'	Output print record.
	CPLST_CONTROL_SELECT	EQU B'0010000 0'	Member selection in effect.
		DC XL3	Reserved. Must be zero.
04 04		DS F	Reserved.
The following equates define the range of valid lengths that can be specified in the CPPLST_CONTROL_DATA_LEN field for either an input statement or print record. These values will be used when the length field is validity checked by IEBCOPY.			
	CPLST_INPUT_STATEMENT_MINLEN	EQU 72	Statement minimum length including continuation character in column 72.
	CPLST_INPUT_STATEMENT_MAXLEN	EQU 80	Statement maximum length
	CPLST_PRINT_RECORD_MINLEN	EQU 60	Print record minimum length not including carriage control character.
	CPLST_PRINT_RECORD_MAXLEN	EQU 120	Print record maximum length not including carriage control character.
	CPLST_CONTROL_DATA_NONE	EQU 00	Omitted data record length
08 08	CPLST_CONTROL_DATA_ADDR	DS AD(0)	Double word that contains the 31-bit address of data record. The high order 33 bits must be zero.
12 0C	CPLST_CONTROL_DATA_ADD	EQU CPLST_CONTROL_DATA_ADDR+4,4 (31-bit address of data record)	
16 10	CPLST_CONTROL_DATA_LEN	DS F	Length of data record.
20 14		DS 0D	End of CPLST_RECORD_CONTROL

Offset - Decimal, Hex	Name	Type	Description
The member entry, created by IEBCOPY, contains a member name that is passed to the "member-selection" user exit. In the current release the count of one member entry is set in the CPLST_PARMLIST_DSNUMBER field.			
	ORG CPLST_CONTROL_AREA		Start of exit control area.
00 00	CPLST_MEMBER_ENTRY	DS 0D	One member entry
00 00	CPLST_MEMBER_OFLAGS	DS 0XL4	Flags set by Member-Selection exit. Undefined flags must be zero
00 00	CPLST_MEMBER_OFLAG1	DS XL1	Exit information flag One
	CPLST_MEMBER_REPLACE_ANY	EQU B'1000000 0'	Replace primary or alias name
	CPLST_MEMBER_RENAME	EQU B'0100000 0'	Use the NEWNAME for output name
01 01		DS XL3	Reserved, must be zero
04 04	CPLST_MEMBER_RC	DS F	Return code - set by exit
08 08		DS XL8	Reserved
16 10	CPLST_MEMBER_NAME_ADDR	DC AD(0)	Set by IEBCOPY, double word that contains the 31-bit address of the member name area. The member name area begins with a two-byte name length field followed by the member name. Currently, the length field value is set to eight bytes.
24 18	CPLST_MEMBER_NEWNAME_ADDR	DC AD(0)	Set by the exit, double word that contains the 31-bit address of the member new name area. The member new name area begins with a two-byte name length field followed by the new member name. Currently, the length field value is set to eight bytes.
32 20		DS XL8	Reserved
40 28	CPLST_MEMBER_DIR_ADDR	DS D	Double word that contains the 31-bit address of the directory entry. The high order 33 bits must be zero.
44 2C	CPLST_MEMBER_DIR_ADD	EQU CPLST_MEMBER_DIR_ADDR+4,4 (31-bit address)	
48 30	CPLST_MEMBER_DIR_LEN	DS F	Length of directory entry
		DS XL4	Filler for DW alignment, must be zero.
56 38	CPLST_MEMBER_NAME_AREA	DS 0H	Minimum length member name area
56 38	CPLST_MEMBER_NAME LENG	DC Y(L'CPLST_MEMBER_NAME) – member name length	

Offset - Decimal, Hex	Name	Type	Description
58 3A	CPLST_MEMBER_NAME	DC CL8' '	Member name
66 42	CPLST_MEMBER_NEWNAME_AREA	DS 0H	Minimum length new name area
66 42	CPLST_MEMBER_NEWNAME LENG	DC Y(L'CPLST_MEMBER_NEWNAME) – new name length	
68 44	CPLST_MEMBER_NEWNAME	DC CL8' '	New member name - set by exit.
80 50		DS 0D	End of CPLST_MEMBER_ENTRY

The user exit should check the version field and should give a return code if it exceeds the version number that the exit was designed to handle. Normally IBM expects changes to this parameter list will be upward compatible and the version field will be unchanged in future releases.

The Control Statement Exit

IEBCOPY will call this exit each time that it needs to read a control statement. If this exit is supplied, IEBCOPY will not use SYSIN or a DD substituted for it. On each call to this exit, the exit can return one control statements or zero or more lines to be printed to SYSPRINT. Note: control statement record returned to IEBCOPY may be a SYSIN statement record or a SYSPRINT report record.

Restriction: The storage containing control statements or messages must not be deleted until next control statement exit call or IEBCOPY termination

EBCPLST parameter list fields that are filled in by IEBCOPY:

- CPLST_PARMLIST_HDR
- CPLST_PARMLIST_USR
- CPLST_PARMLIST_ENTRY_ADD
- CPLST_PARMLIST_COUNT (value is always one)
- CPLST_PARMLIST_FLAGS

IEBCPLST parameter list fields that are filled in by the **control statement** exit:

CPLST_PARMLIST_DATA

31-bit address of control statement which must be in the format of an IEBCOPY control statement (including continuation characters if necessary) and will be logically padded on the right with blanks to 80 characters or address of print line which must not include a control character.

CPLST_PARMLIST_DATA_LEN

Length of control statement must be 72-80 or 0 when omitted. length of print line must be 60-120 or 0 when omitted.

CPLST_PARMLIST_RC

Return code in the parameter list, not in register 15:

- 00 = Control Statement exit: One control statement returned. Count is number of input records to process (must be 1)
- 00 = Member-Selection exit: Successful processing.
- 04 = Zero or more control statement returned. End of group - Process previous and new records and execute the group returned.
- 08 = A user exit successfully processed an initialization call and should be called again for the next control statement or member array.
- 16 = Print zero or more lines (length may be zero). Count is number of print lines to process (must be 1).

- 20 = Start a new page and print zero or more lines. Count is number of print lines to process (must be 1).
- 32 = Exit IEBCOPY. Data record is ignored (IEBCOPY only prints final message(s) and exits.)
- 36 = Severe error - Do not call again - IEBCOPY exits with return code 16.

The Member-Selection Exit

IEBCOPY passes a list of the member names in an array to the member-selection exit. The exit can decide and choose which member names to include or exclude from the current copy operation.

Restrictions:

- IEBCOPY will put out a new message and fail if a SELECT or EXCLUDE statement is processed and a member exit exists.
- Members will be presented in alphameric order within each input data set in the "concatenation."
- Only names eight characters or less will be passed.
- New name will be limited to eight characters.

EBGPLST parameter list fields that are filled in by IEBCOPY:

- PLST_PARMLIST_HDR
- CPLST_PARMLIST_USR
- CPLST_PARMLIST_ENTRY_ADD CPLST_PARMLIST_COUNT (value will be the count of member entries in the member array).
- CPLST_PARMLIST_FLAGS
- CPLST_MEMBER_NAME
- CPLST_MEMBER_DIR_ADDR
- CPLST_MEMBER_DIR_LEN

IEBCPLST parameter list fields that are filled in by the **member-selection** exit:

CPLST_PARMLIST_DSNUMBER

Input DS number for current copy operation (origin 1).

CPLST_PARMLIST_RC

Return code in the parameter list, not in register 15:

- 00 = OK
- 36 = Severe error - Do not call again.

CPLST_MEMBER_RC

- 00 = member selected. Must call exit for next member.
- 04 = member excluded. Must call exit for next member.
- 08 = member selected including all remaining members in this input data set. No need to call exit again for this input data set.
- 12 = member excluded including all remaining members in this input data set. No need to call exit again for this input data set.

CPLST_MEMBER_OFLAGS

Flags set by the member-selection exit.

CPLST_MEMBER_NEWNAME

Filled in if CPLST_MEMBER_RENAME flag bit is set on.

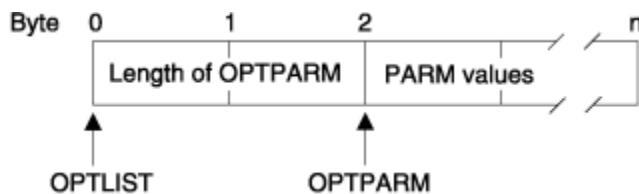
Building Parameter Lists

This section contains information about option lists, ddname lists, and page number parameters.

Options List

The options list is the parameter list that contains the options that are usually specified in the PARM parameter of the EXEC statement. This list is always required, even if you are not passing any PARM options to a utility program.

The general syntax of the PARM options parameter list (OPTLIST) is:



The options list should begin on a halfword boundary. The two high-order bytes of this list must contain a binary count of the number of bytes in the remainder of the options list. The options list is free-form, with fields separated by commas. No blanks or binary zeros should appear in the list outside of the first two bytes (the length indicator).

For example, you can start IEBCOPY and pass it some PARM parameters by coding the following (in assembler):

```
LINK EP=IEBCOPY,PARAM=(OPTLIST),VL=1
.
.
.
OPTLIST DC AL2(L'OPTPARM)
OPTPARM DC C'SIZE=1000K,WORK=1M'
```

Note that in the preceding example, you do not code the parentheses or single quotation marks that you would normally code in the PARM parameter. The PARM parameters for the IEBCOPY example above would normally be coded PARM=(SIZE=1000K,WORK=1M) or PARM='SIZE=1000K,WORK=1M', but you should not pass the enclosing parentheses or single quotations to IEBCOPY when you start the program from an application program.

When you are not passing any PARM parameter values to a utility program, code a halfword of binary zeros for the options list. For instance, to start the IEBGENER program from an application program using no PARM values and the default ddnames, code (in assembler):

```
LINK EP=IEBGENER,PARAM=(OPTLIST),VL=1
.
.
.
OPTLIST DC H'0'
```

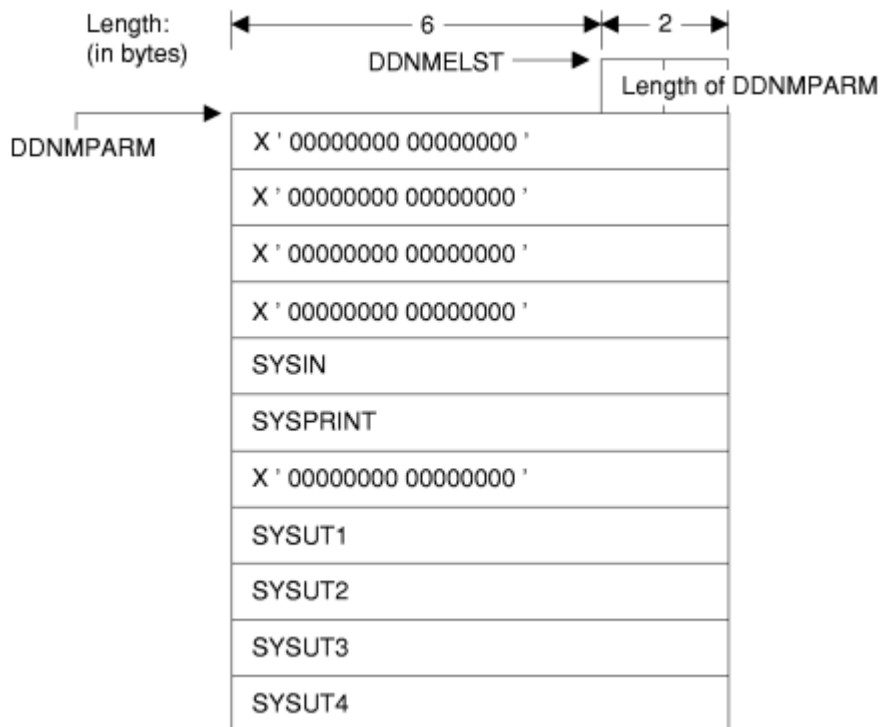
ddname List

The ddname list is a parameter list containing alternate ddnames for the data sets or volumes that you want the utility program to use. If you are using the standard ddnames for your data sets, you do not need to code a ddname list, unless you code a page header parameter.

An example of a case where you might want to use alternate DD names is when your program is calling various programs that use the same DD name for different data sets. Another example is when a program dynamically allocates a data set and writes utility control statements to it before calling a utility.

The ddname list should begin on a halfword boundary. The two high-order bytes must contain a count of the number of bytes in the remainder of the list. Each ddname must take up 8 bytes. If a ddname is shorter than 8 bytes, it must be left aligned and padded with blanks. If you code binary zeros for a ddname, or if you omit a ddname by shortening the ddname list, the standard ddname is assumed. You cannot omit a ddname from the middle of the ddname list without replacing it with binary zeros.

The general structure of the ddname parameter list (DDNMELST) is:



For example, to start IEBCOPY using nonstandard ddnames, you could code:

```

LINK EP=IEBCOPY,PARAM=(OPTLIST,DDNMELST),VL=1
.
.
.
OPTLIST DC H'0'
DDNMELST DC AL2(L'DDNMEND)
DDNMPARM DC 7XL8'0'
DC CL8'INPDS '
DC CL8'OUTPDS '
DDNMEND EQU DDNMPARM,*-DDNMPARM

```

In this example, IEBCOPY is told to use INPDS as the input data set and OUTPDS as the output data set.

To start utilities such as IEBCOPY with multiple input or output data sets, it is necessary to pass a ddname list to the utility with alternative ddnames for SYSUT1 and SYSUT2. The utility control statements will be sufficient to identify the other ddnames that you require.

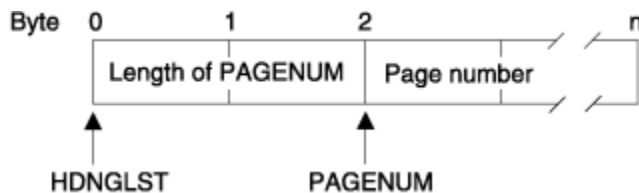
You do not need to code a ddname parameter list when you are invoking the IEH system utilities. The ddnames for these utilities define devices rather than data sets, and the utility control statements used by these utilities are sufficient for identifying the appropriate devices. The IEH utilities only use the entries for SYSIN and SYSPRINT from the ddname list.

Page Header Parameter

You can specify the beginning page number of your printed output by passing to a utility a page header parameter. The first two bytes of this parameter must contain the length of the remainder of the parameter. The page number cannot be longer than 4 bytes and must be in EBCDIC format.

Some utilities update the page number that are passed to them. They replace it with a value that is one greater than the last page number used. This allows for consecutive invocations.

The general syntax of the page header parameter (HDNGLST) is:



For example, to load IEHLIST and get a printout whose first page begins with a page number of 10, you could code:

```

                LINK  EP=IEHLIST,PARAM=(OPTLIST,DDNMELST,HDNGLST),VL=1
                .
                .
OPTLIST        DC    H'0'
DDNMELST       DC    H'0'
HDNGLST        DC    AL2(L'PAGENUM)
PAGENUM        DC    C'10'

```

Some utilities use fewer than 4 bytes per page number. Storing a page number that is too large in the page header parameter could cause unpredictable results. For example, if you link to IEBIMAGE with a page number of 998, the following page numbers result:

```

998
999
(blank)
1
2
(and so on)

```

In this case, you cannot specify a page number larger than 999.

Return Codes

The following sections define the return codes for the utility programs.

IEBCOMPR Return Codes

IEBCOMPR returns a code in register 15 to indicate the results of program execution. The return codes and their meanings are:

Codes

Meaning

00 (X'00')

Successful completion.

08 (X'08')

An unequal comparison. Processing continues.

12 (X'0C')

An unrecoverable error exists. The utility ends.

16 (X'10')

A user routine passed a return code of 16 to IEBCOMPR. The utility ends.

IEBCOPY Return Codes

IEBCOPY returns a code in register 15 to indicate the results of program execution. The return codes and their meanings are:

Codes

Meaning

00 (X'00')

Successful completion.

04 (X'04')

One or more copy group operations ended unsuccessfully or were incompletely performed. Recovery may be possible.

08 (X'08')

An unrecoverable error exists. The utility ends.

IEBCOPY returns an eight-byte structure in register 0, containing an ABEND code and the associated reason code if an ABEND occurred in the FAMS (file access management services) component. Register 0 contains zeros if no ABEND occurred in that component. Note: ABENDs that occur in other tools or macros which IEBCOPY or FAMS reference (for example Binder or DESERV) are not reported in register 0.

If your program stores the eight-byte reason code from register 0 into storage, you can map it with the IEBCREAS macro. It contains these fields:

Table 63. IEBCREAS mapping macro				
Offset	Name	Type	Length	Description
0	IEBRSN	Character	8	
0	IEBRSN_AbndRsn	Fixed	4	If IEBRSN_Abended is on, then ABEND reason code or reg 0 at time of ABEND.
4	IEBRSN_Flags	Bitstring	1	Flags.
	IEBRSN_Abended	1... ..		IEBCOPY recovered from an ABEND. IEBRSN_AbndRsn is valid and IEBRSN_AbndCode contains the system or user ABEND code.
		.xxx xxxx		Reserved
5	IEBRSN_AbndCode	Fixed	3	System completion code (first 12 bits) and user completion code (second 12 bits) in hexadecimal if IEBRSN_Abended is on.

Programs running in any addressing mode (AMODE) can see all eight bytes of the IEBCOPY reason code by using the STG instruction.

IEBCOPY User ABEND Codes

In a diagnostic situation, IEBCOPY may issue a user ABEND. In most cases, an IEBCOPY message precedes the ABEND. The ABEND code is the same as the message number. For example, message IEB1021E will precede user ABEND U1021.

If user ABEND U0001 is encountered, it may indicate that the file mark at the end of a PDS member is missing. In some cases, the end-of-file mark can be restored with a DFSMSdss DUMP/RESTORE (or equivalent) operation on the data set. The reason code associated with ABEND U0001 is for use by IBM support in determining the exact nature of the error.

IEBDG Return Codes

IEBDG returns a code in register 15 to indicate the results of program execution. The return codes and their meanings are:

Codes**Meaning****00 (X'00')**

Successful completion.

04 (X'04')

A user routine returned a code of 16 to IEBDGL. The utility ends at the user's request.

08 (X'08')

An error occurred while processing a set of utility control statements. No data is generated following the error. Processing continues normally with the next set of utility control statements, if any.

12 (X'0C')

An error occurred while processing an input or output data set. The utility ends.

16 (X'10')

An error occurred from which recovery is not possible. The utility ends.

IEBEDIT Return Codes

IEBEDIT returns a code in register 15 to indicate the results of program execution. The return codes and their meanings are:

Codes**Meaning****00 (X'00')**

Successful completion.

04 (X'04')

An error occurred. The output data set may not be usable as a job stream. Processing continues.

08 (X'08')

An unrecoverable error occurred while attempting to process the input, output, or control data set. The utility ends.

IEBGENER Return Codes

IEBGENER returns a code in register 15 to indicate the results of program execution. The return codes and their meanings are:

Codes**Meaning****00 (X'00')**

Successful completion.

04 (X'04')

Probable successful completion. A warning message is written.

08 (X'08')

Either processing was ended after you requested processing of user header labels only, or a DBCS error was encountered.

12 (X'0C')

Either an unrecoverable error exists and the job step is stopped, or a DBCS error was encountered.

16 (X'10')

A user routine passed a return code of 16 to IEBGENER. The utility ends.

IEBIMAGE Return Codes

IEBIMAGE returns a code in register 15 that represents the most severe error condition encountered during the program execution. This return code is also printed in the output listing. The return codes and their meanings are:

Codes**Meaning****00 (X'00')**

Successful completion; operations performed as requested.

04 (X'04')

Operations performed; investigate messages for exceptional circumstances.

08 (X'08')

Operations not performed; investigate messages.

12 (X'0C')

Severe exception; processing may end.

16 (X'10')

Unrecoverable exception; the utility ends.

20 (X'14')

SYSPRINT data set could not be opened; the utility is ended.

24 (X'18')

User parameter list incorrect; the utility is ended.

IEBPDSE Return Codes

IEBPDSE returns a code in register 15 to indicate the results of program execution. The return codes and their meanings are:

Codes

Meaning

00 (X'00')

Successful completion.

04 (X'04')

The input PDSE is slightly damaged. Processing continues.

08 (X'08')

The input PDSE is corrupted. The utility ends.

12 (X'0C')

The input PDSE could not be opened. The utility ends.

16 (X'10')

The input data set is not a PDSE. The utility ends.

IEBTPCH Return Codes

IEBTPCH returns a code in register 15 to indicate the results of program execution. The return codes and their meanings are:

Codes

Meaning

00 (X'00')

Successful completion.

04 (X'04')

Either a physical sequential data set is empty or a partitioned data set has no members.

08 (X'08')

A member specified for printing or punching does not exist in the input data set and processing will continue with the next member, or a DBCS error was encountered.

12 (X'0C')

An unrecoverable error occurred, a user routine passed a return code of 12 to IEBTPCH and the utility is ended, or a DBCS error was encountered.

16 (X'10')

A user routine passed a return code of 16 to IEBTPCH. The utility is ended.

IEBUPDTE Return Codes

IEBUPDTE returns a code in register 15 to indicate the results of program execution. The return codes and their meanings are:

Codes

Meaning

00 (X'00')

Successful completion.

04 (X'04')

A control statement is coded incorrectly or used erroneously. If either the input or output is sequential, the utility is ended. If both are partitioned, the program continues processing with the next function to be performed.

12 (X'0C')

An unrecoverable error exists. The utility is ended.

16 (X'10')

A label processing code of 16 was received from a user's label processing routine. The utility is ended.

IEHINITT Return Codes

IEHINITT returns a code in register 15 to indicate the results of program execution. The return codes and their meanings are:

Codes

Meaning

00 (X'00')

Successful completion. A message data set was created.

04 (X'04')

Successful completion. No message data set was defined by the user.

08 (X'08')

IEHINITT completed its operation, but error conditions were encountered during processing. A message data set was created.

12 (X'0C')

IEHINITT completed its operation, but error conditions were encountered during processing. No message data set was defined by the user.

16 (X'10')

IEHINITT ended operation because of error conditions encountered while attempting to read the control data set. A message data set was created if defined by the user.

IEHLIST Return Codes

IEHLIST returns a code in register 15 to indicate the results of program execution. The return codes and their meanings are:

Codes

Meaning

00 (X'00')

Successful completion.

08 (X'08')

An error condition caused a specified request to be ignored. Processing continues.

12 (X'0C')

A permanent input/output error occurred. The job is ended.

16 (X'10')

An unrecoverable error occurred while reading the data set. The job is ended.

IEHMOVE Return Codes

IEHMOVE returns a code in register 15 to indicate the results of program execution. The return codes and their meanings are:

Code	Meaning
00 (X'00')	Successful completion.
04 (X'04')	A specified function was not completely successful. Processing continues.
08 (X'08')	A condition exists from which recovery is possible. Processing continues.
12 (X'0C')	An unrecoverable error exists. The utility is ended.
16 (X'10')	It is impossible to OPEN the SYSIN or SYSPRINT data set. The utility is ended.

IEHPROGM Return Codes

IEHPROGM returns a code in register 15 to indicate the results of program execution. The return codes and their meanings are:

Codes	Meaning
00 (X'00')	Successful completion.
04 (X'04')	A syntax error was found in the name field of the control statement or in the PARM field in the EXEC statement. Processing continues.
08 (X'08')	A request for a specific operation was ignored because of an incorrect control statement or an otherwise invalid request. The operation is not performed.
12 (X'0C')	An input/output error was detected when trying to read from or write to SYSPRINT, SYSIN or the VTOC. The utility is ended.
16 (X'10')	An unrecoverable error exists. The utility is ended.

Appendix B. Unload partitioned data set format

This appendix contains Programming Interface and Associated Guidance Information.

This appendix is intended to help you use the unload data set that is created by an IEBCOPY unload operation.

Introduction

An unload data set will have a format different from a partitioned data set or a PDSE, no matter if the data set is being stored on a DASD device or on tape. Its records will be longer and may require more space. It will have sequential organization. You cannot always treat an unload data set the same as you would the original partitioned data set or PDSE.

It may be easy to confuse the **unload** data set, which is sequential, from the partitioned data set or PDSE. The *unload* data set or *container* data set is the sequential data set created by the unload operation. The *unloaded* data set is the original partitioned data set or PDSE that was *unloaded* into the *unload* data set.

Records present in an unload data set

This is a list of all of the types of records that may appear in an unload data set in the groupings and order in which they must appear:

1. Unload File Header

- COPYR1, the first header record is always present.
- COPYR2, the second header record is always present.

2. Directory Information

- Directory Block Records

One or more are always present.

Each record contains 1 or more partitioned data set directory blocks.

The last record also ends with an EOF block of 12 bytes of zeros.

For PDSE, a long name directory containing names greater than 8 characters will be included.

- Data Set Attribute Records

These are optional, and can only appear in a PDSE format unload data set.

3. Individual Member Data (This group repeats for each member.)

- Note List Record

Used by the linkage editor and other applications to record relocatable addresses of records inside the member.

- Member Data Records

One or more present.

Each record contains 1 or more physical blocks of the original data set.

The last record also ends with an EOF block (12 bytes of zeros).

- Member Attribute Records

These are optional, and can only appear in a PDSE format unload data set.

A detailed description of the different records is given in later paragraphs.

Different unload data set formats

There are three formats that the unload data set can take. The primary difference between them is which records can appear in the data set. These formats are:

1. Invalid Format

All records after the COPYR1 (if any) are undefined. The condition occurs when an unload operation is ended because of an error. The COPYR1 is re-written as the first record in the container data set with "Invalid Format" as part of error clean-up.

2. Old Format (Pre-PDSE)

- There may be a note list record for each member.
- There are no attribute records.
- The original data set was a partitioned data set, not a PDSE.
- All DASD addresses are valid for a real device, and the DEB and DEVTAB information comes from the DASD device which held the original data set.
- The second batch of data set label information (starting 46 bytes into COPYR1 record) is not present.

3. New Format

- Note list records are now used by the linkage editor and other applications to record relocatable addresses of records inside the member.
- Attribute records may be present if the original data set is a PDSE.
- Records from a PDSE contain DASD addresses from an artificial device that has 256 tracks and 65536 cylinders and tracks of 16M bytes. This convenience maps the maximum number of possible PDSE RLTs and MLTs according to the restrictions for accessing PDSEs with BPAM documented in the [*z/OS DFSMS Using Data Sets*](#).

While these addresses of consecutive records are strictly ascending, some addresses are not to be used. Record numbers for PDSE members are always odd starting with 1 and continuing 3, 5, 7, 9... Even record numbers are reserved.

- The second batch of data set label information (starting 46 bytes into the COPYR1 record) is valid.

Detailed record descriptions

General rules and restrictions

1. The maximum unload record length is 32780, which occurs when the data set being unloaded has a block size of 32760.

Note that this is longer than the maximum permitted physical block length of 32760 bytes, so sometimes records must be spanned across physical blocks.

This number also exceeds the maximum LRECL allowed in a data set label. When the data set label LRECL is 32760, the assembled logical record may actually be longer.

2. When a record must be spanned across physical blocks, each block except the last must be completely filled.
3. The length of any unload data set physical record can never exceed the unload data set logical record length plus 4, even when the data set block size would allow a longer physical record.

Header records

The first two records of an IEBCOPY unload data set contain information that is required to load the data set. The first record (COPYR1) contains a status field, an ID field, and a DCB information field. The status and ID fields are used for validity checking procedures. The DCB field is used to initialize the output data set during a load function. The second record (COPYR2) contains parts of the original data extent block (DEB) of the unloaded partitioned data set. When the data set is to be loaded, this information is used to

update all the user and note list TTRs. [Table 64 on page 337](#) and [Table 65 on page 338](#) show the different fields in the COPYR1 and COPYR2 records.

Table 64. Contents of the COPYR1 descriptor record

Offset Into Record	Field size (Bytes)	Type Of Data	Field Contents
0	64	Structure	COPYR1 - first header record
0	4	Structure	Block Descriptor Word (BDW) for RECFM=VS data sets
0	2	Unsigned Binary	Length of block, including BDW
2	2	reserved	Must be zero
4	4	Structure	Segment Descriptor Word (SDW) for RECFM=VS data sets
4	2	Unsigned Binary	Length of segment, including SDW
6	2	Bit Flags	Must be zero (COPYR1 record is never segmented)
8	1	Bit Flags	Unload Data set Information. Numbering the MSB as "0", 0 & 1 - B'00' = valid unload data set in old format. B'01' = valid unload data set in PDSE format. B'10' = the original data set cannot be reloaded because this unload data set is known to be incomplete or in error. B'11' = Reserved format. 2 - Reserved, and must be zero. 4 - When set, the original data set was known to contain program objects. When not set, it is not known if the contents are or are not programs. 4 - Reserved, and must be zero. 5 - Reserved, and must be zero. 6 - Reserved, and must be zero. 7 - When set, the original data set was a PDSE.
9	3	Binary	The constant value X'CA6D0F'. (The following fields are from the original data set label (Format 1 DSCB).)
12	2	Bitstring	Data set organization (DS1DSORG). X'0200' is PDS.
14	2	Unsigned Binary	Block size (DS1BLKL)
16	2	Unsigned Binary	Logical Record Length (DS1LRECL)
18	1	Bit Flags	Record Format (DS1RECFM) Numbering the MSB as "0", 0 & 1 - B'00' is unknown format B'01' is Variable format (RECFM=V) B'10' is Fixed format (RECFM=F) B'11' is Undefined (RECFM=U) 2 - When set, DASD track overflow may be used. 3 - When set, records may be blocked. 4 - When set, variable format records may be spanned; for fixed format only the last block may be short. 5 & 6 - B'00' is first record byte is a data byte. B'01' first byte is IBM machine carriage control. B'10' first byte is ANSI/ISO carriage control. B'11' is an invalid combination. 7 - Reserved and may be either zero or one.
19	1	Unsigned Binary	Length of record key field (DS1KEYL)
20	1	Bit Flags	Option codes associated with the data set (DS1OPTCD)

Table 64. Contents of the COPYR1 descriptor record (continued)

Offset Into Record	Field size (Bytes)	Type Of Data	Field Contents
21	1	Bit Flags	SMS Indicators (DS1SMSFG). Numbering the MSB as "0", 0 - Managed data set 1 - unpredictable 2 - Data set is reblockable 3 - unpredictable 4 - Data set is a PDSE 5 - unpredictable 6 - Reserved 7 - Reserved, and must be zero. (End of fields from the original data set label).
22	2	Unsigned Binary	The block size of this container data set, which contains the unload data set.
24	20	Structure	Information about the device from which the data set was unloaded. Obtained by a DEVTYPE macro with the DEVTAB parameter. "1" on page 338
44	2	Unsigned Binary	Number of header records. Zero implies 2.
46	18	reserved	Zeros in "old" format unload data set (The following fields are from the original data set label (Format 1 DSCB).)
46	1	reserved	Must be zero
47	3	Structure	Date last referenced yydddd (DS1REFD)
50	3	Structure	Secondary Space Extension (DS1SCEXT)
53	4	Structure	Secondary Allocation (DS1SCALO)
57	3	Structure	Last Track Used TTR (DS1LSTAR)
60	2	Unsigned Binary	Last Track Balance (DS1TRBAL)
62	2	reserved	Must be zero (End of fields from the original data set label).

Notes:

1. These fields are highly device dependent and are required to translate absolute DASD addresses (MBBCHHR) in the member data records to relative addresses (TTR). The DEB control block and DEVTYPE macro are documented in [z/OS DFSMSdfp Advanced Services](#)

Table 65. Contents of the COPYR2 descriptor record

Offset Into Record	Field size (Bytes)	Type Of Data	Field Contents
0	284	Structure	COPYR2 - first header record
0	4	Structure	Block Descriptor Word (BDW) for RECFM=VS data sets
0	2	Unsigned Binary	Length of block, including BDW
2	2	reserved	Must be zero
4	4	Structure	Segment Descriptor Word (SDW) for RECFM=VS data sets
4	2	Unsigned Binary	Length of segment, including SDW
6	2	Bit Flags	Must be zero (COPYR2 record is never segmented.)
8	16	Structure	Last 16 bytes of basic section of the Data Extent Block (DEB) for the original data set. "1" on page 338
24	256	Structure	First 16 extent descriptions from the original DEB. "1" on page 338
280	4	reserved	Must be zero

Notes:

1. These fields are highly device dependent and are required to translate absolute DASD addresses (MBBCHHR) in the member data records to relative addresses (TTR). The DEB control block is documented in [z/OS DFSMSdfp Advanced Services](#). DEVTAB information is documented in DFP System Data Administration.

Directory block records

The directory records are written immediately after the header records. They consist of directory blocks containing the original directory entries for all members to be unloaded. In addition, the last directory record contains an end-of-file block.

The length of each directory record, except the last one, is $8 + n(276)$, where n represents the blocking factor (n is an integer greater than zero). The length of the last directory record is $8 + n(276) + 12$, where n represents the blocking factor (which may be zero).

The directory blocks in the figure contain a count, key, and data field. The count field is set to zero, except for the key length (X'08') and the data length (X'0100').

Figure 46 on page 339 gives the directory record format for the unloaded partitioned data set. The following items have been assumed:

- The block size of the data set to contain the unloaded data set is 900 bytes.
- Seven pseudo directory blocks are required to contain the original directory entries for all of the unloaded members.

Related reading: For more information about directory blocks, see [z/OS DFSMS Using Data Sets](#).

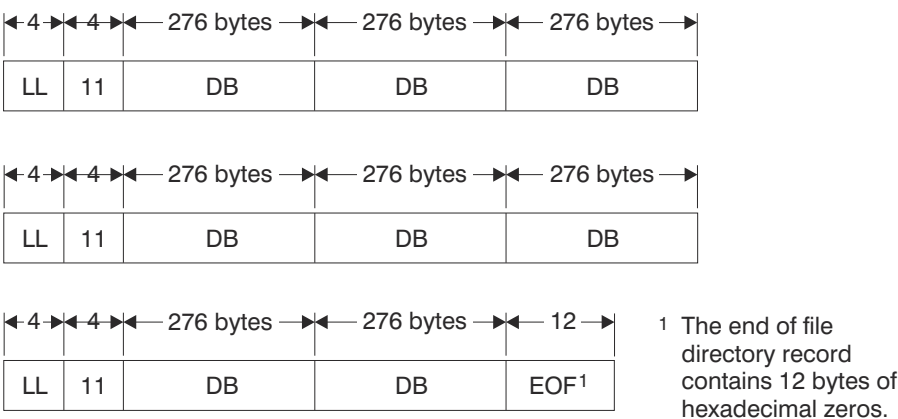


Figure 46. Directory record layout

Attribute records

A PDSE contains attributes which are recorded in attribute records. Attributes that pertain to the whole data set follow the directory records. Those which pertain to a member follow the member data records. The format for an unloaded attribute record is shown in [Figure 47 on page 340](#).

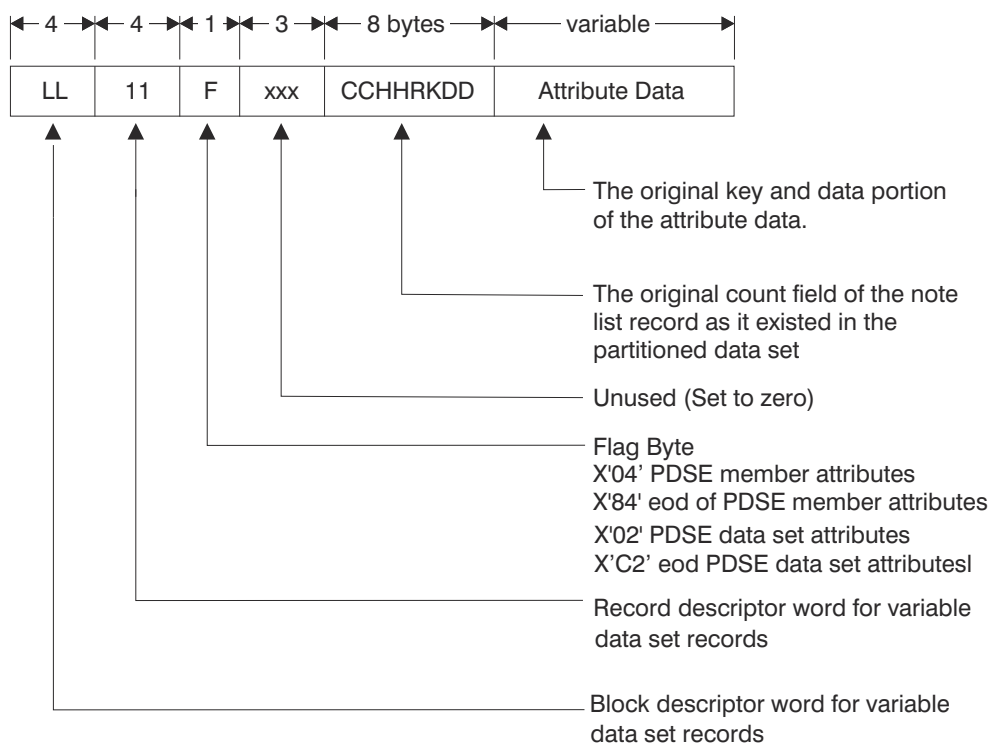


Figure 47. Attribute record layout

Note list records

Note list records as described in the following list are obsolete and do not generally appear even in old format unload data sets. Note lists are treated as member data in new format unload data sets.

If a member to be unloaded contains a note list, the note list is unloaded preceding the member data. The format for an unloaded note list record is shown in [Figure 48 on page 341](#).

IEBCOPY user TTRN limits

- Three user TTRN fields in the directory.
- Only one of these fields may have $n > 0$.
- The maximum length of the note list record identified by the user TTRN with $n > 0$ is 1291 bytes, including any block and record descriptor word.
- No TTRN fields in a note list record may have $n > 0$.
- No user TTRN field in a note list record or in the partitioned data set directory may have the leftmost bit on (that is, the most significant bit of the first "T" in TTRN).

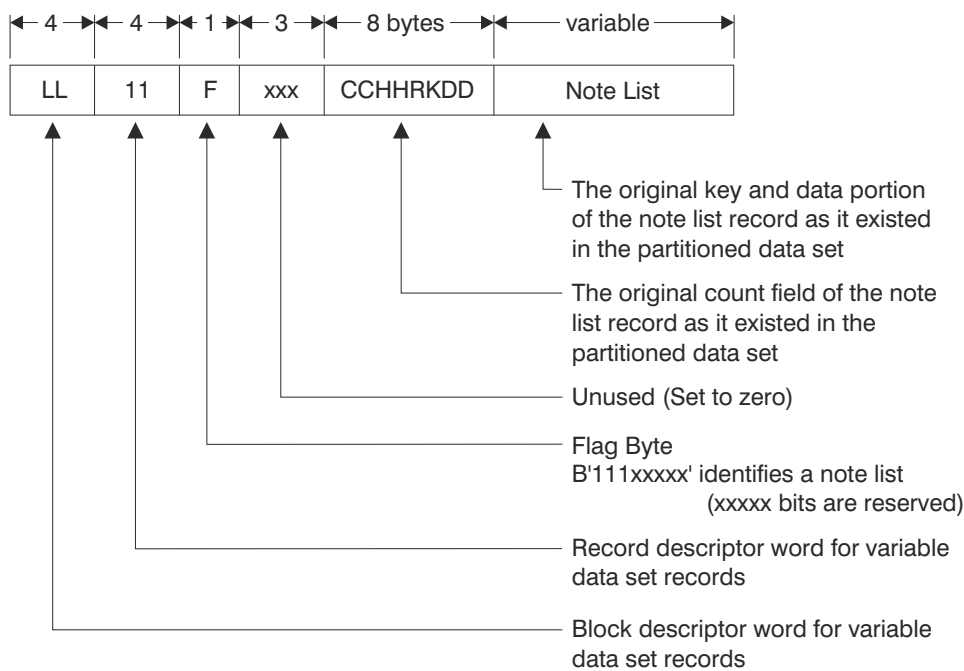


Figure 48. Note list record layout

Member data records

An unloaded member data record consists of the maximum number of set member data blocks that will fit into the unload data set record. The number of blocks in each member data record varies when the partitioned data set or PDSE has undefined or variable length blocks. A member data record contains member data blocks from only one member and has an end of file block after the last member data block.

Figure 49 on page 341 is an example of the format of unloaded member data records. The following items have been assumed:

- The block size of the partitioned data set is 350 bytes.
- There are six member data blocks per member (including the direct access end-of-file block).
- The record syntax of the partitioned data set is fixed.
- The block size of the data set to contain the unloaded data set is 900 bytes.

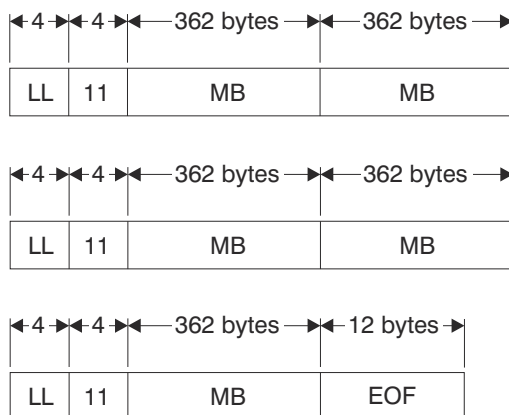


Figure 49. Member data record layout

Figure 50 on page 342 shows the make-up of each member data block.

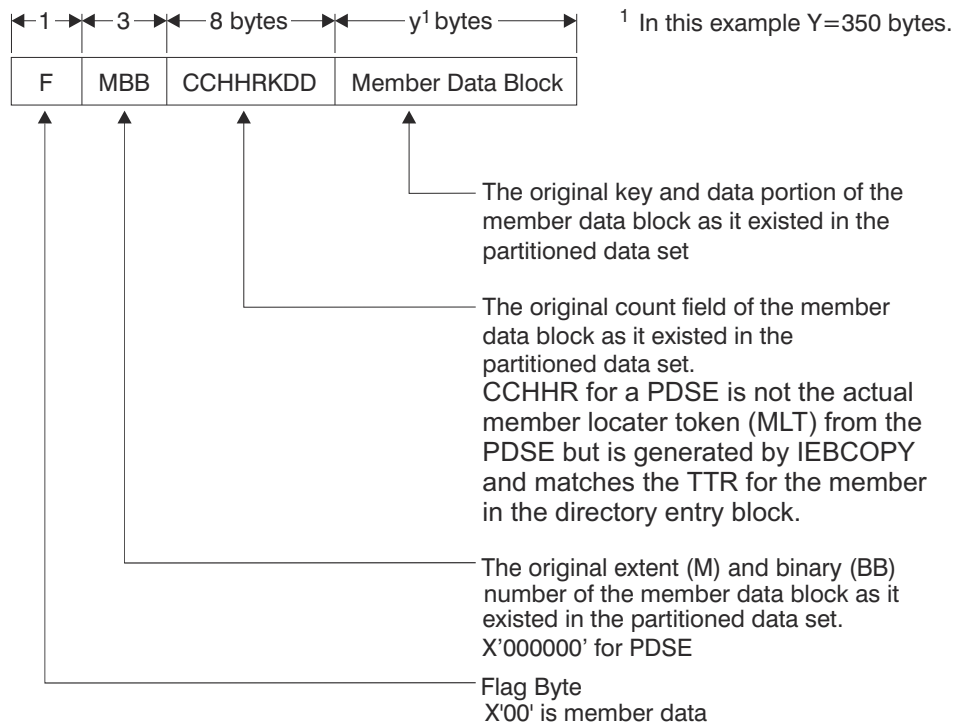


Figure 50. Member data block layout

Figure 51 on page 342 shows the make up of the end-of-file block that follows the last data block of a member.

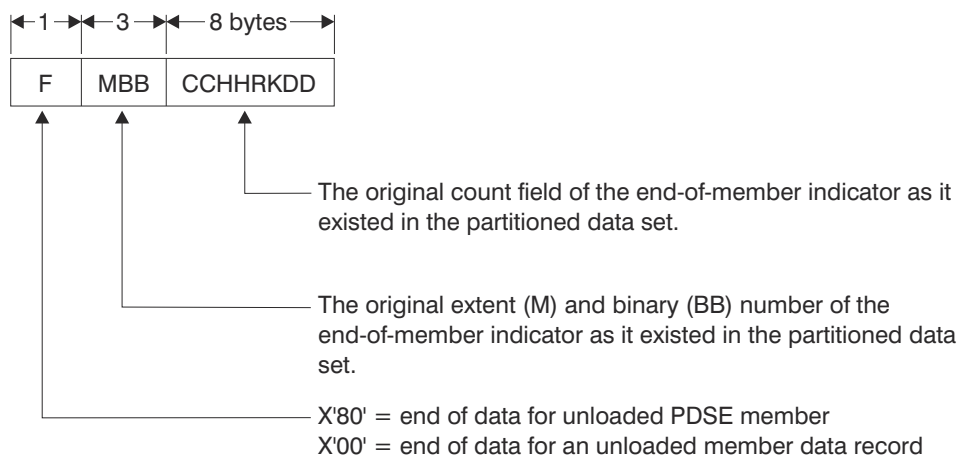


Figure 51. End-of-file block layout

Appendix C. Specifying User Exits with Utility Programs

This appendix documents Programming Interface and Associated Guidance Information.

This appendix is intended to help you write user exits for utility programs.

General Guidance

Exits can be specified with various utilities to:

- Modify physical records
- Handle I/O errors
- Process user input/output header and trailer labels.

The exits are specified in a parameter of the EXITS statement in the various utilities, except for IEBDG and IEBUPDTE. The exits available from utility programs are listed in [Table 66 on page 343](#).

Table 66. User-exit routines specified with utilities

Exit Routine	When Available	Utilities	Where Specified
Modify physical records before processing by IEBGENER	After the physical record is read and before any editing is performed	IEBGENER	DATA parameter of EXITS statement
Input header or trailer label	When the data set is opened for input (header) or closed (trailer)	IEBCOMPR, IEBGENER, IEBTPCH	INHDR/INTLR parameters of EXITS statement
		IEBUPDTE	PARM parameter of the EXEC JCL statement or INHDR, INTLR, OUTHDR, and OUTTLR parameter of ADD, CHANGE, REPL, or REPRO statements.
Output header or trailer label	When the data set is opened for output (header) or closed (trailer)	IEBCOMPR, IEBGENER	OUTHDR/OUTLR parameters of EXITS statement
Totaling	Prior to IEBGENER writing of each physical record (sequential data sets only)	IEBGENER	TOTAL parameter of EXITS statement
		IEBUPDTE	TOTAL parameter on the ADD, CHANGE, REPL, or REPRO Statements
I/O error	When permanent error occurs in IEBGENER	IEBGENER	IOERROR parameter of EXITS statement
Error detected by IEBCOMPR	After unequal comparison	IEBCOMPR	ERROR parameter of EXITS statement
Build output record key	Prior to IEBGENER writing of a record	IEBGENER	KEY parameter of EXITS statement
Process logical records of input data sets before comparison	Before input records are processed by IEBCOMPR	IEBCOMPR	PRECOMP parameter of EXITS statement
Process IEBTPCH input/output records	Before logical record is processed (INREC) or before logical record is written (OUTREC)	IEBTPCH	INREC/OUTREC parameters of EXITS statement

Table 66. User-exit routines specified with utilities (continued)

Exit Routine	When Available	Utilities	Where Specified
Analyze or modify IEBDG output record	After output record is constructed, but before it is placed in the output data set	IEBDG	EXIT parameter of CREATE statement

Register Contents at Entry to Routines from Utility Programs

Register Contents

1

Address of the parameter list.

13

Address of the register save area. The save area must not be used by user label processing routines.

14

Return address to utility.

15

Entry address to the exit routine.

Programming Considerations

The exit routine must reside in the job library, step library, or link library.

Returning from an Exit Routine

An exit routine returns control to the utility program by means of the RETURN macro instruction in the exit routine. Registers 1 through 14 must be restored before control is returned to the utility program.

The format of the RETURN macro instruction is:

Label	Statement	Parameters
[<i>label</i>]	RETURN	[(<i>r,r</i>)] [, RC={ <i>n</i> (15) }]

where:

(*r,r*)

specifies the range of registers, from 0 to 15, to be reloaded by the utility program from the register save area. For example, (14,12) indicates that all registers except register 13 are to be restored. If this parameter is omitted, the registers are considered properly restored by the exit routine.

RC={*n*|(15)}

specifies a decimal return code in register 15. If RC is omitted, register 15 is loaded as specified by (*r,r*).

n

specifies a return code to be placed in register 15.

(15)

specifies that general register 15 already contains a valid return code.

A user label processing routine must return a code in register 15 as shown in [Table 67 on page 345](#) unless:

- The buffer address was set to zero before entry to the label processing routine. In this case, the system resumes normal processing regardless of the return code.
- The user label processing routine was entered after an unrecoverable output error occurred. In this case the system tries to resume normal processing.

Table 67 on page 345 shows the return codes that can be issued to utility programs by user exit routines.

Table 67. Return codes that must be issued by user exit routines

Type of Exit	Return Code	Action
Input Header or Trailer Label (except for IEBUPDTE when UPDATE=INPLACE)	0	The system resumes normal processing. If there are more labels in the label group, they are ignored.
	4	The next user label is read into the label buffer area and control is returned to the user's routine. If there are no more labels, normal processing is resumed.
	16	The utility program is ended on request of the user routine.
Input Header or Trailer Label for IEBUPDTE UPDATE=INPLACE	0	The system resumes normal processing; any additional user labels are ignored.
	4	The system does not write the label. The next user label is read into the label buffer area and control is returned to the user exit routine. If there are no more user labels, the system resumes normal processing.
	8	The system writes the user labels from the label buffer area and resumes normal processing.
	12	The system writes the user label from the label buffer area, then reads the next input label into the label buffer area and returns control to the label processing routine. If there are no more user labels, the system resumes normal processing.
Output Header or Trailer Label	0	The system resumes normal processing. No label is written from the label buffer area.
	4	The user label is written from the label buffer area. The system then resumes normal processing.
	8	The user label is written from the label buffer area. If fewer than eight labels have been created, the user's routine again receives control so that it can create another user label. If eight labels have been created, the system resumes normal processing.
	16	The utility program is ended on request of the user routine.
Totaling Exits	0	Processing continues, but no further user exits are taken.
	4	Normal operation continues.
	8	Processing ceases, except for EOD processing on output data set (user label processing).
	16	Utility program is stopped.

Table 67. Return codes that must be issued by user exit routines (continued)

Type of Exit	Return Code	Action
ERROR	0	Record is not placed in the error data set. Processing continues with the next record.
	4	Record is placed in the error data set (SYSUT3).
	8	Record is not placed in error data set but is processed as a valid record (sent to OUTREC and SYSUT2 if specified).
	16	Utility program is ended.
OUTREC (IEBPTPCH)	4	Record is not placed in normal output data set.
	12 or 16	Utility program is ended.
	Any other number	Record is placed in normal output data set (SYSUT2).
All other exits	0-11 (Set to next multiple of four)	Return code is compared to highest previous return code; the higher is saved and the other discarded. At the normal end of job, the highest return code is passed to the calling processor.
	12 or 16	Utility program is stopped and this return code is passed to the calling processor.

Parameters Passed to Label Processing Routines

The parameters passed to a user label processing routine are addresses of: the 80-byte label buffer, the DCB being processed, the status information if an unrecoverable input/output error occurs, and the totaling area.

The 80-byte label buffer contains an image of the user label when an input label is being processed. When an output label is being processed, the buffer contains no significant information at entry to your label processing routine. When the utility program has been requested to generate labels, your label processing routine must construct a label in the label buffer.

If standard user labels (SUL) are specified on the DD statement for a data set, but the data set has no user labels, the system still takes the specified exits to the appropriate user routine. In such a case, the user input label processing routine is entered with the buffer address parameter set to zero.

The format and content of the DCB are explained in [z/OS DFSMS Macro Instructions for Data Sets](#).

Bit 0 of flag 1 in the DCB-address parameter is set to a value of 0 except when:

- Volume trailer or header labels are being processed at volume switch time.
- The trailer labels of a DISP=MOD data set are being processed (when the data set is opened).

If an unrecoverable input/output error occurs while reading or writing a user label, the appropriate label processing routine is entered with bit 0 of flag 2 in the status information address parameter set on. The three low order bytes of this parameter contain the address of standard status information as supplied for SYNAD routines. (The SYNAD routine is not entered.)

Parameters Passed to Nonlabel Processing Routines

Table 68 on page 347 shows the programs from which exits can be taken to nonlabel processing routines, the names of the exits, and the parameters available for each exit routine.

Table 68. Parameter lists for nonlabel processing exit routines

Program	Exit	Parameters
IEBGENER	KEY	Address at which key is to be placed (record follows key); address of DCB.
	DATA	Address of SYSUT1 record; address of DCB.
	IOERROR	Address of DECB; cause of the error and address of DCB. (Address in lower order three bytes and cause of error in high order byte.)
IEBCOMPR	ERROR	Address of DCB for SYSUT1; address of DCB for SYSUT2.
	PRECOMP	Address of SYSUT1 record; length of SYSUT1 record, address of SYSUT2 record; length of SYSUT2 record.
IEBTPCH	INREC	Address of input record; length of the input record.
	OUTREC	Address of output record; length of the output record.

Processing User Labels

User labels can be processed by IEBCOMPR, IEBGENER, IEBTPCH, IEBUPDTE, and IEHMOVE. In some cases, user-label processing is automatically performed; in other cases, you must indicate the processing to be performed. In general, you can:

- Process user labels as data set descriptors.
- Process user labels as data.
- Total the processed records before they are written (IEBGENER and IEBUPDTE only).

For either of the first two options, you must specify SUL on the DD statement that defines each data set for which user-label processing is desired. For totaling routines, OPTCD=T must be specified on the DD statement.

You cannot update labels by means of the IEBUPDTE program. This function must be performed by a user label processing routine. IEBUPDTE will, however, allow you to create labels on the output data set from data supplied in the input stream.

IEHMOVE does not allow exits to user routines and does not recognize options concerning the processing of user labels as data. IEHMOVE always moves or copies user labels directly to a new data set.

Volume switch labels of a multivolume data set cannot be processed by IEHMOVE, IEBGENER, or IEBUPDTE. Volume switch labels are lost when these utilities create output data sets. To ensure that volume switch labels are retained, process multivolume data sets one volume at a time.

Processing User Labels as Data Set Descriptors

When user labels are to be processed as data set descriptors, one of your label processing routines receives control for each user label of the specified type. Your routine can include, exclude, or modify the user label. Processing of user labels as data set descriptors is indicated on an EXITS statement with keyword parameters that name the label processing routine to be used.

The user exit routine receives control each time the OPEN, EOVS, or CLOSE routine encounters a user label of the type specified.

Figure 52 on page 348 illustrates the action of the system at OPEN, EOVS, or CLOSE time. When OPEN, EOVS, or CLOSE recognizes a user label and when SUL has been specified on the DD statement for the data set, control is passed to the utility program. Then, if an exit has been specified for this type of label, the utility program passes control to the user routine. Your routine processes the label and returns control, along with a return code, to the utility program. The utility program then returns control to OPEN, EOVS, or CLOSE.

This cycle is repeated up to eight times, depending upon the number of user labels in the group and the return codes supplied by your routine.

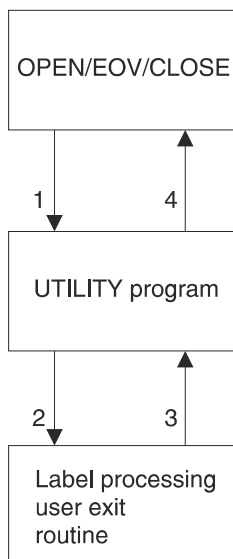


Figure 52. System action at OPEN, EOV, or CLOSE time

Exiting to a Totaling Routine

When an exit is taken to a totaling routine, an output record is passed to the routine just before the record is written. The first halfword of the totaling area pointed to by the parameter contains the length of the totaling area, and should not be used by your routine. If you have specified user label exits, this totaling area (or an image of this area) is pointed to by the parameter list passed to the appropriate user label routine.

An output record is defined as a physical record (block), except when IEBGENER is used to process and reformat a data set that contains spanned records.

The code that is returned by the totaling routine determines system response, as follows:

Codes

Meaning

00 (X'00')

Processing is to continue, but no further exits are to be taken.

04 (X'04')

Normal processing is to continue.

08 (X'08')

Processing is to stop, except for EOD processing on the output data set (user label processing).

16 (X'10')

Processing is to be stopped.

Processing User Labels as Data

When user labels are processed as data, the group of user labels, as well as the data set, is subject to the normal processing done by the utility program. You can have labels printed or punched by IEBTPCH, compared by IEBCOMPR, or copied by IEBGENER.

To specify that user labels are to be processed as data, include a LABELS statement in the job step that is to process user labels as data.

There is no direct relationship between the LABELS statement and the EXITS statement. Either or both can appear in the control statement stream for an execution of a utility program. If there are user label-processing routines, however, their return codes may influence the processing of the labels as data. In addition, a user output label-processing routine can override the action of a LABELS statement because it receives control before each output label is written. At this time, the label created by the utility as a result of the LABELS statement is in the label buffer, and your routine can modify it.

Using an Exit Routine with IEBDG

IEBDG provides a user exit so you can analyze or further modify a newly constructed record before it is placed in the output data set. This exit routine is specified on the CREATE statement.

The CREATE statement defines the contents of records to be made available to a user routine or to be written directly as output records.

When an exit routine is loaded and you return control to IEBDG, register 1 contains the address of the first byte of the output record.

After processing each potential output record, the user routine should provide a return code in register 15 to instruct IEBDG how to handle the output record. The return codes are listed as follows:

Codes

Meaning

00 (X'00')

The record is to be written.

04 (X'04')

The record is not to be written. The skipped record is not to be counted as a generated output record; processing is to continue as though a record were written. If skips are requested through user exits and input records are supplied, each skip causes an additional input record to be processed in the generation of output records. For example, if a CREATE statement specifies that 10 output records are to be generated and a user exit indicates that two records are to be skipped, 12 input records are processed.

12 (X'0C')

The processing of the remainder of this set of utility control statements is to be bypassed. Processing is to continue with the next DSD statement.

16 (X'10')

All processing is to halt.

Appendix D. IEHLIST VTOC Listing

IEHLIST Sample Output

Figure 53 on page 351 is a listing of a volume table of contents from an extended format sequential data set. The VTOC listing of an extended format sequential data set will differ from that of sequential and partitioned data sets and PDSEs. Figure 54 on page 352 is a IEHLIST VTOC listing that is typical for a sequential or partitioned data set, or a PDSE. The primary difference between these two kinds of VTOC listings (extended format sequential data sets and all other types of data sets) can be found in the LAST BLK field of the outputs. In addition, if IEHLIST is producing a VTOC listing of an extended format sequential data set, the attribute E will be included under the field SMS .IND. For more information on the characteristics of extended format sequential data sets, see *z/OS DFSMS Using Data Sets*.

A detailed explanation of the fields in the VTOC listing follows the figures. The explanation of the LAST BLK field contains two different entries—one for an extended format sequential data set (TTTT-R), and one for a sequential or partitioned data set, or PDSE (T-R-L).

The following figures are only examples of IEHLIST VTOC listings. Your actual VTOC listing produced by IEHLIST will differ.

For VSAM data sets, the LAST BLK(T-R-L) field might be blank.

SYSTEMS SUPPORT UTILITIES---IEHLIST															PAGE 1	
DATE: 1992.260		TIME: 17.13.16		CONTENTS OF VTOC ON VOL 1P0401 <THIS IS AN SMS MANAGED VOLUME>												
-----DATA SET NAME-----				SER NO	SEQNO	DATE.CRE	DATE.EXP	DATE.REF	EXT	DSORG	RECFM	OPTCD	BLKSIZE			
EXAMPLE.OF.FORMAT				1P0401	1	1992.260	00.000	1992.260	F2	OR	F3(C-H-R)	00	20480			
SMS.IND	LRECL	KEYLEN	INITIAL	ALLOC	2ND ALLOC	EXTEND	LAST BLK(TTTT-R)		DIR.REM							
S	E	20480	TRKS		1		19	2				0	1	6	0	
EXTENTS		NO	LOW(C-H)	HIGH(C-H)	NO	LOW(C-H)	HIGH(C-H)	NO	LOW(C-H)	HIGH(C-H)						
		0	0 4	0 4	1	1 0	1 0	2	1 1	1 1						
		3	1 5	1 5	4	1 6	1 6	5	1 7	1 7						
		6	1 8	1 8	7	1 9	1 9	8	1 10	1 10						
		9	1 11	1 11	10	1 12	1 12	11	1 13	1 13						
		12	1 14	1 14	13	2 0	2 0	14	2 1	2 1						
		15	2 2	2 2												
		16	2 3	2 3	17	2 4	2 4	18	2 5	2 5						
		19	2 6	2 6												
---ON THE ABOVE DATA SET,THERE ARE 0 EMPTY TRACK(S).																

Figure 53. IEHLIST sample output—VTOC (for extended format sequential data sets)

SYSTEMS SUPPORT UTILITIES---IEHLIST															PAGE 1		
DATE: 2008.064 TIME: 11.12.55																	
CONTENTS OF VTOC ON VOL 1P0301 <THIS IS AN SMS MANAGED VOLUME>																	
THERE IS A 1 LEVEL VTOC INDEX																	
DATA SETS ARE LISTED IN ALPHANUMERIC ORDER																	
FORMAT 4	DSCB	NO AVAIL/MAX	DSCB /MAX	DIRECT	NO AVAIL	NEXT ALT	FORMAT 6	LAST FMT 1	VTOC EXTENT	THIS DSCB							
VI	DSCBS	PER TRK	BLK PER TRK		ALT TRK	TRK(C-H)	(C-H-R)	DSCB(C-H-R)	/LOW(C-H)	HIGH(C-H)	(C-H-R)	(C-H-R)	(C-H-R)	(C-H-R)	(C-H-R)		
81	697	50	45		0	0	0	1	0	50	0	2	1	0	0	2 1	
-----DATA SET NAME-----																	
SER NO		SEQNO		DATE.CRE		DATE.EXP		DATE.REF		EXT		DSORG		RECFM		OPTCD	BLKSIZE
EXAMPL		1		1987.284		00.000		00.000		1		PS		F		00 2048	
SMS.IND	LRECL	KEYLEN	INITIAL	ALLOC	2ND	ALLOC	EXTEND	LAST		BLK(T-R-L)		DIR.REM		F2 OR F3(C-H-R)		DSCB(C-H-R)	
SU	2048		TRKS	CONTIG		0		13		21		0				0 2 3	
EXTENTS		NO		LOW(C-H)		HIGH(C-H)											
		0		2		0		2		13							
----ON THE ABOVE DATA SET, THERE ARE 0 EMPTY TRACK(S).																	
VPSM																	
A = NUMBER OF TRKS IN ADDITION TO FULL CYLS IN THE EXTENT																	
TRK	FULL	A	TRK	FULL	A	TRK	FULL	A	TRK	FULL	A	TRK	FULL	A	TRK	FULL	
ADDR	CYLS		ADDR	CYLS		ADDR	CYLS		ADDR	CYLS		ADDR	CYLS		ADDR	CYLS	
1	0	1	16	0	14	44	21	1									
THERE ARE 21 EMPTY CYLINDERS PLUS 16 EMPTY TRACKS ON THIS VOLUME																	
THERE ARE 21 EMPTY CYLINDERS PLUS 16 EMPTY TRACKS FROM THE TRACK-MANAGED SPACE																	
THERE ARE 697 BLANK DSCBS IN THE VTOC ON THIS VOLUME																	
THERE ARE 290 UNALLOCATED VIRS IN THE INDEX																	

Figure 54. IEHLIST sample output—VTOC (for sequential, partitioned data sets and PDSEs)

IEHLIST Sample Output for Data Sets Supporting Extended Attribute DSCBs

“IEHLIST Sample Output— a table of contents of a volume that has data sets that support extended attribute DSCBs.” on page 352 and “IEHLIST DUMP Sample Output— format 8 and 9 DSCBs present for data sets that support the extended attribute DSCBs.” on page 355 shows corresponding contents and dump IEHLIST sample output for data sets supporting extended attribute DSCBs.

“IEHLIST Sample Output— a table of contents of a volume that has data sets that support extended attribute DSCBs.” on page 352 shows the table of contents of a volume that has data sets that support extended attribute DSCBs. In this specific example, these data sets are described by the VNDM7AG.TEST.VSAM* qualifier. For data sets that support extended attribute DSCBs, the extent descriptions are adjusted to support larger cylinder numbers. For volumes that support cylinder-managed space the format 4 DSCB display identifies the size of the volume, the location of the address of the cylinder where cylinder-managed space begins, and the allocation size of multiple cylinder unit that the system uses to allocate space in cylinder-managed space.

At the end of the report there is a listing of the number of empty cylinders and additional empty tracks for the track-managed space of volumes that support cylinder-managed space. In the VPSM report of free extents, the TRK ADDR and FULL CYLS columns are adjusted to support larger numbers for the track address and full cylinders when the volume is an EAV.

“IEHLIST DUMP Sample Output— format 8 and 9 DSCBs present for data sets that support the extended attribute DSCBs.” on page 355 shows the sample output of the IEHLIST DUMP command with format 8 and format 9 DSCBs that are present for data sets that support extended attribute DSCBs. This display corresponds to the IEHLIST FORMAT command output in “IEHLIST Sample Output— a table of contents of a volume that has data sets that support extended attribute DSCBs.” on page 352.

IEHLIST Sample Output— a table of contents of a volume that has data sets that support extended attribute DSCBs.

SYSTEMS SUPPORT UTILITIES---IEHLIST														PAGE 1					
DATE: 2008.302 TIME: 07.04.01																			
CONTENTS OF VTOC ON VOL IN7907 <THIS IS AN SMS MANAGED VOLUME>																			
THERE IS A 1 LEVEL VTOC INDEX																			
DATA SETS ARE LISTED IN ALPHANUMERIC ORDER																			
FORMAT 4		DSCB		NO AVAIL/MAX DSCB		/MAX DIRECT		NO AVAIL		NEXT ALT		FORMAT 6		LAST FMT 1		VTOC EXTENT		THIS DSCB	
VI		DSCBS		PER TRK		BLK PER TRK		ALT TRK		TRK(C-H)		(C-H-R)		DSCB(C-H-R)		/LOW(C-H)		HIGH(C-H)	
81		4996		50		45		0		0		0		6		10		50	
0		1		6		10		0		1		6		10		0		1	
0		1		1															
NUMBER OF MULTICYLINDER UNITS																			
CYLINDERS FIRST CYL ADDR SPACE																			

Appendix D. IEHLIST VTOC Listing **353**

```

-----ON THE ABOVE DATA SET, THERE ARE 0 EMPTY CYLINDER(S) PLUS 8 EMPTY TRACK(S).
SYSTEMS SUPPORT UTILITIES---IEHLIST PAGE 4

-----DATA SET NAME----- SER NO SEQNO DATE.CRE DATE.EXP DATE.REF EXT DSORG RECFM OPTCD BLKSIZE
VNDM7VM.EFS.NS IN7907 1 2008.302 00.000 2008.302 1 PS FB 00 800
SMS.IND LRECL KEYLEN INITIAL ALLOC 2ND ALLOC EXTEND LAST BLK(TTTT-R) DIR.REM F2 OR F3(C-H-R) DSCB(C-H-R)
S E 80 CYLS 30 168 16 0 2 39
EATTR
NS
EXTENTS NO LOW(C-H) HIGH(C-H)
0 1618 0 1647 14
-----ON THE ABOVE DATA SET, THERE ARE 18 EMPTY CYLINDER(S) PLUS 11 EMPTY TRACK(S).

-----DATA SET NAME----- SER NO SEQNO DATE.CRE DATE.EXP DATE.REF EXT DSORG RECFM OPTCD BLKSIZE
VNDM7VM.EFS.OPT IN7907 1 2008.302 00.000 2008.302 1 PS FB 00 800
SMS.IND LRECL KEYLEN INITIAL ALLOC 2ND ALLOC EXTEND LAST BLK(TTTT-R) DIR.REM F2 OR F3(C-H-R) DSCB(C-H-R)
S E 80 CYLS 30 168 16 0 2 36
EATTR JOB STEP CREATE TIME
OPT CREEAS DEFINE1 06:53:33.556983
EXTENTS NO LOW(C-H) HIGH(C-H)
0 66948 0 66989 14
-----ON THE ABOVE DATA SET, THERE ARE 30 EMPTY CYLINDER(S) PLUS 11 EMPTY TRACK(S).

-----DATA SET NAME----- SER NO SEQNO DATE.CRE DATE.EXP DATE.REF EXT DSORG RECFM OPTCD BLKSIZE
VNDM7VM.SEQ.NO IN7907 1 2008.302 00.000 2008.302 15 PS F 00 80
SMS.IND LRECL KEYLEN INITIAL ALLOC 2ND ALLOC EXTEND LAST BLK(T-R-L) DIR.REM F2 OR F3(C-H-R) DSCB(C-H-R)
S R A 80 CYLS 4 820 40 28866 0 2 11 0 2 8
EATTR
NO
EXTENTS NO LOW(C-H) HIGH(C-H) NO LOW(C-H) HIGH(C-H) NO LOW(C-H) HIGH(C-H)
0 1482 0 1483 14 1 1531 0 1534 14 2 1535 0 1538 14
3 1539 0 1542 14 4 1543 0 1546 14 5 1547 0 1550 14
6 1551 0 1554 14 7 1555 0 1558 14 8 1559 0 1562 14
9 1563 0 1566 14 10 1567 0 1570 14 11 1571 0 1574 14
12 1575 0 1578 14 13 1579 0 1582 14 14 1583 0 1586 14
-----ON THE ABOVE DATA SET, THERE ARE 3 EMPTY CYLINDER(S) PLUS 4 EMPTY TRACK(S).
SYSTEMS SUPPORT UTILITIES---IEHLIST PAGE 5

-----DATA SET NAME----- SER NO SEQNO DATE.CRE DATE.EXP DATE.REF EXT DSORG RECFM OPTCD BLKSIZE
VNDM7VM.SEQ.NS IN7907 1 2008.302 00.000 00.000 1 PS F 00 80
SMS.IND LRECL KEYLEN INITIAL ALLOC 2ND ALLOC EXTEND LAST BLK(T-R-L) DIR.REM F2 OR F3(C-H-R) DSCB(C-H-R)
S B 80 CYLS 4 0 0 58786 0 2 10
EATTR
NS
EXTENTS NO LOW(C-H) HIGH(C-H)
0 1489 0 1490 14
-----ON THE ABOVE DATA SET, THERE ARE 2 EMPTY CYLINDER(S) PLUS 0 EMPTY TRACK(S).

-----DATA SET NAME----- SER NO SEQNO DATE.CRE DATE.EXP DATE.REF EXT DSORG RECFM OPTCD BLKSIZE
VNDM7VM.SEQ.OPT IN7907 1 2008.302 00.000 2008.302 2 PS F 00 80
SMS.IND LRECL KEYLEN INITIAL ALLOC 2ND ALLOC EXTEND LAST BLK(T-R-L) DIR.REM F2 OR F3(C-H-R) DSCB(C-H-R)
S R 80 CYLS 30 820 40 28866 0 2 6
EATTR
OPT
EXTENTS NO LOW(C-H) HIGH(C-H) NO LOW(C-H) HIGH(C-H)
0 1452 0 1481 14 1 1501 0 1530 14
-----ON THE ABOVE DATA SET, THERE ARE 5 EMPTY CYLINDER(S) PLUS 4 EMPTY TRACK(S).

-----DATA SET NAME----- SER NO SEQNO DATE.CRE DATE.EXP DATE.REF EXT DSORG RECFM OPTCD BLKSIZE
VNDM7VM.VSAM.NO.DATA IN7907 1 2008.302 00.000 2008.302 1 VS U 80 4096
SMS.IND LRECL KEYLEN INITIAL ALLOC 2ND ALLOC EXTEND LAST BLK(T-R-L) DIR.REM F2 OR F3(C-H-R) DSCB(C-H-R)
S A 0 CYLS 21 0 0 0 0 2 29
EATTR
NO
EXTENTS NO LOW(C-H) HIGH(C-H)
0 1431 0 1451 14
-----UNABLE TO CALCULATE EMPTY SPACE.

-----DATA SET NAME----- SER NO SEQNO DATE.CRE DATE.EXP DATE.REF EXT DSORG RECFM OPTCD BLKSIZE
VNDM7VM.VSAM.NO.INDEX IN7907 1 2008.302 00.000 00.000 1 VS U 80 4096
SMS.IND LRECL KEYLEN INITIAL ALLOC 2ND ALLOC EXTEND LAST BLK(T-R-L) DIR.REM F2 OR F3(C-H-R) DSCB(C-H-R)
S A 0 TRKS CONTIG 1 0 0 0 0 2 30
EATTR
NO
EXTENTS NO LOW(C-H) HIGH(C-H)
0 1373 4 1373 4
-----UNABLE TO CALCULATE EMPTY SPACE.
SYSTEMS SUPPORT UTILITIES---IEHLIST PAGE 6

-----DATA SET NAME----- SER NO SEQNO DATE.CRE DATE.EXP DATE.REF EXT DSORG RECFM OPTCD BLKSIZE
VNDM7VM.VSAM.NS.DATA IN7907 1 2008.302 00.000 2008.302 1 VS U 80 4096
SMS.IND LRECL KEYLEN INITIAL ALLOC 2ND ALLOC EXTEND LAST BLK(T-R-L) DIR.REM PTR TO F3(C-H-R) DSCB(C-H-R)
S 0 CYLS 21 0 0 0 0 2 31
EATTR JOB STEP CREATE TIME
NS CREPS DEFINE1 06:15:38.309957
EXTENTS NO LOW(C-H) HIGH(C-H)
0 67011 0 67031 14
-----UNABLE TO CALCULATE EMPTY SPACE.

-----DATA SET NAME----- SER NO SEQNO DATE.CRE DATE.EXP DATE.REF EXT DSORG RECFM OPTCD BLKSIZE
VNDM7VM.VSAM.NS.INDEX IN7907 1 2008.302 00.000 00.000 1 VS U 80 4096
SMS.IND LRECL KEYLEN INITIAL ALLOC 2ND ALLOC EXTEND LAST BLK(T-R-L) DIR.REM PTR TO F3(C-H-R) DSCB(C-H-R)
S 0 TRKS CONTIG 1 0 0 0 0 2 33
EATTR JOB STEP CREATE TIME
NS CREPS DEFINE1 06:15:38.321190
EXTENTS NO LOW(C-H) HIGH(C-H)
0 1373 5 1373 5
-----UNABLE TO CALCULATE EMPTY SPACE.

-----DATA SET NAME----- SER NO SEQNO DATE.CRE DATE.EXP DATE.REF EXT DSORG RECFM OPTCD BLKSIZE
VNDM7VM.VSAM.OPT1.DATA IN7907 1 2008.302 00.000 2008.302 12 VS U 80 4096
SMS.IND LRECL KEYLEN INITIAL ALLOC 2ND ALLOC EXTEND LAST BLK(T-R-L) DIR.REM PTR TO F3(C-H-R) DSCB(C-H-R)
S 0 TRKS CONTIG 3 0 2 27 0 2 23
EATTR JOB STEP CREATE TIME
OPT CREPS DEFINE1 06:03:07.903068
EXTENTS NO LOW(C-H) HIGH(C-H) NO LOW(C-H) HIGH(C-H)
0 1085 1 1085 3 1 1085 5 1086 13 2 1319 0 1336 2
3 1336 4 1345 9 4 1345 11 1355 1 5 1355 3 1364 8
6 1364 10 1373 0 7 1373 6 1373 14 8 1375 0 1375 5

```

[illegible]Appendix D. IEHLIST VTOC Listing **355**

[illegible]

Figure 55 on page 356 and Figure 56 on page 357 show IEHLIST sample output that includes vendor data.

Figure 55. IEHLIST sample output—VTOC showing vendor data

Figure 56. DUMP sample output showing vendor data

Field	Explanation
BLKSIZE	Block size, in bytes, up to 32760 or device maximum. <ul style="list-style-type: none">For fixed-length records, block size is set.For variable or undefined-length records, maximum block size is indicated.Format V unblocked records have a block size 4 greater than the LRECL value.
CREATE TIME	Time of creation for the data set. This will be the number of microseconds since midnight in the format <i>hh:mm:ss.mmmmmm</i> where: <ul style="list-style-type: none"><i>hh</i> is the hours<i>mm</i> is the minutes<i>ss</i> is the seconds<i>mmmmmm</i> is the number of microseconds
DATA SET NAME	Maximum length 44 bytes.
DATE.CRE	Creation date for the data set, in the Julian form <i>yyyy.ddd</i> , where <i>ddd</i> is the day and <i>yyyy</i> is the year from 1900 to 2155.
DATE.EXP	Expiration date for the data set, in the Julian form <i>yyyy.ddd</i> , where <i>ddd</i> is the day and <i>yyyy</i> is the year from 1900 to 2155.
DATE.REF	Last referenced date for the data set, in the Julian form <i>yyyy.ddd</i> where <i>ddd</i> is the day and <i>yyyy</i> is the year from 1900 to 2155.
DIR.REM	In a partitioned data set in which the last directory block is being used, this value will be the number of bytes consumed in that 256-byte block. If no value appears here, the partitioned data set has not yet reached the last directory block.
DSORG	Data set organization (by access method): <ul style="list-style-type: none">DA = Direct (BDAM)PO = Partitioned (BPAM)PS = Physical Sequential (SAM, QSAM, BSAM) The following condition may also appear after any of the preceding organizations: <ul style="list-style-type: none">U = Unmovable (location-dependent).
EATTR	The current EATTR setting for the data set, which specifies whether a data set can have extended attributes (format 8 and 9 DSCBs) and optionally reside in EAS: <ul style="list-style-type: none">NS = EATTR has not been specified. The defaults for EAS eligibility should apply.NO = No extended attributes. The data set can not have extended attributes (format 8 and 9 DSCBs) or optionally reside in EAS. This is the default behavior for non-VSAM data sets.OPT = Extended attributes are optional. The data set can have extended attributes (format 8 and 9 DSCBs) and can optionally reside in EAS. This is the default behavior for VSAM data sets.
EXT	Number of extents (sections) the data set has on this volume.
EXTEND	Original secondary allocation quantity if type of space request was bytes, kilobytes or megabytes. Original average block length if type of space request was average block. The actual secondary value is followed by one of the following 2-character identifiers: <ul style="list-style-type: none">AV Average block lengthBY Original secondary quantity in bytesKB Original secondary quantity in kilobytesMB Original secondary quantity in megabytes

Field	Explanation
EXTENT NO LOW (C-H) HIGH (C-H)	The cylinder and head (track) address of each extent.
FMT 2 OR 3 (C-H-R)/DSCB (C-H-R)	<p>Two addresses are possible here, each pointing to a data set control block (DSCB) in the VTOC. The cylinder-head(track)-record address on the right always appears and points to the DSCB whose partial contents you are now looking at: the Format 1 or 8 DSCB.</p> <p>There may also be a Format 2 or Format 3 DSCB associated with it. The Format 3 address will be present only for data sets that have exceeded three extents, such that a Format 3 DSCB must be used to contain information about the additional extents.</p>
INITIAL ALLOC	<p>Describes the space attribute that was used for allocating all data set extents.</p> <ul style="list-style-type: none"> • RECS = average block size • TRKS = Tracks • BLKS = Blocks • CYLS = Cylinders • ABSTR = Absolute tracks (absolute addresses)
JOB	Job name (eight bytes in EBCDIC).
KEYLEN	Byte length (1-255) of the key of the data records in this data set. 0 indicates that no key exists.
LAST BLK PTR (T-R-L)	Points to the last block written in a sequential or partitioned data set or PDSE. The first number (two digits in this example) is the track, relative to the beginning of the data set. The second number is the block, relative to the beginning of the track. The last number is the number of bytes remaining on the track following that block. If the data set is a PDSE, then the field may be blank.
LAST BLK PTR (TTTT-R)	Points to the last block written in an extended format sequential data set. The first number (four bytes but two bytes in this example) is the track, relative to the beginning of the data set. The second number is the block, relative to the beginning of the track.
LRECL	<p>Logical record length, in bytes, up to 32760 for nonspanned and 32756 for spanned records.</p> <ul style="list-style-type: none"> • For fixed-length records, LRECL is the actual record length. • For variable-length records, LRECL is the maximum length permitted by the device. • For undefined-length records, LRECL is zero.
MULTICYLINDER UNITS FIRST CYL ADDR	First cylinder address where cylinder-managed space begins on the volume.
MULTICYLINDER UNITS SPACE	The fixed unit of disk space in cylinders that is larger than a cylinder. This is the minimum allocation unit size for cylinder-managed space. Each extent in cylinder-managed space must have its first cylinder number and total size a multiple of this number of cylinders.
NUMBER OF CYLINDERS	Number of cylinders available on the volume.
OPTCD	Option code (as supplied in the DCB used to create the data set). This 1-byte code is given in hexadecimal characters. See the DS1OPTCD field in the DSCB1 data area in z/OS DFSMSdfp Advanced Services .
RECFM	<p>Record format:</p> <ul style="list-style-type: none"> • F = Fixed length • V = Variable length • D = ASCII variable length (not valid on disk) • U = Undefined length. <p>The following options may also be specified:</p> <ul style="list-style-type: none"> • B = Blocked records • S = Spanned records • T = Track overflow permitted • A = ISO/ANSI control characters • M = Machine control characters.
SEQNO	Order of this volume relative to the first volume containing the data set. (SEQ NO will be equal to 1, unless this is a multivolume data set.)

Field	Explanation
SER NO	Serial number of volume containing the data set. Maximum length 6 bytes. (The serial number may vary if the volume has been renamed since the data set was written, but this field should be the same for each format 1 DSCB for the data set.)
SMS.IND	<p>System-managed storage attributes.</p> <ul style="list-style-type: none"> • A = Extended attributes exist for the data set • B = Optimal block size selected by DADSM create • C = Compressed format • E = Extended format • H = HFS type of data set • I = Data set is a PDSE • L = Large format data set • N = Encrypted data set • R = Data set is reblockable • S = SMS-managed data set • U = No BCS entry exists for data set • ? = One of the following: <ul style="list-style-type: none"> – PDSE and extended format sequential data sets cannot coexist. Both bits are on – one must be turned off. – Extended format sequential data set bit must be on when the compressed extended format data set bit is on.
STEP	Step name (eight bytes in EBCDIC).
VENDOR CODE	Vendor identification defined by the vendor in field DS9ATRV1 in the format 9 DSCB. See Format-9 DSCB in z/OS DFSMSdfp Advanced Services .
VENDOR DATA	Beginning of variable-length field (up to 15 bytes) containing vendor data defined by the vendor in field DS9ATRV1 in the format 9 DSCB. See Format-9 DSCB in z/OS DFSMSdfp Advanced Services .
2ND ALLOC	Secondary allocation quantity. If zero, the data set is limited to its primary allocated extent; otherwise, it can expand as necessary into many more extents, each of which is this number of blocks, tracks, or cylinders in size. There are various limits.

Appendix E. Accessibility

Accessible publications for this product are offered through [IBM Documentation for z/OS \(www.ibm.com/docs/en/zos\)](http://www.ibm.com/docs/en/zos).

If you experience difficulty with the accessibility of any z/OS documentation see [How to Send Feedback to IBM](#) to leave documentation feedback.

Notices

This information was developed for products and services that are offered in the USA or elsewhere.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

*IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
United States of America*

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

*Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan*

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

This information could include missing, incorrect, or broken hyperlinks. Hyperlinks are maintained in only the HTML plug-in output for IBM Documentation. Use of hyperlinks in other output formats of this information is at your own risk.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

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

*IBM Corporation
Site Counsel
2455 South Road*

Poughkeepsie, NY 12601-5400
USA

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

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Terms and conditions for product documentation

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability

These terms and conditions are in addition to any terms of use for the IBM website.

Personal use

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial use

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or

reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

IBM Online Privacy Statement

IBM Software products, including software as a service solutions, ("Software Offerings") may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user, or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering's use of cookies is set forth below.

Depending upon the configurations deployed, this Software Offering may use session cookies that collect each user's name, email address, phone number, or other personally identifiable information for purposes of enhanced user usability and single sign-on configuration. These cookies can be disabled, but disabling them will also eliminate the functionality they enable.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM's Privacy Policy at ibm.com/privacy and IBM's Online Privacy Statement at ibm.com/privacy/details in the section entitled "Cookies, Web Beacons and Other Technologies," and the "IBM Software Products and Software-as-a-Service Privacy Statement" at ibm.com/software/info/product-privacy.

Policy for unsupported hardware

Various z/OS elements, such as DFSMSdfp, JES2, and MVS, contain code that supports specific hardware servers or devices. In some cases, this device-related element support remains in the product even after the hardware devices pass their announced End of Service date. z/OS may continue to service element code; however, it will not provide service related to unsupported hardware devices. Software problems related to these devices will not be accepted for service, and current service activity will cease if a problem is determined to be associated with out-of-support devices. In such cases, fixes will not be issued.

Minimum supported hardware

The minimum supported hardware for z/OS releases identified in z/OS announcements can subsequently change when service for particular servers or devices is withdrawn. Likewise, the levels of other software products supported on a particular release of z/OS are subject to the service support lifecycle of those

products. Therefore, z/OS and its product publications (for example, panels, samples, messages, and product documentation) can include references to hardware and software that is no longer supported.

- For information about software support lifecycle, see: [IBM Lifecycle Support for z/OS \(www.ibm.com/software/support/systemsz/lifecycle\)](http://www.ibm.com/software/support/systemsz/lifecycle)
- For information about currently-supported IBM hardware, contact your IBM representative.

Programming Interface Information

This book is intended to help you use the DFSMS utility programs.

This book also documents General-use Programming Interface and Associated Guidance Information provided by DFSMS. General-use programming interfaces allow the customer to write programs that obtain the services of DFSMS.

General-use Programming Interface and Associated Guidance Information is identified where it occurs, either by an introductory statement to a chapter or section or by the following marking: General-use Programming Interface and Associated Guidance Information.

This book contains no programming interface information except where stated otherwise. That exception is Appendixes A, B, and C.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at [Copyright and Trademark information \(www.ibm.com/legal/copytrade.shtml\)](http://www.ibm.com/legal/copytrade.shtml).

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, and service names may be trademarks or service marks of others.

Glossary

This glossary defines technical terms and abbreviations used in DFSMS documentation. If you do not find the term you are looking for, refer to the index of the appropriate DFSMS manual.

This glossary includes terms and definitions from:

- The *American National Standard Dictionary for Information Systems*, ANSI X3.172-1990, copyright 1990 by the American National Standards Institute (ANSI). Copies may be purchased from the American National Standards Institute, 11 West 42nd Street, New York, New York 10036. Definitions are identified by the symbol (A) after the definition.
- The *Information Technology Vocabulary* developed by Subcommittee 1, Joint Technical Committee 1, of the International Organization for Standardization and the International Electrotechnical Commission (ISO/IEC JTC1/SC1). Definitions of published part of this vocabulary are identified by the symbol (I) after the definition; definitions taken from draft international standards, committee drafts, and working papers being developed by ISO/IEC JTC1/SC1 are identified by the symbol (T) after the definition, indicating that final agreement has not yet been reached among the participating National Bodies of SC1.
- The *IBM Dictionary of Computing*, New York: McGraw-Hill, 1994.

The following cross-reference is used in this glossary:

See:

This refers the reader to (a) a related term, (b) a term that is the expanded form of an abbreviation or acronym, or (c) a synonym or more preferred term.

active control data set (ACDS)

A VSAM linear data set that contains a copy of the most recently activated SMS configuration and subsequent updates. The ACDS is shared by each system that is using the same SMS configuration to manage storage. See also *source control data set and communications data set*.

active data

(1) Data that can be accessed without any special action by the user, such as data on primary storage or migrated data. Active data also can be stored on tape volumes. (2) For tape mount management, application data that is frequently referenced, small in size, and managed better on DASD than on tape. (3) Contrast with *inactive data*.

aggregate backup

The process of copying an aggregate group and recovery instructions so that a collection of data sets can be recovered later as a group.

aggregate group

A collection of related data sets and control information that have been pooled to meet a defined backup or recovery strategy.

automated tape library data server

A device consisting of robotic components, cartridge storage areas, tape subsystems, and controlling hardware and software, together with the set of tape volumes that reside in the library and can be mounted on the library tape drives. Contrast with *manual tape library*. See also *tape library*.

automatic backup

(1) In DFSMSHsm, the process of automatically copying data sets from primary storage volumes or migration volumes to backup volumes. (2) In OAM, the process of automatically copying a primary copy of an object from disk, or an optical or tape volume to a backup volume contained in an object backup storage group. automatic class selection (ACS) routine. A procedural set of ACS language statements. Based on a set of input variables, the ACS language statements generate the name of a predefined SMS class, or a list of names of predefined storage groups, for a data set.

automatic data set protection (ADSP)

In z/OS, a user attribute that causes all permanent data sets created by the user to be automatically defined to RACF with a discrete RACF profile.

automatic dump

In DFSMSHsm, the process of using DFSMSdss automatically to do a full-volume dump of all allocated space on a primary storage volume to designated tape dump volumes.

automatic primary-space management

In DFSMSHsm, the process of deleting expired data sets, deleting temporary data sets, releasing unused space, and migrating data sets from primary storage volumes automatically.

automatic secondary-space management

In DFSMSHsm, the process of automatically deleting expired migrated data sets, deleting expired records from the migration control data sets, and migrating eligible data sets from migration level 1 volumes to migration level 2 volumes.

automatic volume-space management

In DFSMSHsm, the process that includes automatic primary space management and interval migration.

availability

For a storage subsystem, the degree to which a data set or object can be accessed when requested by a user.

backup

The process of creating a copy of a data set or object to be used in case of accidental loss.

basic format

The format of a data set that has a data set name type (DSNTYPE) of BASIC. A basic format data set is a sequential data set that is specified to be neither large format nor extended format. The size of a basic format data set cannot exceed 6 5 535 tracks on each volume.

base configuration

The part of an SMS configuration that contains general storage management attributes, such as the default management class, default unit, and default device geometry. It also identifies the systems or system groups that an SMS configuration manages.

basic catalog structure (BCS)

The name of the catalog structure in the catalog environment.

binder

The DFSMS program that processes the output of language translators and compilers into an executable program (load module or program object). It replaces the linkage editor and batch loader in z/OS.

cache fast write

A storage control capability in which the data is written directly to cache without using nonvolatile storage. Cache fast write is useful for temporary data or data that is readily recreated, such as the sort work files created by DFSORT. Contrast with *DASD fast write*.

capacity planning

The process of forecasting and calculating the appropriate amount of physical computing resources required to accommodate an expected workload.

Cartridge System Tape

The base tape cartridge media used with 3480 or 3490 Magnetic Tape Subsystems. Contrast with *Enhanced Capacity Cartridge System Tape*.

class transition

An event that brings about change to an object's service-level criteria, causing OAM to invoke ACS routines to assign a new storage class or management class to the object.

compress

(1) To reduce the amount of storage required for a given data set by having the system replace identical words or phrases with a shorter token associated with the word or phrase. (2) To reclaim the unused and unavailable space in a partitioned data set that results from deleting or modifying members by moving all unused space to the end of the data set.

compressed format

A particular type of extended-format data set specified with the (COMPACTION) parameter of data class. VSAM can compress individual records in a compressed-format data set. SAM can compress individual blocks in a compressed-format data set. See *compress*.

communications data set (COMMDS)

The primary means of communication among systems governed by a single SMS configuration. The COMMDS is a VSAM linear data set that contains the name of the ACDS and current utilization statistics for each system-managed volume, which helps balance space among systems running SMS. See also *active control data set* and *source control data set*.

concurrent copy

A function to increase the accessibility of data by enabling you to make a consistent backup or copy of data concurrent with the usual application program processing.

connectivity

(1) The considerations regarding how storage controls are joined to DASD and processors to achieve adequate data paths (and alternative data paths) to meet data availability needs. (2) In a system-managed storage environment, the system status of volumes and storage groups.

convert in place

See *in-place conversion*.

DASD fast write

An extended function of some models of the IBM 3990 Storage Control in which data is written concurrently to cache and nonvolatile storage and automatically scheduled for destaging to DASD. Both copies are retained in the storage control until the data is completely written to the DASD, providing data integrity equivalent to writing directly to the DASD. Use of DASD fast write for system-managed data sets is controlled by storage class attributes to improve performance. See also *dynamic cache management*. Contrast with *cache fast write*.

DASD volume

A DASD space identified by a common label and accessed by a set of related addresses. See also *volume*, *primary storage*, *migration level 1*, *migration level 2*.

data class

A collection of allocation and space attributes, defined by the storage administrator, that are used to create a data set.

Data Facility Sort

An IBM licensed program that is a high-speed data processing utility. DFSORT provides an efficient and flexible way to handle sorting, merging, and copying operations, as well as providing versatile data manipulation at the record, field, and bit level.

data set

In DFSMS, the major unit of data storage and retrieval, consisting of a collection of data in one of several prescribed arrangements and described by control information to which the system has access. In z/OS non-UNIX environments, the terms data set and file are generally equivalent and sometimes are used interchangeably. See also *file*. In z/OS UNIX environments, the terms data set and file have quite distinct meanings.

default device geometry

Part of the SMS base configuration, it identifies the number of bytes per track and the number of tracks per cylinder for converting space requests made in tracks or cylinders into bytes, when no unit name has been specified.

default management class

Part of the SMS base configuration, it identifies the management class that should be used for system-managed data sets that do not have a management class assigned.

default unit

Part of the SMS base configuration, it identifies an esoteric (such as SYSDA) or generic (such as 3390) device name. If a user omits the UNIT parameter on the JCL or the dynamic allocation equivalent, SMS applies the default unit if the data set has a disposition of MOD or NEW and is not system-managed.

Device Support Facilities (ICKDSF)

A program used for initialization of DASD volumes and track recovery.

DFSMS environment

An environment that helps automate and centralize the management of storage. This is achieved through a combination of hardware, software, and policies. In the DFSMS environment for z/OS, this function is provided by DFSMS, DFSORT, and RACF. See also *system-managed storage*.

DFSMSdfp

A DFSMS functional component or base element of z/OS, that provides functions for storage management, data management, program management, device management, and distributed data access.

DFSMSdss

A DFSMS functional component or base element of z/OS, used to copy, move, dump, and restore data sets and volumes.

DFSMShsm

A DFSMS functional component or base element of z/OS, used for backing up and recovering data, and managing space on volumes in the storage hierarchy.

DFSMShsm-managed volume

(1) A primary storage volume, which is defined to DFSMShsm but which does not belong to a storage group. (2) A volume in a storage group, which is using DFSMShsm automatic dump, migration, or backup services. Contrast with *system-managed volume*, *DFSMSrmm-managed volume*.

DFSMShsm control data set

In DFSMShsm, one of three VSAM key-sequenced data sets that contain records used in DFSMShsm processing. See also *backup control data set*, *migration control data set*, *offline control data set*.

DFSMS

See *Data Facility Storage Management Subsystem*.

DFSMSrmm

A DFSMS functional component or base element of z/OS, that manages removable media. DFSMSrmm-managed volume. A tape volume that is defined to DFSMSrmm. Contrast with *system-managed volume*, *DFSMShsm-managed volume*.

drive definition

A set of attributes used to define an optical disk drive as a member of a real optical library or pseudo optical library.

dual copy

A high availability function made possible by nonvolatile storage in some models of the IBM 3990 Storage Control. Dual copy maintains two functionally identical copies of designated DASD volumes in the logical 3990 subsystem, and automatically updates both copies every time a write operation is issued to the dual copy logical volume.

dummy storage group

A type of storage group that contains the serial numbers of volumes no longer connected to a system. Dummy storage groups allow existing JCL to function without having to be changed. See also *storage group*.

dump class

A set of characteristics that describes how volume dumps are managed by DFSMShsm.

duplexing

The process of writing two sets of identical records in order to create a second copy of data.

dynamic cache management

A function that automatically determines which data sets will be cached based on the 3990 subsystem load, the characteristics of the data set, and the performance requirements defined by the storage administrator. Enhanced Capacity Cartridge System Tape. Cartridge system tape with increased capacity that can only be used with 3490E Magnetic Tape Subsystems. Contrast with *Cartridge System Tape*.

error recovery

A procedure for copying, storing, and recovering data essential to an installation's business in a secure location, and for recovering that data in the event of an error at installation. Contrast with *vital records*.

esoteric unit name

A name used to define a group of devices having similar hardware characteristics, such as TAPE or SYSDA. Contrast with *generic unit name*.

expiration

(1) The process by which data sets or objects are identified for deletion because their expiration date or retention period has passed. On DASD, data sets and objects are deleted. On tape, when all data sets have reached their expiration date, the tape volume is available for reuse. (2) In DFSMSrmm, all volumes have an expiration date or retention period set for them either by vital record specification policy, by user-specified JCL when writing a data set to the volume, or by an installation

default

When a volume reaches its expiration date or retention period, it becomes eligible for release.

extended format

The format of a data set that has a data set name type (DSNTYPE) of EXTENDED. The data set is structured logically the same as a data set that is not in extended format but the physical format is different. Data sets in extended format can be striped or compressed. Data in an extended format VSAM KSDS can be compressed. See also *striped data set*, *compressed format*.

filtering

The process of selecting data sets based on specified criteria. These criteria consist of fully or partially-qualified data set names or of certain data set characteristics.

generic unit name

A name assigned to a class of devices with the same geometry (such as 3390). Contrast with *esoteric unit name*.

global access checking

In RACF, the ability to establish an in-storage table of default values containing authorization levels for selected resources. RACF refers to this table prior to performing its usual RACHECK processing, and grants the request without performing a RACHECK if the requested access authority does not exceed the global value. Global access checking can grant a user access to the resource, but it cannot deny access.

global resource serialization (GRS)

A component of z/OS used for serializing use of system resources and for converting hardware reserves on DASD volumes to data set enqueues.

group

(1) With respect to partitioned data sets, a member and the member's aliases that exist in a PDS or PDSE, or in an unloaded PDSE. (2) A collection of users who can share access authorities for protected resources.

Hardware Configuration Definition (HCD)

An interactive interface in MVS/ESA SP that enables an installation to define hardware configurations from a single point of control.

implementation by milestone

A conversion approach that allows for a staged conversion of your installation's data to system-managed storage on DASD, tape, or optical devices.

large format

The format of a data set that has a data set name type (DSNTYPE) of LARGE. A large format data set has the same characteristics as a sequential (non-extended format) data set, but its size on each volume can exceed 65 535 tracks. There is no minimum size requirement for a large format data set.

management class

A collection of management attributes, defined by the storage administrator, used to control the release of allocated but unused space; to control the retention, migration, and backup of data sets; to control the retention and backup of aggregate groups, and to control the retention, backup, and class transition of objects.

manual tape library

Installation-defined set of tape drives defined as a logical unit together with the set of system-managed volumes which can be mounted on the drives.

migration

The process of moving unused data to lower cost storage in order to make space for high-availability data. If you wish to use the data set, it must be recalled. See also *migration level 1*, *migration level 2*.

migration level 1

DFSMSHsm-owned DASD volumes that contain data sets migrated from primary storage volumes. The data can be compressed. See also *storage hierarchy*. Contrast with *primary storage*, *migration level 2*.

migration level 2

DFSMSHsm-owned tape or DASD volumes that contain data sets migrated from primary storage volumes or from migration level 1 volumes. The data can be compressed. See also *storage hierarchy*. Contrast with *primary storage*, *migration level 1*.

nondisruptive installation

A capability that allows users to access data in an existing storage subsystem while installing additional devices in the subsystem.

nonvolatile storage (NVS)

Additional random access electronic storage with a backup battery power source, available with an IBM Cache Storage Control, used to retain data during a power outage. Nonvolatile storage, accessible from all storage directors, stores data during DA SD fast write and dual copy operations.

OAM-managed volumes

Optical or tape volumes controlled by the object access method (OAM).

object

A named byte stream having no specific format or record orientation.

object access method (OAM)

An access method that provides storage, retrieval, and storage hierarchy management for objects and provides storage and retrieval management for tape volumes contained in system-managed libraries.

object backup storage group

A type of storage group that contains optical or tape volumes used for backup copies of objects. See also *storage group*.

object directory tables

A collection of DB2 tables that contain information about the objects that have been stored in an object storage group.

object storage group

A type of storage group that contains objects on DASD, tape, or optical volumes. See also *storage group*.

object storage hierarchy

A hierarchy consisting of objects stored in DB2 table spaces on DASD, on optical or tape volumes that reside in a library, and on optical or tape volumes that reside on a shelf. See also *storage hierarchy*.

object storage tables

A collection of DB2 tables that contain objects.

optical disk drive

The mechanism used to seek, read, and write data on an optical disk. An optical disk drive can be operator-accessible or library-resident.

optical library

A storage device that houses optical drives and optical cartridges, and contains a mechanism for moving optical disks between a cartridge storage area and optical disk drives.

optical volume

Storage space on an optical disk, identified by a volume label. See also *volume*.

partitioned data set (PDS)

A data set on direct access storage that is divided into partitions, called members, each of which can contain a program, part of a program, or data.

partitioned data set extended (PDSE)

A data set that contains an indexed directory and members that are similar to the directory and members of partitioned data sets. A PDSE can be used instead of a partitioned data set.

partitioned data set unloaded (PDSU)

An IEBCOPY unload data set. A sequential file that can be restored by IEBCOPY to create a PDS.

performance

- (1) A measurement of the amount of work a product can produce with a given amount of resources.
- (2) In a system-managed storage environment, a measurement of effective data processing speed with respect to objectives set by the storage administrator. Performance is largely determined by throughput, response time, and system availability.

permanent data set

A user-named data set that is normally retained for longer than the duration of a job or interactive session. Contrast with *temporary data set*.

pool storage group

A type of storage group that contains system-managed DASD volumes. Pool storage groups allow groups of volumes to be managed as a single entity. See also *storage group*.

primary space allocation

Amount of space requested by a user for a data set when it is created. Contrast with *secondary space allocation*.

primary storage

A DASD volume available to users for data allocation. The volumes in primary storage are called primary volumes. See also *storage hierarchy*. Contrast with migration level 1, migration level 2.

program object

All or part of a computer program in a form suitable for loading into virtual storage for execution. Program objects are stored in PDSE program libraries and have fewer restrictions than load modules. Program objects are produced by the binder.

pseudo optical library

A set of shelf-resident optical volumes associated with an operator-accessible optical disk drive; see also *real optical library*.

real optical library

Physical storage device that houses optical disk drives and optical cartridges, and contains a mechanism for moving optical disks between a cartridge storage area and optical disk drives. Contrast with *pseudo optical library*.

recovery

The process of rebuilding data after it has been damaged or destroyed, often by using a backup copy of the data or by reapplying transactions recorded in a log.

removable media library

The volumes that are available for immediate use, and the shelves where they could reside. Resource Access Control Facility (RACF). An IBM licensed program that is included in z/OS Security Server and is also available as a separate program for the z/OS and VM environments. RACF provides access control by identifying and verifying the users to the system, authorizing access to protected resources, logging detected unauthorized attempts to enter the system, and logging detected accesses to protected resources.

Resource Measurement Facility (RMF)

An IBM licensed program or optional element of z/OS, that measures selected areas of system activity and presents the data collected in the format of printed reports, system management facilities (SMF) records, or display reports. Use RMF to evaluate system performance and identify reasons for performance problems.

secondary space allocation

Amount of additional space requested by the user for a data set when primary space is full. Contrast with *primary space allocation*.

sequential data striping

A software implementation of a disk array that distributes data sets across multiple volumes to improve performance.

service-level agreement

(1) An agreement between the storage administration group and a user group defining what service-levels the former will provide to ensure that users receive the space, availability, performance, and security they need. (2) An agreement between the storage administration group and operations defining what service-level operations will provide to ensure that storage management jobs required by the storage administration group are completed.

Service Level Reporter (SLR)

An IBM licensed program that provides the user with a coordinated set of tools and techniques and consistent information to help manage the data processing installation. For example, SLR extracts information from SMF, IMS, and CICS logs, formats selected information into tabular or graphic reports, and gives assistance in maintaining database tables.

shelf

A place for storing removable media, such as tape and optical volumes, when they are not being written to or read.

shelf location

(1) A single space on a shelf for storage of removable media. (2) In DFSMSrmm, a shelf location is defined in the removable media library by a rack number, and in a storage location, it is defined by a bin number. See also *rack number*, *bin number*.

SMS configuration

A configuration base, Storage Management Subsystem class, group, library, and drive definitions, and ACS routines that the Storage Management Subsystem uses to manage storage. See also *configuration*, *base configuration*, *source control data set*.

source control data set (SCDS)

A VSAM linear data set containing an SMS configuration. The SMS configuration in an SCDS can be changed and validated using ISMF. See also *active control data set*, *communications data set*.

storage administration group

A centralized group within the data processing center that is responsible for managing the storage resources within an installation.

storage administrator

A person in the data processing center who is responsible for defining, implementing, and maintaining storage management policies.

storage class

A collection of storage attributes that identify performance goals and availability requirements, defined by the storage administrator, used to select a device that can meet those goals and requirements.

storage control

The component in a storage subsystem that handles interaction between processor channel and storage devices, runs channel commands, and controls storage devices.

storage director

In a 3990 Storage Control, a logical entity consisting of one or more physical storage paths in the same storage cluster. In a 3880, a storage director is equivalent to a storage path.

storage group

A collection of storage volumes and attributes, defined by the storage administrator. The collections can be a group of DASD volumes or tape volumes, or a group of DASD, optical, or tape volumes treated as a single object storage hierarchy. See also *VIO storage group*, *pool storage group*, *tape storage group*, *object storage group*, *object backup storage group*, *dummy storage group*.

storage group category

A grouping of specific storage groups which contain the same type of data. This concept is analogous to storage pools in a non-system-managed environment.

storage hierarchy

An arrangement of storage devices with different speeds and capacities. The levels of the storage hierarchy include main storage (memory, DASD cache), primary storage (DASD containing uncompressed data), migration level 1 (DASD containing data in a space-saving format), and migration level 2 (tape cartridges containing data in a space-saving format). See also *primary storage*, *migration level 1*, *migration level 2*, *object storage hierarchy*.

storage location

A location physically separate from the removable media library where volumes are stored for disaster recovery, backup, and vital records management.

storage management

The activities of data set allocation, placement, monitoring, migration, backup, recall, recovery, and deletion. These can be done either manually or by using automated processes. The Storage Management Subsystem automates these processes for you, while optimizing storage resources. See also *Storage Management Subsystem*.

Storage Management Subsystem (SMS)

A DFSMS facility used to automate and centralize the management of storage. Using SMS, a storage administrator describes data allocation characteristics, performance and availability goals, backup and retention requirements, and storage requirements to the system through data class, storage class, management class, storage group, and ACS routine definitions.

storage subsystem

A storage control and its attached storage devices. See also *tape subsystem*.

stripe

In DFSMS, the portion of a striped data set, such as an extended format data set, that resides on one volume. The records in that portion are not always logically consecutive. The system distributes records among the stripes such that the volumes can be read from or written to simultaneously to gain better performance. Whether it is striped is not apparent to the application program.

striped data set

In DFSMS, an extended-format data set consisting of two or more stripes. SMS determines the number of stripes to use based on the value of the SUSTAINED DATA RATE in the storage class. Striped data sets can take advantage of the sequential data striping access technique. See *stripe*, *striping*.

system data

The data sets required by z/OS or its subsystems for initialization and control.

system-managed data set

A data set that has been assigned a storage class.

system-managed storage

Storage managed by the Storage Management Subsystem. SMS attempts to deliver required services for availability, performance, and space to applications. See also *system-managed storage environment*.

system-managed tape library

A collection of tape volumes and tape devices, defined in the tape configuration database. A system-managed tape library can be automated or manual. See also *tape library*.

system-managed volume

A DASD, optical, or tape volume that belongs to a storage group. Contrast with *DFSMSHsm-managed volume*, *DFSMSrmm-managed volume*.

system management facilities (SMF)

A component of z/OS that collects input/output (I/O) statistics, provided at the data set and storage class levels, which helps you monitor the performance of the direct access storage subsystem.

system programmer

A programmer who plans, generates, maintains, extends, and controls the use of an operating system and applications with the aim of improving overall productivity of an installation.

tape configuration database

One or more volume catalogs used to maintain records of system-managed tape libraries and tape volumes.

tape librarian

The person who manages the tape library. This person is a specialized storage administrator.

tape library

A set of equipment and facilities that support an installation's tape environment. This can include tape storage racks, a set of tape drives, and a set of related tape volumes mounted on those drives. See also *system-managed tape library*, *automated tape library data server*.

Tape Library Dataserver

A hardware device that maintains the tape inventory that is associated with a set of tape drives. An automated tape library dataserver also manages the mounting, removal, and storage of tapes.

tape mount management

The methodology used to optimize tape subsystem operation and use, consisting of hardware and software facilities used to manage tape data efficiently.

tape storage group

A type of storage group that contains system-managed private tape volumes. The tape storage group definition specifies the system-managed tape libraries that can contain tape volumes. See also *storage group*.

tape subsystem

A magnetic tape subsystem consisting of a controller and devices, which allows for the storage of user data on tape cartridges. Examples of tape subsystems include the IBM 3490 and 3490E Magnetic Tape Subsystems.

tape volume

A tape volume is the recording space on a single tape cartridge or reel. See also *volume*.

temporary data set

An uncataloged data set whose name begins with & or &&, that is normally used only for the duration of a job or interactive session. Contrast with *permanent data set*.

unmovable data set

A DASD data set required by an application, which cannot be moved. Unmovable data sets need to be identified to the system so that they are not relocated. They can be identified by allocation the data set in absolute tracks or by allocating a data set with the data set organization that includes the unmovable attribute. For example, data sets allocated as PSU, POU, DAU, or ABSTR are considered unmovable.

use attribute

(1) The attribute assigned to a DAD volume that controls when the volume can be used to allocate new data sets; use attributes are public, private, and storage. (2) For system-managed tape volumes, use attributes are scratch and private.

user group

A group of users in an installation who represent a single department or function within the organization.

user group representative

A person within a user group who is responsible for representing the user group's interests in negotiations with the storage administration group.

validate

To check the completeness and consistency of an individual ACS routine or an entire SMS configuration.

virtual input/output (VIO) storage group

A type of storage group that allocates data sets to paging storage, which simulates a DASD volume. VIO storage groups do not contain any actual DASD volumes. See also *storage group*.

vital records

A data set or volume maintained for meeting an externally-imposed retention requirement, such as a legal requirement. Compare with disaster recovery.

vital record specification

Policies defined to manage the retention and movement of data sets and volumes for disaster recovery and vital records purposes.

volume

The storage space on DASD, tape, or optical devices, which is identified by a volume label. See also *DASD volume*, *optical volume*, *tape volume*.

volume mount analyzer

A program that helps you analyze your current tape environment. With tape mount management, you can identify data sets that can be redirected to the DASD buffer for management using SMS facilities.

volume status

In the Storage Management Subsystem, indicates whether the volume is fully available for system management:

- "Initial" indicates that the volume is not ready for system management because it contains data sets that are ineligible for system management.
- "Converted" indicates that all of the data sets on a volume have an associated storage class and are cataloged.
- "Non-system-managed" indicates that the volume does not contain any system-managed data sets and has not been initialized as system-managed.

VSAM volume data set (VVDS)

A data set that describes the characteristics of VSAM and system-managed data sets residing on a given DASD volume; part of a catalog. See also *basic catalog structure*.

Index

Numerics

- 3800 Printing Subsystem Model 1
 - FCB module structure [152](#)
 - GRAPHIC module structure [161](#)
 - module types [149](#)
- 3800 Printing Subsystem Model 3
 - creating graphic character set modules [179](#)
 - creating library character set modules [179](#)
 - GRAPHIC module structure [161](#)
- 4248 FCB module structure [153](#)
- 4248 printer module [149](#)
- 4248 printer, creating FCB module [179](#)

A

- abbreviations
 - exceptions [60](#)
 - key words [60](#)
- ABEND codes, IEBCOPY program [329](#)
- ACCESS parameter
 - INITT statement [250](#)
- accessibility
 - contact IBM [361](#)
- ADD statement
 - IEBUPDTE program [224](#), [228](#)
 - IEHPROGM program [304](#)
- alias name
 - changing member [295](#)
 - copying member [45](#)
 - creating for partitioned member [231](#)
- ALIAS statement, IEBUPDTE program [231](#)
- allocation/deallocation [300](#)
- ALTERMOD statement
 - IEBCOPY program [50](#), [60](#)
 - restrictions [50](#)
- anyname DD statement
 - IEHINITT program [247](#), [248](#)
 - IEHLIST program [261](#)
 - IEHMOVE program [279](#)
 - IEHPROGM program [299](#), [300](#)
- anyname1 DD statement, IEBCOPY program [55](#), [58](#)
- anyname2 DD statement, IEBCOPY program [55](#)
- APF (Authorized Program Facility), invoking utility program [315](#)
- assistive technologies [361](#)
- attribute records, IEBCOPY unload data set [339](#)

B

- backup
 - data set [127](#)
 - example [75](#)
 - IEBCOPY program [43](#)
 - member [127](#)
 - verify with IEBCOMPR program [31](#)
- basic direct access method [274](#)

- BDAM
 - macro [274](#)
- BDAM (basic direct access method)
 - copying [274](#)
 - moving [274](#)
 - renaming [295](#)
 - scratching [295](#)
- block size
 - COPYMOD statement [65](#)
 - unload data set
 - MAXBLK [43](#)
 - MAXBLK, MINBLK [58](#)
 - MINBLK [43](#)
- buffer
 - IEHMOVE program [277](#)
 - work area [51](#)
- building an index with IEHPROGM [295](#)

C

- cataloged data set, moving
 - example [294](#)
- CATLG statement, IEHPROGM program [303](#)
- CHANGE statement, IEBUPDTE program [224](#), [228](#)
- character arrangement table
 - 3800 modules [149](#)
 - building example [187](#), [189](#)
 - creating [157](#), [172](#)
 - deleting graphic reference example [189](#)
 - listing [159](#)
 - modifying example [187](#), [189](#)
 - structure [158](#)
- CHARSET statement
 - IEBIMAGE program syntax [175](#), [177](#)
 - module [163](#)
 - statement [175](#), [177](#)
- CLOSE macro
 - totaling routine [347](#)
 - user label [347](#)
- COMPARE statement
 - IEBCOMPR program [33](#)
 - syntax [33](#)
- comparing, IEBCOMPR program
 - copying partitioned data set example [38](#)
 - copying sequential data set example [37](#)
- data sets
 - card input [37](#)
 - partitioned [31](#), [38](#), [39](#)
 - PDSE [31](#), [39](#)
 - sequential [31](#), [35](#), [38](#)
 - tape input [37](#)
- different density sequential data sets example [36](#)
- partitioned data set example [38](#)
- PDSE example [39](#)
- sequential data set example [37](#)
- tape resident data set example [36](#)
- COMPRESS parameter, IEBCOPY program [56](#)

- compress-in-place operation
 - data set [49](#)
 - processing considerations [49](#)
- contact
 - z/OS [361](#)
- control statements [6](#)
- converting
 - H-set BCDIC to EBCDIC [141](#)
 - load modules to program objects [41](#)
 - packed decimal to unpacked decimal [141](#)
 - partitioned data set to PDSE [42](#), [90](#)
 - program objects to load modules [41](#)
 - sequential to partitioned data set or PDSE [127](#)
 - unpacked decimal to packed decimal [141](#)
- COPY DSGROUP statement, IEHMOVE program [283](#), [284](#)
- COPY DSNAM statement, IEHMOVE program [281](#), [282](#)
- COPY PDS statement, IEHMOVE program [280](#), [284](#), [286](#)
- COPY statement, IEBCOPY program [49](#), [60](#), [62](#)
- COPY VOLUME statement
 - IEHMOVE program [287](#)
- COPY VOLUME statement, IEHMOVE progra [286](#)
- COPYAUTH statement
 - moving and copying [270](#)
- COPYGROUP statement
 - IEBCOPY program [60](#), [64](#)
 - syntax [64](#)
- COPYGRP member, replacing [48](#)
- COPYGRP statement
 - IEBCOPY program [60](#), [63](#)
 - syntax [63](#)
- copying
 - basic
 - sequential data set [271](#)
 - BDAM data set [274](#)
 - BDAM macro [274](#)
 - cataloged data sets [275](#)
 - COPYAUTH statement [270](#)
 - data set with delimiter example [124](#)
 - data sets [270](#), [276](#)
 - DBCS
 - example [147](#)
 - DBCS data
 - IEBGENER program [130](#)
 - directory information [43](#)
 - edited statements example [123](#)
 - edited steps example [123](#)
 - examples [122](#), [125](#)
 - IEBCOPY program
 - COPY statement [60](#), [62](#)
 - COPYGROUP statement [60](#), [64](#)
 - COPYGRP statement [60](#), [63](#)
 - COPYMOD statement [60](#), [65](#)
 - data set [42](#)
 - load modules [50](#)
 - members with aliases [45](#)
 - unload data set [43](#)
 - IEBUPDTE program example [241](#)
 - IEHMOVE program [280](#), [281](#)
 - job statement
 - example [122](#)
 - job statement example [122](#), [123](#), [125](#), [202](#)
 - job step example [122](#), [123](#), [125](#), [202](#)
 - load modules [50](#)
 - member
 - copying (*continued*)
 - member (*continued*)
 - excluding [45](#)
 - with an alias [45](#)
 - modification module
 - creating [156](#), [170](#)
 - example, building [186](#), [187](#)
 - IEBIMAGE listing [180](#)
 - overrun notes [180](#)
 - printing data, specifying [149](#)
 - multiple jobs example [122](#)
 - multiple operations examples [78](#), [86](#)
 - multivolume [274](#)
 - optional
 - sequential data set [271](#)
 - partitioned data set
 - example [71](#), [75](#)
 - IEBCOPY program [41](#), [90](#)
 - partitioned data set example [38](#)
 - partitioned data set extended
 - IEBCOPY program [92](#), [94](#)
 - partitioned data set extended, IEBCOPY program [91](#), [92](#)
 - partitioned data set, IEBCOPY program [42](#)
 - PDSE
 - example [71](#)
 - IEBCOPY program [42](#)
 - RACF [270](#)
 - reblocking [270](#)
 - select members example [73](#)
 - sequential data set
 - example [124](#), [125](#), [144](#), [145](#), [147](#)
 - sequential data set example [37](#)
 - SMS-managed volume [267](#)
 - unloaded [274](#)
 - unloaded data set [43](#)
 - unmovable [275](#)
 - unsuccessful space allocated [269](#)
 - user label [127](#)
 - volume [276](#)
 - COPYMOD statement
 - description [50](#)
 - example [186](#)
 - IEBCOPY program [60](#), [65](#), [66](#)
 - IEBIMAGE listing with overrun notes [172](#)
 - IEBIMAGE program [170](#), [172](#)
 - module [156](#)
 - syntax [65](#), [170](#)
 - CREATE parameter, user exit routines [343](#)
 - CREATE statement [105](#)
 - CREDSNAM program
 - description [13](#)

D

- DATA parameter, user exit routines [343](#)
- data set
 - comparing [31](#)
 - copying
 - BDAM [270](#), [274](#)
 - cataloged group [275](#)
 - description [267](#)
 - IEBCOPY program [41](#)
 - IEBGENER program [127](#)
 - IEHMOVE program [270](#), [281](#)

- data set (*continued*)
 - copying (*continued*)
 - multivolume [274](#)
 - partitioned [270](#)
 - sequential [271](#)
 - unloaded [274](#)
 - unmovable attribute [275](#)
 - volume [276](#)
 - creating new master example [236](#)
 - edited [129](#)
 - example
 - scratching [307](#)
 - uncataloging [307](#)
 - I/O [119](#)
 - IEBUPDTE program library [221](#)
 - library member example, adding records [238](#)
 - logical record length [130](#)
 - maintaining a password [296](#)
 - member
 - aliases [295](#)
 - renaming [47](#)
 - replacing [46](#)
 - merging [42](#)
 - moving
 - BDAM [270](#), [274](#)
 - cataloged group [275](#)
 - description [267](#)
 - IEHMOVE program [270](#), [281](#)
 - multivolume [274](#)
 - partitioned [270](#)
 - sequential [271](#)
 - unloaded [274](#)
 - unmovable attribute [275](#)
 - volume [276](#)
 - new partitioned, create example [241](#)
 - organization, changing [221](#)
 - printing [203](#)
 - punching [203](#)
 - reblocking [270](#)
 - renaming example [307](#)
 - scratching [307](#)
 - scratching example [307](#)
 - sequential example, card input [240](#)
 - space allocation
 - IEHMOVE program [269](#)
 - SYS1.IMAGELIB [149](#)
 - uncataloging [307](#)
 - uncataloging example [308](#)
 - unload
 - copying, IEBCOPY program [43](#)
 - DCB parameter, IEBCOPY program [43](#)
 - loading, IEBCOPY program [43](#)
 - utility programs summary [5](#)
- data statements
 - IEBUPDTE program [230](#)
 - user-designed characters [175](#), [177](#)
- DATATYPE parameter, SYSIN [7](#)
- DATE field in formatted VTOC listing [357](#)
- DBCS (double-byte character set)
 - copying [130](#)
 - editing [130](#)
 - example, editing [147](#)
 - printing [130](#), [204](#)
 - printing example [220](#)
- DBCS (double-byte character set) (*continued*)
 - punching [204](#)
 - reblocking [130](#)
 - SO/SI
 - characters [130](#)
 - deleting [140](#)
 - inserting [140](#)
 - validating [130](#)
- DCB parameter
 - IEBCOPY program, unloading data se [58](#)
 - IEBCOPY program, unloading data set [43](#)
 - OPTCD=W [58](#)
- DD JCL statement
 - IEBCOMPR program [32](#)
 - IEBCOPY program [57](#), [59](#)
 - IEBDG program [99](#)
 - IEBEDIT program [120](#)
 - IEBGENER program [133](#)
 - IEBIMAGE program [165](#), [166](#)
 - IEBPDSE program [202](#)
 - IEBPTPCH program [205](#)
 - IEBUPDTE program [223](#), [224](#)
 - IEHINITT program [247](#), [248](#)
 - IEHLIST program [261](#), [262](#)
 - IEHMOVE program [279](#), [280](#)
 - IEHPROGM program [300](#)
 - IFHSTATR program [313](#)
- DDM attributes, copying [42](#)
- ddname statement
 - INITT statement [249](#)
- debugging tool, IEBDG utility [95](#)
- DELETEP statement, IEHPROGM program
 - delete password [305](#)
 - syntax [305](#)
- density, comparing sequential data sets example [36](#)
- detail statement
 - IEBUPDTE program [228](#), [230](#)
 - restrictions [230](#)
- DETAIL statement, IEBUPDTE program [228](#), [230](#)
- determining the IEBCOPY operation
 - COPY operation [60](#)
 - load operation
 - partitioned output data set [60](#)
 - sequential input data set [60](#)
 - partitioned data sets
 - input [60](#)
 - output [60](#)
 - unload operation
 - partitioned input data set [60](#)
 - sequential output data set [60](#)
- device variable [xxii](#)
- DFSMSdss (Data Facility Data Set Services) [276](#)
- DFSMSrmm [246](#)
- DFSORT (Data Facility Sort) [127](#)
- diagnosis
 - CVOL [257](#)
- directory block allocation
 - merging data sets [42](#)
- directory list
 - edited format [257](#)
- directory records, IEBCOPY unload data set [339](#)
- DISP parameter
 - INITT statement [249](#)
- DSD statement, IEBDG program [100](#)

E

- EDGINERS program [246](#)
- EDIT statement syntax, IEBEDIT program [120](#)
- edited data set
 - creating [129](#)
 - printing [203](#)
 - punching [203](#)
- editing
 - DBCS [130](#)
 - DBCS example [147](#)
 - sequential data set example [145](#), [147](#)
- END statement, IEBDG program [110](#)
- ENDUP statement, IEBUPDTE program [232](#)
- entry routines, utility programs [344](#)
- EOV (end-of-volume)
 - CLOSE macro [347](#)
 - user label [347](#)
- ERROR parameter, user exit routines [343](#)
- ESV (error statistics by volume) data, IFHSTATR program
 - example, printing [313](#)
 - printing [311](#)
 - printout [311](#)
 - sorting [311](#)
- EXCLUDE statement
 - COPY statement
 - DSGROUP [280](#)
 - PDS [280](#)
 - IEBCOPY program [60](#), [66](#), [68](#)
 - IEHMOVE program [288](#)
 - MOVE statement
 - DSGROUP [280](#)
 - PDS [280](#)
- exclusive copying, IEBCOPY program [46](#)
- EXEC JCL statement
 - IEBCOPY program [55](#)
 - IEBEDIT program [119](#)
 - IEBPDSE program [201](#)
- EXEC statement
 - IEBGGENER program [131](#), [132](#)
 - IEBIMAGE program [165](#)
 - IEBPTPCH program [205](#)
 - IEBUPDTE program [222](#)
 - IEHINITT program [247](#)
 - IEHLIST program
 - syntax [261](#)
 - IEHMOVE program
 - syntax [277](#)
 - IEHPROGM program [299](#)
 - IFHSTATR program [313](#)
- exit routine
 - identifying [136](#)
 - IEBCOMPR program [34](#)
 - IEBDG program [110](#)
 - IEBPTPCH program [210](#)
 - IEBUPDTE program [222](#), [226](#)
 - nonlabel processing routine [346](#)
 - return codes [345](#)
 - RETURN macro [344](#)
 - totaling [348](#)
 - utility programs [343](#)
- EXITS statement
 - IEBCOMPR program syntax [34](#)
 - IEBGGENER program syntax [136](#)

EXITS statement (*continued*)

- IEBPTPCH program
 - exit routines [210](#)
 - syntax [210](#)

F

- FCB (forms control buffer)
 - 3800 printer module [149](#)
 - 4248 printer module [149](#)
 - character arrangement table module
 - example, building [187](#), [189](#)
 - example, deleting graphic reference [189](#)
 - example, modifying [187](#), [189](#)
 - graphic character modification module
 - example, defining [193](#)
 - example, printing [190](#), [193](#)
 - example, using [193](#)
- IEBIMAGE program syntax [167](#)
- library character set module
 - example, building [195](#)
 - example, building from multiple sources [196](#)
 - example, listing [194](#)
 - example, modifying [195](#)
- module
 - 3800 FCB module [152](#), [167](#)
 - 4248 FCB module [153](#), [167](#)
 - creating [167](#)
 - example, building [182](#), [187](#)
 - example, replacing [182](#), [184](#)
 - IEBIMAGE listing [155](#)
 - IEBIMAGE program [167](#)
 - printer information [152](#)
 - structure [152](#), [153](#)
- printers [151](#)
- statement, IEBIMAGE program [167](#), [170](#)
- FD statement, IEBDG program [100](#), [105](#)
- formatting, IFHSTATR program, type 21 records [311](#)
- full copy, IEBCOPY program [46](#)
- FUNCTION statement
 - IEBUPDTE program
 - begin operation [224](#)
 - example [241](#)
 - syntax [224](#)
 - restrictions [227](#)

G

- GDKUTIL program
 - description [11](#), [12](#), [14–16](#), [26–30](#)
- GENERATE statement, IEBGENER program [136](#)
- glossary [367](#)
- graphic character modification module
 - building [187](#), [194](#)
 - creating [160](#), [173](#)
 - example, building [191](#), [192](#)
 - example, printing [190](#)
 - IEBIMAGE program listing [161](#)
 - listing [187](#), [194](#)
 - modifying character arrangement table [191](#)
 - multiple sources [192](#)
 - structure [161](#)
 - World Trade GRAFMOD [190](#)

graphic character modification module (*continued*)
 World Trade National Use Graphics [190](#)
GRAPHIC module, IEBIMAGE program [160](#)
GRAPHIC statement syntax, IEBIMAGE program [173](#), [175](#)
guide to utility program functions [1](#)

H

HDNGLST page header parameter syntax [328](#)
hexadecimal output printing example [219](#)

I

IBM TotalStorage Enterprise Automated Tape Library (3495)
[246](#)

ICEGENER program [127](#)

ICF (integrated catalog facility) [308](#)

IEBCOMPR program

- COMPARE statement [33](#)
- comparing data sets [31](#)
- description [31](#)
- examples [35](#), [39](#)
- EXEC statement [32](#)
- EXITS statement [34](#)
- input [32](#)
- job control statement [32](#), [33](#)
- LABELS statement [34](#)
- output [32](#)
- return codes [328](#)
- sequential data set example [35](#)
- SYSIN DD statement [32](#)
- SYSPRINT DD statement [32](#)
- SYSUT1 DD statement [32](#)
- SYSUT2 DD statement [32](#)
- user exit routines [343](#)
- utility control statement [33](#), [35](#)
- verifying backup copy [31](#)

IEBCOPY program

- ABEND codes [329](#)
- altering load modules in place [50](#)
- ALTERMOD statement [60](#), [61](#)
- anyname1 DD statement [58](#)
- anyname2 DD statement [58](#)
- buffer size [51](#)
- compressing data set, processing considerations [49](#)
- converting
 - load modules to program objects [41](#)
 - partitioned data set to PDSE [42](#)
 - program objects to load modules [41](#)
- COPY statement [60](#), [62](#)
- COPYGROUP statement [60](#), [64](#)
- COPYGRP statement [60](#), [63](#)
- copying
 - data set [42](#)
 - DDM attributes [42](#)
 - directory information [43](#)
 - load modules [50](#)
 - members with aliases [45](#)
 - program objects, COPYGRP [47](#)
 - unload data sets [43](#)
- COPYMOD example [89](#)
- COPYMOD statement [60](#), [65](#)
- data set

IEBCOPY program (*continued*)

data set (*continued*)

- backing up [43](#)
- copying [42](#)
- merging [42](#)
- unloading [43](#)
- description [41](#)
- directory information
 - copying [43](#)
- examples [69](#)
- EXCLUDE statement [44](#), [60](#), [66](#)
- excluding members [45](#)
- EXEC JCL statement [55](#)
- INDD statement [60](#)
- input [54](#)
- inserting RLD counts [50](#)
- invoking from application program
 - example [326](#)
- invoking from application program example [327](#)
- job control statement [54](#)
- JOB statement [55](#)
- load operation [41](#)
- load processing [46](#)
- loading unload data sets [43](#)
- logical record length [43](#)
- long names [48](#)
- MEMBER parameter [47](#), [48](#)
- merging data set [42](#)
- output [54](#)
- reblocking load modules [50](#)
- renaming selected members [47](#)
- replacing data set members [46](#)
- restrictions [52](#)
- return codes [328](#)
- SELECT statement
 - description [67](#)
 - renaming members [47](#)
 - replacing aliases [48](#)
 - selecting members [44](#), [60](#)
- selecting members to be copied [44](#)
- selecting members to be loaded or unloaded [44](#)
- selective copy [46](#)
- SYSIN DD statement [59](#)
- SYSMDUMP DD statement [57](#)
- SYSPRINT DD statement [57](#)
- SYSUT1 DD statement [58](#)
- SYSUT2 DD statement [58](#)
- SYSUT3 statement [59](#)
- SYSUT4 statement [59](#)
- table size [51](#)
- unload data set
 - attribute records [339](#)
 - DCB parameters [43](#), [58](#)
 - directory records [339](#)
 - format [335](#)
 - member data records [341](#)
 - note list records [340](#)
 - rules and restrictions [336](#)
- unloading data set [43](#)
- user ABEND codes [329](#)
- utility control statement [59](#)
- virtual storage [51](#)
- work area size [51](#)

IEBDG program

IEBDG program (*continued*)

- actions [97, 103](#)
- anyname1 DD statement [99](#)
- anyname2 DD statement [99](#)
- CREATE statement [349](#)
- defining record fields [95](#)
- description [95](#)
- DSD statement [100](#)
- END statement [110](#)
- examples [110, 118](#)
- EXEC statement [98](#)
- exits [349](#)
- FD statement [100](#)
- IBM supplied patterns [95](#)
- input [98](#)
- job control statement [98, 99](#)
- modifying record fields [97](#)
- output [98](#)
- return codes [329](#)
- SYSPRINT DD statement [99](#)
- user exit routines [343](#)
- user-specified patterns [96](#)
- utility control statement [100, 110](#)

IEBEDIT program

- description [119](#)
- examples [121, 125](#)
- EXEC JCL statement [119](#)
- input [119](#)
- job control statement [119, 120](#)
- JOB statement [119](#)
- output [119](#)
- return codes [330](#)
- SYSIN DD statement [120](#)
- SYSPRINT DD statement [120](#)
- SYSUT1 DD statement [120](#)
- SYSUT2 DD statement [120](#)
- utility control statement [120, 121](#)

IEBGENER program

- changing logical record length [130](#)
- converting
 - H-set BCDIC to EBCDIC [141](#)
 - packed decimal to unpacked decimal [141](#)
 - unpacked decimal to packed decimal [141](#)
- copying, no directory entry processing [127](#)
- creating
 - edited data set [129](#)
 - partitioned data sets or PDSEs [127](#)
- DBCS
 - data [130](#)
 - example [147](#)
- DD JCL statement [133](#)
- deleting SO/SI [140](#)
- description [127](#)
- examples [141, 147](#)
- EXEC statement [131](#)
- input [131](#)
- inserting SO/SI [140](#)
- invoking from application program example [326](#)
- job control statement [131, 135](#)
- JOB statement [131](#)
- output [131](#)
- partitioned data set
 - adding members [128](#)
 - example [142, 144](#)

IEBGENER program (*continued*)

- PDSE [128](#)
- reblocking example [144](#)
- return codes [330](#)
- sequential data set example [144, 147](#)
- SYSIN DD statement [132](#)
- SYSPRINT DD statement [132, 133](#)
- SYSUT1 DD statement [132](#)
- SYSUT2 DD statement [132](#)
- user exit routines [343](#)
- utility control statement
 - EXITS [136](#)
 - GENERATE statement [136](#)
 - LABELS statement [137](#)
 - member [138](#)
 - RECORD statement [139](#)

IEBIMAGE program

- 3800 FCB module structure [152](#)
- 4248 FCB module structure [153](#)
- character arrangement table module [157](#)
- CHARSET module listing [163](#)
- CHARSET module structure
 - 3800 Model 1 and Model 3 [163](#)
 - description [163](#)
- CHARSET statement [175, 177](#)
- COPYMOD module
 - structure and listing [156](#)
- COPYMOD module structure and listing [156](#)
- COPYMOD statement [170, 172](#)
- creating
 - character arrangement table module [157](#)
 - copying modification module [156](#)
 - FCB module [152](#)
 - graphic character modification module [160](#)
 - library character set module [162](#)
- description [149](#)
- examples [181, 196](#)
- EXEC statement [165](#)
- FCB module listing [155](#)
- FCB statement [167, 170](#)
- GRAPHIC module
 - listing [161](#)
 - structure [160](#)
- GRAPHIC statement [173, 175](#)
- INCLUDE statement [177](#)
- input [164](#)
- job control statement [165, 166](#)
- JOB statement [165](#)
- module
 - naming conventions [152](#)
 - structure [151](#)
- NAME statement [178](#)
- operation groups [166](#)
- OPTION statement [179](#)
- output [164](#)
- printer models supported [149](#)
- return codes [330](#)
- SYS1.IMAGELIB data set [150](#)
- SYSIN DD statement [165](#)
- SYSPRINT DD statement [165](#)
- SYSUT1 DD statement [165](#)
- TABLE module
 - listing [159](#)
 - structure [158](#)

- IEBIMAGE program (*continued*)
 - TABLE statement [172, 173](#)
 - utility control statement [166, 180](#)
- IEBISAM program
 - description [199](#)
- IEBPDSE program
 - description [201](#)
 - examples [202](#)
 - EXEC JCL statement [201](#)
 - input [201](#)
 - job control statement [201](#)
 - JOB statement [201](#)
 - output [201](#)
 - return codes [331](#)
 - SYSIN DD statement [202](#)
 - SYSPRINT DD statement [202](#)
- IEBTPCH program
 - description [203](#)
 - edited data set [203](#)
 - examples [214, 220](#)
 - EXEC statement [205](#)
 - input [205](#)
 - job control statement [205, 206](#)
 - output [205](#)
 - printing
 - data set [203](#)
 - DBCS [204](#)
 - disk [204](#)
 - partitioned data set [220](#)
 - partitioned directory [204](#)
 - select member [203](#)
 - selected records [204](#)
 - tape [204](#)
 - punching
 - data set [203](#)
 - DBCS [204](#)
 - disk [204](#)
 - partitioned data set [220](#)
 - partitioned directory [204](#)
 - select member [203](#)
 - selected records [204](#)
 - tape [204](#)
 - return codes [331](#)
 - SYSIN DD statement [205](#)
 - SYSPRINT DD statement [205](#)
 - SYSUT DD statement [205](#)
 - SYSUT2 DD statement [205](#)
 - user exit routines [343](#)
 - utility control statement
 - EXITS [210](#)
 - LABELS [213, 214](#)
 - MEMBER [211](#)
 - RECORD [211](#)
 - TITLE [210](#)
 - use of [206](#)
- IEBUPDTE program
 - ALIAS statement syntax [231](#)
 - creating data set
 - example [236](#)
 - example, partitioned data set [236](#)
 - creating master [221](#)
 - creating with card input
 - example [240](#)
 - data statement [230](#)

- IEBUPDTE program (*continued*)
 - deleting records
 - example [236](#)
 - description [221](#)
 - detail statement [228, 229](#)
 - DETAIL statement
 - restrictions [230](#)
 - syntax [228](#)
 - ENDUP statement
 - syntax [232](#)
 - example, new partitioned [241](#)
 - examples [232, 241](#)
 - EXEC statement [222](#)
 - FUNCTION statement
 - description [224](#)
 - restrictions [227](#)
 - syntax [224](#)
 - input [221](#)
 - job control statement [222, 224](#)
 - JOB statement [222](#)
 - LABEL statement syntax [230](#)
 - library
 - adding records [238](#)
 - creating [221](#)
 - example [233, 234, 237, 238, 241](#)
 - insert [237](#)
 - partitioned [234](#)
 - renumber [238](#)
 - SYS1.PROCLIB [233](#)
 - updating [221](#)
 - logical record [230](#)
 - modifying existing data set [221](#)
 - organization [221](#)
 - output [221](#)
 - REPLACE statement example [235](#)
 - return codes [332](#)
 - sequential data set example, copying [241](#)
 - SYSPRINT DD statement [223](#)
 - SYSUT1 DD statement [223](#)
 - SYSUT2 DD statement [223](#)
 - updating data set example, partitioned data set [236](#)
 - utility control statement
 - ENDUP [232](#)
 - LABEL [230](#)
 - use of [224](#)
- IEHINITT program
 - anyname DD statement [247](#)
 - description [243](#)
 - examples [251, 254](#)
 - EXEC statement [247](#)
 - input [246](#)
 - job control statement [247, 248](#)
 - JOB statement [247](#)
 - output [246](#)
 - PARM parameter, EXEC statement [247](#)
 - return codes [332](#)
 - standard label for magnetic tape [245](#)
 - syntax [247](#)
 - SYSIN DD statement [247](#)
 - SYSPRINT DD statement [247](#)
 - utility control statement [248, 250](#)
- IEHLIST program
 - description [257](#)
 - directory

IEHLIST program (*continued*)

directory (*continued*)

partitioned data set [257](#)

PDSE [257](#)

unedited (dump) format [258](#)

examples [264](#), [266](#)

EXEC statement [261](#)

input [260](#)

invoking from application program example [328](#)

job control statement [261](#), [262](#)

JOB statement [261](#)

listing

edited format [257](#), [258](#)

formatted VTOC [357](#)

indexed VTOC [259](#)

partitioned data set directory [257](#), [258](#)

PDSE directory [257](#), [258](#)

unedited (dump) format [258](#), [260](#)

VTOC [258](#), [260](#)

output [260](#)

partitioned data set [262](#)

return codes [332](#)

sample VTOC listing [351](#)

SYSIN DD statement [261](#)

SYSPRINT DD statement [261](#)

utility control statement

LISTPDS statement [262](#)

LISTVTOC statement [263](#)

IEHMOVE program

anyname DD statement [279](#)

buffers [277](#)

copying

BDAM [274](#)

data set [270](#)

group of cataloged data sets [275](#)

multivolume data sets [274](#)

partitioned data set [271](#)

sequential data set [271](#)

unmovable data sets [275](#)

volume [276](#)

description [267](#), [280](#)

examples [289](#), [290](#)

EXCLUDE statement [293](#)

EXEC statement

syntax [277](#)

input [276](#)

job control statement [277](#), [280](#)

JOB statement [277](#)

moving

BDAM [274](#)

data set [270](#)

group of cataloged data sets [275](#)

multivolume data sets [274](#)

partitioned data set [271](#)

sequential data set [271](#)

unloaded data set [274](#)

unmovable data sets [275](#)

volume [276](#)

output [276](#)

partitioned data set

example [291](#)

partitioned data set example [290](#), [291](#)

RACF protection [270](#)

reblocking data set [270](#)

IEHMOVE program (*continued*)

return codes [333](#)

sequential data set example [290](#), [293](#)

SMS volumes, move or copy [276](#)

space allocation [269](#)

SYSIN DD statement [279](#)

SYSPRINT DD statement [279](#)

SYSUT1 DD statement [279](#)

tape DD statement [280](#)

utility control program

COPY DSGROUP [283](#), [284](#)

COPY PDS statement [284](#)

MOVE [284](#)

MOVE DSGROUP [283](#)

MOVE DSNAME statement [281](#)

MOVE PDS statement [284](#)

utility control statement

COPY VOLUME [286](#)

EXCLUDE [288](#)

INCLUDE [287](#)

MOVE VOLUME [286](#)

REPLACE [289](#)

SELECT [288](#)

use of [280](#)

volume size compatibility [268](#)

IEHPROGM program

anyname DD statement [299](#)

description [295](#)

examples [306](#)

EXEC statement [299](#)

input [299](#)

job control statement [299](#), [300](#)

JOB statement [299](#)

output [299](#)

password

adding [298](#)

deleting [298](#)

maintaining [296](#)

password entries, listing [298](#)

renaming

data set [295](#)

member [295](#)

return codes [333](#)

scratching data set or member [295](#)

SYSIN DD statement [299](#)

SYSPRINT DD statement [299](#)

utility control statement

CATLG statement [303](#)

DELETP [305](#)

LIST [306](#)

RENAME statement [302](#)

SCRATCH statement [301](#)

UNCATLG statement [303](#)

use of [301](#)

IFASMFD P 134

IFASMFD P tape [311](#)

IFHSTATR program

description [311](#)

example [313](#)

EXEC statement [313](#)

I/O [311](#)

job control statement [313](#)

JOB statement [313](#)

sample printed output [311](#)

- IFHSTATR program (*continued*)
 - SYSUT1 DD statement [313](#)
 - SYSUT2 DD statement example [313](#)
 - tape quality [311](#)
- image
 - library, IEBIMAGE program [150](#), [177](#)
 - printer, IEBIMAGE program [149](#)
- INCLUDE statement
 - COPY statement
 - DSGROUP [280](#)
 - PDS [280](#)
 - IEBIMAGE program syntax [178](#)
 - IEHMOVE program [287](#)
 - MOVE statement
 - DSGROUP [280](#)
 - PDS [280](#)
- INDD parameter
 - COPY statement [62](#)
 - COPYGROUP statement [64](#)
 - COPYGRP statement [63](#)
 - COPYMOD statement [65](#)
- INDD statement, IEBCOPY program [60](#), [66](#)
- index, copying
 - directory entry processing [127](#)
 - IEBGENER program [127](#)
- Indexed sequential access method [199](#)
- indexed VTOC
 - listing [259](#)
- INHDR/INTLR parameter, user exit routines [343](#)
- INITT statement, IEHINITT program
 - ACCESS parameter [250](#)
 - DISP parameter [249](#)
 - example [256](#)
 - LABTYPE parameter [250](#)
 - NUMBTAPe parameter [250](#)
 - OWNER parameter [249](#)
 - SER parameter [249](#)
 - syntax [249](#)
- input stream [37](#)
- input, IEHPROGM program [299](#)
- INREC/OUTREC parameter, user exit routines [343](#)
- INSERT parameter restrictions [229](#)
- integrated catalog facility [308](#)
- IOERROR parameter, user exit routines [343](#)
- ISAM data sets, converting to VSAM [199](#)
- ISO/ANSI
 - volume access security [250](#)
- ISO/ANSI volume access security [250](#)

J

- JCL (job control language), IEBDG program [98](#)
- job control statement
 - controlling utility programs [6](#)
 - IEBCOMPR program [32](#), [33](#)
 - IEBCOPY program [54](#)
 - IEBEDIT program [119](#), [120](#)
 - IEBGENER program [131](#), [135](#)
 - IEBIMAGE program [165](#), [166](#)
 - IEBPDSE program [201](#)
 - IEBPTPCH program [205](#), [206](#)
 - IEBUPDTE program [222](#), [224](#)
 - IEHINITT program [247](#), [248](#)
 - IEHLIST program [261](#), [262](#)

- job control statement (*continued*)
 - IEHMOVE program [277](#), [280](#)
 - IEHPROGM program [299](#), [300](#)
 - IFHSTATR program [313](#)
- job step
 - copying example [124](#)
 - copying multiple jobs example [122](#)
 - copying to output data set example [122](#)
 - edit copying example [123](#)
 - selective example [123](#), [202](#)

K

- key
 - creating [127](#)
 - output record [137](#)
- KEY parameter, user exit routines [343](#)
- keyboard
 - navigation [361](#)
 - PF keys [361](#)
 - shortcut keys [361](#)
- KEYENCD1 parameter
 - INITT statement [251](#)
- KEYENCD2 parameter
 - INITT statement [251](#)
- KEYLABL1 parameter
 - INITT statement [251](#)
- KEYLABL2 parameter
 - INITT statement [251](#)

L

- label processing [347](#)
- LABELS statement
 - IEBCOMPR program syntax [34](#)
 - IEBGENER program syntax [138](#)
 - IEBPTPCH program syntax [213](#)
 - IEBUPDTE program [230](#)
 - processing routines [346](#)
- LABTYPE parameter
 - INITT statement [250](#)
- LC parameter, IEBCOPY program [56](#)
- library
 - character set module
 - building [194](#), [196](#)
 - creating [162](#), [175](#)
 - description [149](#)
 - IEBIMAGE listing [163](#)
 - listing [194](#), [196](#)
 - structure [163](#)
 - IEBUPDTE program
 - example, creating [234](#)
 - example, partitioned members [234](#)
 - partitioned members [221](#)
 - maintaining [150](#)
 - printer data [149](#)
 - tape quality [311](#)
 - update example [235](#)
- library member
 - creating data set example [236](#)
 - updating data set example [236](#)
- line overrun conditions [170](#), [179](#)
- LINECNT parameter, IEBCOPY program [56](#)

- LINK macro
 - parameter lists [328](#)
 - utility program syntax [315](#)
- LIST parameter, IEBCOPY program [56](#)
- LIST statement syntax, IEHPROGM program [306](#)
- listing
 - edited load modules [257](#)
 - IEBIMAGE program, COPYMOD module [156](#)
 - library character set module example [194](#)
 - partitioned data set directory
 - entries [262](#)
 - example [265](#)
 - password entries [298](#)
 - PDSE directory
 - example [265](#)
 - unedited format [258](#)
 - variables [xxii](#)
 - VTOC
 - edited format [259](#)
 - entries [263](#)
 - example [265](#)
 - IEHLIST program [351](#)
 - indexed [259](#)
 - unedited (dump) format [260](#)
- LISTPDS statement, IEHLIST program
 - syntax [262](#)
- LISTVTOC statement, IEHLIST program
 - syntax [263](#)
- load module
 - alter in place [50](#), [88](#)
 - ALTERMOD statement [61](#)
 - block size [65](#)
 - copying [50](#)
 - listing, edited [257](#)
 - reblocking
 - example [89](#), [90](#)
 - IEBCOPY program [50](#)
 - replacing example [89](#)
- load modules
 - converting to program objects [41](#)
- load processing, IEBCOPY program [46](#)
- loading
 - labeled 7-track tape example [293](#)
 - merging example [87](#)
 - partitioned from sequential data set example [86](#)
 - re-creating a partitioned data set [41](#)
 - sequential data set example [293](#)
- loading library
 - distributing example [89](#)
 - reblocking example [89](#)
- loading processing, IEBCOPY program [46](#)
- logical record
 - IEBCOPY program, length [43](#)
 - IEBUPDTE program, data statement [230](#)
 - length, change [130](#)
- LPP parameter, IEBCOPY program [56](#)

M

- macro
 - BDAM [274](#)
 - CLOSE macro [347](#)
 - LINK [247](#), [315](#)
 - OPEN macro [347](#)

- macro (*continued*)
 - RETURN [344](#)
 - SETPRT [152](#), [165](#)
- macro libraries [221](#)
- MAXBLK parameter, IEBCOPY program [65](#)
- member data records, IEBCOPY unload data set [341](#)
- MEMBER parameter, IEBCOPY program
 - EXCLUDE statement [66](#)
 - renaming member [47](#)
 - SELECT statement [67](#)
- MEMBER statement
 - IEBGENER program
 - description [138](#)
 - syntax [138](#)
 - IEBPTPCH program
 - description [211](#)
 - syntax [211](#)
- merging
 - COPY statement [284](#)
 - copying example [87](#)
 - loading example [87](#)
 - MOVE statement [284](#)
 - partitioned data set example [72](#), [76](#)
 - partitioned data set, IEBCOPY program [41](#)
 - sequential data set, partitioned data sets [143](#)
- merging data sets, directory block allocation [42](#)
- MINBLK parameter, IEBCOPY program [66](#)
- module
 - alias name [151](#)
 - alter in place [50](#), [88](#)
 - copying [50](#)
 - naming conventions for IEBIMAGE [152](#)
 - reblocking [50](#)
 - structure [151](#)
- MOVE DSGROUP statement, IEHMOVE program [283](#), [284](#)
- MOVE DSNAMES statement, IEHMOVE program [281](#), [282](#)
- MOVE PDS statement, IEHMOVE program [284](#), [286](#)
- MOVE VOLUME statement, IEHMOVE program [286](#), [287](#)
- moving
 - basic
 - sequential data set [271](#)
 - BDAM macro [274](#)
 - cataloged data set example [294](#)
 - data set
 - BDAM [274](#)
 - cataloged data sets [275](#)
 - COPYAUTH (copy authorization) [270](#)
 - IEHMOVE program [281](#)
 - multivolume [274](#)
 - partitioned data set [271](#)
 - RACF [270](#)
 - reblocking [270](#)
 - unloaded [274](#)
 - unmovable [275](#)
 - disk volume to separate volume example [290](#)
 - IEHMOVE program example [289](#), [292](#)
 - merging example [290](#)
 - optional
 - sequential data set [271](#)
 - sequential data set [271](#)
 - unsuccessful
 - space allocated [269](#)
 - volume [276](#)
 - volume of data sets example [291](#), [293](#)

- multiple copy operation examples [78, 86](#)
- multitasking environment [300](#)
- multivolume data set
 - copying [274](#)
 - moving [274](#)

N

- NAME statement syntax, IEBIMAGE program [178](#)
- naming
 - modules created by IEBIMAGE [152](#)
 - new image library module [178](#)
- navigation
 - keyboard [361](#)
- nonlabel processing routine [346](#)
- notational conventions, utility programs [xxi](#)
- note list records, IEBCOPY unload data set [340](#)
- NUMBER statement, IEBUPDTE program [228, 230](#)
- NUMTAPE parameter
 - INITT statement [250](#)

O

- objnamex program
 - description [14](#)
- OPEN macro
 - CLOSE macro [347](#)
 - user label [347](#)
- operation groups, IEBIMAGE program [166](#)
- OPTCD=W [58](#)
- OPTION statement, IEBIMAGE program
 - library character set modules [179](#)
 - syntax [179](#)
- OUTHDR/OUTLR parameter, user exit routines [343](#)
- output records, generating with IEBDG [112, 115](#)
- output, IEHPROGM program [299](#)

P

- page header parameter, HDNGLST syntax [328](#)
- page margins, specifying for 3800 and 4248 printers [169](#)
- parameter list, building [325](#)
- parameters
 - label processing [346](#)
 - nonlabel processing [346](#)
- PARM parameter (EXEC statement)
 - IEBGENER program [132](#)
 - IEHINITT program [247](#)
- partitioned data set
 - back up [43](#)
 - block size when copied [65](#)
 - comparing
 - example [38, 39](#)
 - comparing example [38](#)
 - compressing
 - processing considerations [49](#)
 - converting
 - to PDSE [90](#)
 - to sequential [221](#)
 - converting to [42](#)
 - copying
 - example [38, 71](#)
 - excluding members [45](#)

- partitioned data set (*continued*)
 - copying (*continued*)
 - IEBCOPY program [41, 42](#)
 - members with alias names [45](#)
 - selecting members [44](#)
 - copying multiple operations
 - example [86](#)
 - copying multiple operations example [78](#)
 - copying selected members
 - example [75](#)
 - copying selected members example [72, 73](#)
 - creating data set library [221](#)
 - creating from sequential input
 - example [142, 144](#)
 - utility control statements [127](#)
 - creating master [221](#)
 - directory
 - comparing data sets [31](#)
 - copying information [43](#)
 - example [218](#)
 - listing [257](#)
 - printing [204, 262](#)
 - punching [204](#)
 - unedited (dump) format [258](#)
 - listing directory
 - example [265](#)
 - listing members [262](#)
 - loading
 - example [86](#)
 - unload data sets [43](#)
 - member
 - adding [113, 128](#)
 - loading [44](#)
 - renaming [47, 295](#)
 - replacing [46](#)
 - unloading [44](#)
 - merging
 - example [76](#)
 - IEBCOPY program [41](#)
 - sequential data set [143](#)
 - modifying [221](#)
 - moving [271](#)
 - printing a directory
 - example [265](#)
 - printing example
 - data set [203](#)
 - DBCS [220](#)
 - directory [218](#)
 - selected member [203, 215](#)
 - punching [203](#)
 - renaming members example [309](#)
 - replacing select members example [73](#)
 - scratching [295](#)
 - selected records, printed example [219](#)
 - source language modifications [221](#)
 - unload format, IEBCOPY [335](#)
 - unloading [43](#)
 - unused areas (gas) [49](#)
 - updating data set library [221](#)
- partitioned output [138](#)
- password
 - adding [298](#)
 - deleting [298](#)
 - example

- password (*continued*)
 - example (*continued*)
 - defining [308](#)
 - listing [309](#)
 - listing entries [298](#)
 - maintaining [296](#)
 - replacing
 - example [309](#)
 - for data set [298](#)
- patterns of test data
 - IBM supplied [95](#)
 - specifying type [102](#)
 - user-specified
 - example [117](#)
 - format [96](#)
- PDS (partitioned data set)
 - members
 - copy to a PDS [93](#)
 - copy with aliases to a PDS [92](#)
- PDSE
 - validation [201](#)
- PDSE (partitioned data set extended)
 - backing up [43](#)
 - comparing [31](#), [39](#)
 - converting
 - from partitioned data set example [90](#)
 - to partitioned data set [42](#)
 - to sequential [221](#)
 - copying [42](#)
 - creating
 - library of partitioned members [221](#)
 - sequential input [127](#)
 - directory
 - edited list [257](#)
 - listing [257](#), [258](#), [262](#)
 - printing [204](#)
 - punching [204](#)
 - unedited (dump) format [258](#)
 - directory information
 - copying [43](#)
 - directory printing
 - example [265](#)
 - loading [43](#)
 - member
 - adding [128](#)
 - copying [44](#)
 - copying with alias names [45](#)
 - excluding from copy [45](#)
 - renaming [47](#)
 - replacing [46](#), [47](#)
 - modifying [221](#)
 - printing [203](#)
 - program objects
 - replace [91](#), [92](#)
 - to PDSE [91](#)
 - punching [203](#)
 - source language modifications [221](#)
 - unload format, IEBCOPY [335](#)
 - unloading [43](#)
 - updating [221](#)
- PRECOMP parameter, user exit routines [343](#)
- PRINT statement, IEBTPCH program [207](#), [209](#)
- printer
 - 3800 FCB module structure, Model 1 [152](#)
- printer (*continued*)
 - 4248 FCB module structure [153](#)
 - FCB [151](#)
- printer channel codes
 - conventions for channels 1, 9 and 12 [168](#)
 - FCB module [152](#)
 - specifying in FCB statement [168](#)
- printing
 - data set [203](#)
 - DBCS
 - example [220](#)
 - IEBGENER program [130](#)
 - disk [204](#)
 - edited data set [203](#)
 - hexadecimal output example [219](#)
 - IEBIMAGE program [149](#)
 - IFHSTATR program
 - example [313](#)
 - type 21 records [311](#)
 - members of data set [203](#)
 - partitioned data set
 - data set [203](#)
 - directory [204](#), [257](#), [262](#)
 - example [218–220](#), [265](#)
 - example, printing member [215](#)
 - select member [203](#)
 - PDSE
 - directory [204](#), [257](#), [262](#)
 - example, directory [265](#)
 - select member [203](#)
 - record group example [217](#)
 - selected records
 - example [219](#)
 - IEBTPCH program [204](#)
 - sequential data set
 - data set [203](#)
 - example [142](#), [216](#), [218](#)
 - select member [203](#)
 - tape [204](#)
 - titles [210](#)
 - type 21 SMF records example [313](#)
- processing considerations for compress [49](#)
- processing routine
 - label parameters [346](#)
 - nonlabel parameters [346](#)
- program objects
 - alias names [47](#)
 - converting to load modules [41](#)
 - member [47](#)
- protection
 - data set
 - adding passwords [298](#), [308](#)
 - deleting passwords [298](#)
 - example [308](#)
 - replacing passwords [298](#)
 - listing passwords example [309](#)
 - maintaining a password [296](#)
 - password, listing entries of data set [298](#)
 - RACF [270](#)
 - replacing passwords example [309](#)
- PUNCH statement, IEBTPCH program [207](#), [209](#)
- punching
 - data set
 - partitioned [203](#)

punching (*continued*)

data set (*continued*)

PDSE [203](#)

sequential [203](#)

DBCS [204](#)

disk [204](#)

edited data set [203](#)

partitioned data set [203](#)

partitioned data set directory [204](#)

PDSE [203](#)

PDSE directory [204](#)

selected records [204](#)

sequential data set

example [215](#)

IEBTPCH program [203](#)

tape [204](#)

R

RACF (Resource Access Control Facility)

IEHMOVE program [270](#)

reblocking

data set [270](#)

DBCS [130](#)

example [144](#)

load modules [50](#)

recommendations

DFSORT and ICEGENER [127](#)

PDSU block size [44](#)

SuperC utility [31](#)

record

fields

altering contents with IEBDG [103](#)

changing the contents with IEBDG [97](#)

defining contents with IEBDG [95](#), [100](#), [105](#)

rippling contents of [111](#)

group

defining [139](#), [211](#)

dividing sequential data sets [127](#)

fields [212](#)

record formats, IEBCOPY unload data set [335](#)

record group, printing example [217](#)

RECORD statement

IEBGENER program [139](#), [141](#)

IEBTPCH program

defining record group [211](#)

syntax [211](#), [213](#)

referencing aids, special [8](#)

RENAME parameter

MOVE/COPY to SMS volume

[267](#)

RENAME statement

IEHPROGM program [303](#)

RENAME statement, IEHPROGM program [302](#), [303](#)

renaming

data set [295](#)

multivolume data set example [307](#)

partitioned data set example [309](#)

REPEAT statement, IEBDG program [105](#)

REPL statement, IEBUPDTE program [228](#)

REPLACE option, using [47](#)

REPLACE parameter, IEBCOPY program [56](#)

REPLACE statement

IEBUPDTE program [224](#)

REPLACE statement (*continued*)

IEBUPDTE program, example [235](#)

IEHMOVE program [289](#)

IEHPROGM program [304](#)

subordinate control statements [280](#)

replacement level

data set [46](#)

member [47](#)

replacing

data set members [46](#)

select members example [73](#)

REPRO statement, IEBUPDTE program [224](#), [228](#)

restrictions

COPYGROUP statement

Multiple INDD statements [64](#)

COPYGRP statement

EXCLUDE statement [45](#)

Multiple INDD statements [63](#)

detail statement, IEBUPDTE program [230](#)

FUNCTION statement, IEBUPDTE program [227](#)

IEBCOPY program [52](#)

IEBGENER, directory entry processing [127](#)

IEBIMAGE, UCS images [149](#)

INSERT parameter, IEBUPDTE program [229](#)

output records, generating with IEBDG [115](#)

unload data set, IEBCOPY [336](#)

UPDATE parameter, IEBUPDTE program [227](#)

utility programs [8](#)

return codes

IEBDG user exit routine [349](#)

IEBIMAGE program [330](#)

IEBTPCH program [331](#)

IEBUPDTE program [332](#)

IEHINITT program [332](#)

IEHLIST program [332](#)

IEHMOVE program [333](#)

IEHPROGM program [333](#)

totaling routine [348](#)

user exit routine [345](#)

utility programs [328–331](#), [333](#)

RETURN macro

format [344](#)

RETURN macro, exit routine [344](#)

RLD (relocation dictionary), inserting counts [50](#)

S

SCRATCH statement

IEHPROGM program [301](#), [302](#)

SCRATCH statement, IEHPROGM program [301](#)

scratching data set

example [307](#)

IEHPROGM program [295](#)

SDB (storage descriptor block) [132](#), [134](#)

SELECT statement

COPY statement, DSGROUP [280](#)

IEBCOPY program [60](#), [67](#), [68](#)

IEHMOVE program [288](#)

MOVE statement, DSGROUP [280](#)

renaming members [47](#)

replacing members [48](#)

selecting members

for loading [44](#)

for unloading [44](#)

- selecting members (*continued*)
 - to be copied [44](#)
- sequential data set
 - comparing
 - density example [36](#)
 - example [35](#), [37](#), [38](#)
 - IEBCOMPR program [31](#)
 - converting [221](#)
 - copying
 - example [37](#), [144](#), [145](#), [147](#)
 - creating master [221](#)
 - creating partitioned, example [142](#)
 - editing example [145](#), [147](#)
 - example
 - comparing [37](#)
 - copying [37](#)
 - fields [111](#)
 - IEHMOVE program [271](#)
 - merging, partitioned data set [143](#)
 - modifying [221](#)
 - moving [271](#)
 - printing
 - example [142](#), [218](#)
 - IEBTPCH program [203](#)
 - punching
 - example [215](#), [216](#)
 - IEBTPCH program [203](#)
 - reblocking [144](#)
 - renaming [295](#)
 - scratching [295](#)
 - source language modifications [221](#)
 - user specifications, example [216](#)
- SER parameter
 - INITT statement [249](#)
- SETPRT macro [152](#), [165](#)
- shortcut keys [361](#)
- SIO usage count [312](#)
- SIZE parameter, IEBCOPY program [56](#)
- SMF (System Management Facilities), format of type 21 records [311](#)
- SMS (Storage Management Subsystem)
 - indexed VTOC list [260](#)
 - moving [267](#)
 - preallocate data set [267](#)
 - scratching [295](#)
- SMS.IND field in formatted VTOC listing [359](#)
- space allocation
 - IEHMOVE program [269](#)
- SPCLCMOD [57](#)
- standard label
 - magnetic tape volumes [245](#)
- Storage descriptor block (SDB) [132](#), [134](#)
- Storage Management Subsystem [295](#)
- summary of changes [xxvii](#)
- syntax
 - HDNGLST parameter [328](#)
 - IEBCOMPR program
 - COMPARE statement [33](#)
 - EXITS statement [34](#)
 - LABELS statement [34](#)
 - IEBCOPY program
 - ALTERMOD statement [61](#)
 - COPY statement [62](#)
 - COPYGROUP statement [64](#)

- syntax (*continued*)
 - IEBCOPY program (*continued*)
 - COPYGRP statement [63](#)
 - COPYMOD statement [65](#)
 - EXCLUDE statement [66](#)
 - SELECT statement [67](#)
 - IEBEDIT program, EDIT statement [120](#)
 - IEBGENER program
 - EXITS statement [136](#)
 - GENERATE statement [136](#)
 - LABELS statement [138](#)
 - MEMBER statement [138](#)
 - RECORD statement [139](#)
 - IEBIMAGE program
 - CHARSET statement [175](#), [177](#)
 - COPYMOD statement [170](#)
 - FCB statement [167](#)
 - GRAPHIC statement [173](#), [175](#)
 - INCLUDE statement [178](#)
 - NAME statement [178](#)
 - OPTION statement [179](#)
 - TABLE statement [172](#)
 - IEBTPCH program
 - EXITS statement [210](#)
 - LABELS statement [213](#)
 - MEMBER statement [211](#)
 - PRINT statement [207](#)
 - PUNCH statement [207](#)
 - RECORD statement [211](#)
 - TITLE statement [210](#)
 - IEBUPDTE program
 - ALIAS statement [231](#)
 - DETAIL statement [228](#)
 - ENDUP statement [232](#)
 - FUNCTION statement [224](#)
 - LABEL statement [230](#)
 - IEHINITT program [249](#)
 - IEHLIST program
 - LISTPDS statement [262](#)
 - LISTVTOC statement [263](#)
 - IEHMOVE program
 - COPY DSGROUP statement [283](#)
 - COPY DSNAME statement [281](#)
 - COPY PDS statement [284](#)
 - COPY VOLUME statement [286](#)
 - INCLUDE statement [287](#)
 - MOVE DSGROUP statement [283](#)
 - MOVE DSNAME statement [281](#)
 - MOVE PDS statement [284](#)
 - MOVE VOLUME statement [286](#)
 - REPLACE statement [289](#)
 - SELECT statement [288](#)
 - IEHPROGM program
 - ADD statement [304](#)
 - CATLG statement [303](#)
 - DELETEP statement [305](#)
 - EXEC statement [299](#)
 - LIST statement [306](#)
 - RENAME statement [303](#)
 - REPLACE statement [304](#)
 - SCRATCH statement [301](#)
 - UNCATLG statement [303](#)
 - page header parameter, HDNGLST [328](#)
 - utility program, LINK macro [315](#)

- SYS1.IMAGELIB data set
 - maintaining [150](#)
 - storage requirements [149](#)
- SYS1.MAN tape [311](#)
- SYS1.MANX data set [311](#)
- SYS1.MANY data set [311](#)
- SYS1.PROCLIB example [233](#)
- SYS1.VTOCIX data set [264](#)
- SYSIN DD statement
 - IEBCOMPR program [33](#)
 - IEBCOPY program [55](#), [59](#)
 - IEBEDIT program [120](#), [262](#)
 - IEBGENER program [132](#), [135](#)
 - IEBIMAGE program [165](#), [166](#)
 - IEBPDSE program [202](#)
 - IEBPTPCH program [205](#), [206](#)
 - IEBUPDTE program [224](#)
 - IEHINITT program [247](#), [248](#)
 - IEHLIST program [261](#)
 - IEHMOVE program [280](#)
 - IEHPROGM program [299](#), [300](#)
- SYSLIB DD statement
 - IEBPDSE program [202](#)
- SYSMDUMP DD statement
 - IEBCOPY program [55](#), [57](#)
- SYSOUT data set example, printing [218](#)
- SYSPRINT DD statement
 - IEBCOMPR program [32](#)
 - IEBCOPY program [55](#), [57](#)
 - IEBEDIT program [120](#)
 - IEBGENER program [132](#), [133](#)
 - IEBIMAGE program [165](#)
 - IEBPDSE program [202](#)
 - IEBPTPCH program [205](#)
 - IEBUPDTE program [223](#)
 - IEHINITT program [247](#)
 - IEHLIST program [261](#), [262](#)
 - IEHMOVE program [279](#)
 - IEHPROGM program [299](#), [300](#)
- SYSPRINT DD Statement
 - IEHINITT program [247](#)
- system utility programs [5](#)
- system-determined block size [132](#)
- SYSUT DD program, IEBPTPCH program [205](#)
- SYSUT1 DD statement
 - IEBCOMPR program [32](#)
 - IEBCOPY program [55](#), [58](#)
 - IEBEDIT program [120](#)
 - IEBGENER program [132](#), [133](#)
 - IEBIMAGE program [165](#)
 - IEBPTPCH program [206](#)
 - IEBUPDTE program [223](#)
 - IEHMOVE program [279](#)
 - IFHSTATR program [313](#)
- SYSUT2 DD statement
 - IEBCOMPR program [33](#)
 - IEBCOPY program [55](#)
 - IEBEDIT program [120](#)
 - IEBGENER program [132](#)
 - IEBPTPCH program [205](#), [206](#)
 - IEBUPDTE program [223](#)
 - IFHSTATR example [313](#)
- SYSUT3 DD statement, IEBCOPY program [55](#), [59](#)
- SYSUT4 DD statement, IEBCOPY program [55](#), [59](#)

T

- TABLE
 - module listing, IEBIMAGE program [159](#)
 - module structure, IEBIMAGE program [158](#)
 - statement
 - description [172](#)
 - syntax [172](#), [173](#)
- tape
 - input, comparing data sets [37](#)
 - library condition [311](#)
- tape DD statement
 - IEHMOVE program [280](#)
- tape labels
 - creating [254](#)
 - example [252](#), [253](#)
 - IEHINITT program
 - volume [243](#)
- tape-resident data sets, comparing, 7-track tape example [36](#)
- TIOT [300](#)
- TITLE statement, IEBPTPCH program
 - description [210](#)
 - syntax [210](#)
- TOTAL parameter, user exit routines [343](#)
- totaling routine, return codes [348](#)
- translation table, module structure [158](#)

U

- uncataloging data set example [307](#)
- UNCATLG statement (IEHPROGM) [303](#)
- UNCATLG statement, IEHPROGM program [303](#)
- unload data set, IEBCOPY program
 - attribute records [339](#)
 - copying [43](#)
 - DCB parameter [43](#)
 - directory records [339](#)
 - formats
 - invalid [336](#)
 - new [336](#)
 - old (pre-PDSE) [336](#)
 - partitioned data set [335](#)
 - transfer [336](#)
 - loading [43](#)
 - member data records [341](#)
 - note list records [340](#)
 - record formats [335](#)
 - rules and restrictions [336](#)
- unloading
 - copying [274](#)
 - creating sequential data sets [41](#)
 - disk volume example [293](#)
 - excluding members
 - example [75](#)
 - excluding members example [87](#)
 - moving [274](#)
 - partitioned data set [43](#), [335](#)
 - PDSE [43](#)
 - reorganizing [267](#)
 - selecting members example [87](#)
 - unlabeled tape volume example [293](#)
- unloading and compressing partitioned data set or PDSE, example [75](#)
- unmovable data sets

- unmovable data sets (*continued*)
 - copying [275](#)
 - moving [275](#)
- UPDATE parameter restrictions [227](#)
- User ABEND codes [329](#)
- user exit routine, returning to utility program [344](#)
- user interface
 - ISPF [361](#)
 - TSO/E [361](#)
- user label
 - modifying physical record [138](#)
 - processing
 - data [348](#)
 - data set descriptors [347](#)
 - processing with IEBGENER [137](#)
 - system action [347](#)
 - treated as data [214](#)
- user-specified
 - example [117](#)
 - patterns of test data [96](#), [102](#), [109](#), [117](#)
- utility control statement
 - ADD [224](#), [228](#), [304](#)
 - CATLG [303](#)
 - CHANGE [224](#), [228](#)
 - coding [7](#), [8](#)
 - continuing [8](#)
 - COPY DSGROUP [283](#), [284](#)
 - COPY DSNAME [281](#), [282](#)
 - COPY PDS statement [284](#), [286](#)
 - COPY VOLUME [286](#), [287](#)
 - CREATE [105](#), [110](#)
 - DELETE [228](#), [230](#)
 - DELETP [305](#)
 - description [7](#)
 - DSD [100](#)
 - EDIT [120](#), [121](#)
 - END [110](#)
 - ENDUP [232](#)
 - EXCLUDE [288](#)
 - EXITS [34](#), [136](#), [210](#)
 - FD [100](#), [105](#)
 - fields [7](#)
 - format [7](#)
 - GENERATE [136](#)
 - IEBCOMPR program [33](#), [35](#)
 - IEBCOPY program [59](#)
 - IEBIMAGE program [166](#)
 - IEBPTPCH program [206](#), [214](#)
 - IEBUPDTE program [232](#)
 - IEHINITT program [246](#)
 - IEHLIST program [262](#), [264](#)
 - IEHPROGM program [301](#), [306](#)
 - INCLUDE [287](#)
 - INITT [248](#), [250](#)
 - LABEL [230](#)
 - LABELS [137](#), [213](#)
 - LIST [306](#)
 - LISTVTOC [263](#)
 - MEMBER [138](#), [211](#)
 - MOVE DSGROUP [283](#), [284](#)
 - MOVE DSNAME [281](#), [282](#)
 - MOVE PDS statement [284](#), [286](#)
 - MOVE VOLUME [286](#), [287](#)
 - NUMBER [228](#), [230](#)

- utility control statement (*continued*)
 - PRINT [207](#), [209](#)
 - PUNCH [207](#), [209](#)
 - RECORD [139](#), [141](#), [211](#), [213](#)
 - RENAME [302](#), [303](#)
 - REPEAT [105](#)
 - REPL [224](#), [228](#)
 - REPLACE [289](#), [304](#)
 - REPRO [224](#), [228](#)
 - SCRATCH [301](#), [302](#)
 - SELECT [288](#)
 - TITLE [210](#)
 - UNCATLG [303](#)
- utility programs
 - data set [5](#)
 - description [1](#)
 - exit routine
 - overview [343](#)
 - programming considerations [344](#)
 - register contents [344](#)
 - return codes [345](#), [348](#)
 - returning [344](#)
 - totaling [348](#)
 - functions [1](#), [9](#)
 - invoking from application program [315](#)
 - notational conventions [xxi](#)
 - restrictions [8](#)
 - RETURN macro
 - exit routine [344](#)
 - sharing data sets [6](#)
 - system [1](#)

V

- variable-spanned records
 - BDAM data sets
 - copying [274](#)
 - moving [274](#)
- VERSION parameter
 - INITT statement [250](#)
- vertical line spacing [152](#)
- virtual storage requirement
 - graphic character modification module [150](#)
 - library character set [150](#)
- volume
 - table of contents [258](#)
 - volume data set copy [276](#)
 - volume data set move [276](#)
 - volume label initializing [248](#)
 - volume label set [243](#)
 - volume size compatibility
 - IEHMOVE program [268](#)
- VTOC (volume table of contents)
 - formatted listing produced by IEHLIST [357](#)
 - IEHLIST program output [351](#)
 - listing
 - edited format [258](#)
 - entries [263](#)
 - example [265](#)
 - IEHLIST program [258](#), [260](#)
 - indexed [259](#)
 - unedited (dump) format [260](#)

W

WORK parameter, IEBCOPY program [57](#)



Product Number: 5655-ZOS

SC23-6864-70

