File Number S360-32
Order Number GC28-6586-15

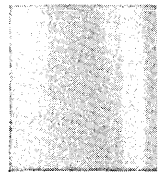
OS

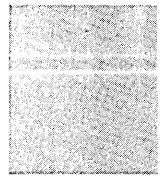
Systems Reference Library

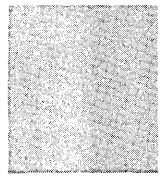
OS Utilities

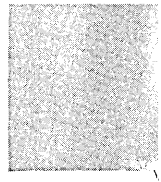
Program Numbers 360S-UT-506
360S-UT-507

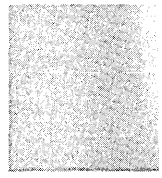
This publication discusses the capabilities of the IBM System/360 Operating System utility programs and the control statements used with each program. These programs are used by programmers responsible for organizing and maintaining operating system data.

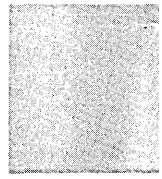


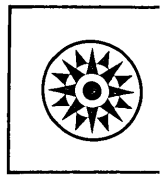


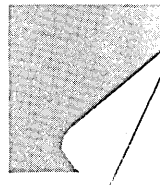












Sixteenth Edition (April 1973)

This edition is a major revision of, and makes obsolete, *IBM System/360 Operating System: Utilities*, Order Number GC28-6586-14. For a summary of the major technical and editorial changes to this edition, see "Summary of Major Changes for Release 21.7."

Technical changes are indicated by a vertical line to the left of the change.

This edition applies to Release 21.7 of the IBM System/360 Operating System. It also applies to any subsequent versions and modification levels until otherwise specified in new editions or technical newsletters.

The information contained in this publication is subject to significant change. Any such changes will be published in new editions or technical newsletters. Before using this publication, consult the latest "*IBM System/360 and System/370 Bibliography*," GA22-6822, and the current SRL Newsletters.

Requests for copies of IBM publications should be made to the IBM branch office that serves you.

A form for reader's comments appears at the back of this publication. Address any additional comments concerning this publication to IBM Corporation, Programming Publications, Post Office Box 1900, Boulder, Colorado 80302.

Front part of book	Back part of book
IBCDASDI	IEBTCRIN
IBCDMPRS	IEBUPDAT—Class C
IBCRCVRP—Class C	IEBUPDTE
ICAPRTBL	IEHATLAS
IEBCOMPR—Class C	IEHDASDR
IEBCOPY	IEHINITT
IEBDG	IEHIOSUP
IEBEDIT	IEHLIST
IEBGENER—Class C	IEHMOVE
IEBISAM—Class C	IEHPROGM
IEBPTPCH—Class C	IFHSTATR

How to Use This Publication

This publication provides a full description of the use of the IBM System/360 Operating System utility programs. This publication assumes that the reader is familiar with IBM System/360 Operating System terms and concepts.

Effective use of this publication requires an understanding of the following:

- Organization of the publication as a whole.
- Organization of each program description.
- Use of special referencing aids that help you find the right utility program and the right example.
- Required publications.
- Related publications.
- Notational conventions used to describe the syntax (or format) of utility control statements.

These topics are discussed below.

Organization of the Publication

In addition to the preface you are now reading, a table of contents, a list of figures, and a list of tables, this publication has these major parts:

- "Summary of Major Changes for Release 21.7," which is a summary of the major changes in this edition.
- "Guide to Utility Program Functions," which is a table arranged in alphabetical order of utility program functions and the programs that perform them. This table enables you to get to the program that can do what you need to have done. For additional information, see "Special Referencing Aids" below.
- "Introduction," which introduces the utility programs and provides information on the differences among system, data set, and independent utility programs. This chapter contains basic information about how the programs are executed and about the utility control statements used to specify program functions. New or infrequent users of the utility programs should give particular attention to this chapter.
- 22 individual chapters—one for each utility program. These chapters are in alphabetical order, beginning with IBCDASDI and ending with IFHSTATR. For a discussion of the organization of these chapters, which will help you find the information you need about a particular program, see "Organization of Program Descriptions" below.
- "Appendix A: Exit Routine Linkage," which provides information about linking to and returning from optional user-supplied exit routines. You should read this appendix if you plan to code or use an exit routine. If you are coding an exit routine, this appendix provides linkage conventions, descriptions of parameter lists, and return codes. If you are using an existing exit routine, you may be interested in the meaning of return codes from the exit routine.
- "Appendix B: Invoking Utility Programs from a Problem Program," which describes the macro instructions used to invoke a utility program from a problem program rather than executing the utility program by job control statements or by a procedure in the procedure library. You should read this appendix if you plan to invoke a utility program from a problem program.
- "Appendix C: DD Statements for Defining Mountable Devices," which provides a review of how to define mountable volumes to ensure that no one else has access to them. For a definitive explanation of this subject, see *OS Job Control Language Reference*, GC28-6704.
- "Appendix D: Generation Data Groups," which describes generation data groups and their indexes, and how to catalog and retrieve generation data sets. This appendix is included because generation data groups are not fully described elsewhere and because you need this background information if you are to manipulate generation data groups with the utility programs. You should read this appendix if you intend to use utility programs to create or manipulate generation data sets.
- "Appendix E: Processing User Labels," which describes the user-label processing that can be performed by IEBGENER, IEBCOMPR, IEBPTPCH, IEHMOVE, IEBTCRIN, and IEBUPDTE. You should read this appendix if you plan to use a utility program for processing user labels.

- "Index," which is a subject index to this publication.

Organization of Program Descriptions

Program descriptions are all organized the same way to enable you to find information more easily. Each program is discussed according to the following pattern:

- Introduction to 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, illustrations, etc.
- Input and output (including return codes) used and produced by the program.
- Control of the program through job control statements and utility control statements. A brief explanation of the job control statements used to execute the program appears in a table under "Job Control Statements." Any restrictions on job control statements appear under a "Restrictions" heading. The utility control statements are introduced in a list under "Utility Control Statements" so that you can determine which of the statements are required for the task to be performed.
- Examples of using the program, including the job control statements and utility control statements.

Special Referencing Aids

Two special referencing aids are included in this publication to help you:

1. Locate the right utility program.
2. Locate the right example.

To locate the right utility program, refer to Table 1 in "Guide to Utility Program Functions," which immediately precedes the "Introduction." Figure 1 shows a portion of the table. The figure shows that you can use IEHINITT to label a magnetic tape volume or IEHLIST to list a volume table of contents.

Label	magnetic tape volumes	IEHINITT
List	a password entry	IEHPROGM
	a volume table of contents	IEHLIST
	partitioned directories	IEHLIST

Figure 1. Locating the Right Program

To locate the right example, use the table—called an "example directory"—that precedes each program's examples. Figure 2 shows a portion of the example directory for IEHMOVE. The figure shows that IEHMOVE Example 1 is an example of moving a sequential data set and that IEHMOVE Example 2 is an example of copying a sequential data set.

MOVE Sequential	2301 Drum, 2311 Disks	Source volume is demounted after job completion. Two mountable disks.	1
COPY Sequential	2311 Disk, 2301 Drum, 2314 or 2319 Disks	Three cataloged sequential data sets are to be copied. The 2314 or 2319 are mountable.	2

Figure 2. Locating the Right Example

Required Publications

The reader should be familiar with the following publications:

- *OS Messages & Codes*, GC28-6631, which contains a complete listing and explanation of the messages and codes issued by the utility programs and other programs.
- *OS JCL Reference*, GC28-6704, which contains a complete explanation of the job control statements available for the operating system.
- *OS Data Management Services Guide*, GC26-3746, which describes the input/output facilities of the operating system. It contains information on record formats, data set organization, access methods, direct access device characteristics, data set disposition, and space allocation.
- *OS Supervisor Services Guide*, GC28-6646, which contains information on how to use the services of the supervisor. Among the services of the supervisor are program management, task creation and management, main storage management, and checkpoint and restart.
- *OS Supervisor and Data Management Macro Instructions*, GC28-6647, which contains a description of the WRITE SZ, LINK, and RETURN macro instructions, and contains the format and contents of the DCB.

Related Publications

The additional publications referred to in this publication are:

- *OS Data Management for System Programmers*, GC28-6550, which contains information on the PASSWORD data set and on writing optional user-supplied exit routines.
- *OS Storage Estimates*, GC28-6551, which contains storage estimates.
- *OS System Control Blocks*, GC28-6628, which contains a complete description of the control blocks used by the operating system.
- *IBM System/360 Principles of Operation*, GA22-6821, which contains a description of system structure; of the arithmetic, logical, branching, status switching, and input/output operations; and of the interruption system.

Notational Conventions

A uniform system of notation is used to describe the syntax (or format) of utility control statements. This notation provides a basis for describing the structure of utility control statements. That is, it describes which parameters are required and which are optional, the options available in expressing values, and the required punctuation.

Bold Type

In the notation, bold type (**LIST**, **0**, etc.) is used to indicate specific values that can be entered.

Italic Type

Italic type (*nn*, *user-information*, etc.) is used where a number, character string, or keyword is to be inserted by the user. For instance, the italic letters in

MODE = *mm*
T*n*

must be replaced by a value when coded.

Punctuation

The period, comma, equal sign, parentheses, and apostrophe are used for punctuation and must be coded as shown. These punctuation marks serve to separate the parameters of a utility control statement.

Brackets

Brackets ([ ]) indicate that the elements and punctuation they enclose are optional. The brackets themselves are for descriptive purposes only, and are not to be coded. For instance

value = element1,element2,element3[,element4]

indicates that "value = " must be followed by three required parameters (element1, element2, and element3) separated by commas. As indicated by the brackets, element4 is optional and need not be coded. If element4 is coded, however, the comma that immediately precedes it must also be coded.

When choices are available for an optional value, the choices appear in brackets, one choice above another, as follows:

```
value = element0 [,element1 ]
                 [,element2 ]
                 [,element3 ]
```

In the above example, "value = " must be followed by element0. Optionally, element1, element2, or element3 can be coded.

Braces

Braces ({ }) indicate a required choice. The braces themselves are for descriptive purposes only, and are not to be coded. For example:

```
value = {element1 }
        {element2 }
```

indicates that "value = " must be followed by either element1 or element2.

Underscoring

Underscoring indicates a value that is assumed by the program if no value is entered for that element. For example, given that no optional value is coded in the following:

```
value = [element1 ]
        [element2 ]
```

element1 is assumed.

Ellipsis

Ellipsis (...) is used to indicate that one or more additional parameters or sets of parameters, each of the same format, optionally can be added to the operand. For example, given the following

value = element1,element2...

the ellipsis indicates that everything preceding the ellipsis and following the equal sign can be repeated.

KEYWORD = device-list

The term KEYWORD is replaced by either VOL, CVOL, FROM, or TO.

The term *device* is replaced by either a generic name, e.g. 3330; or a substitute for a generic name, e.g. DISK, if this substitute has been generated into your system.

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

For tape, the term *list* is replaced by either one or more volume serial number-comma-data set sequence number pairs. Each pair is enclosed in braces and separated from the next pair by a comma. When there is more than one pair, the entire *list* field must be enclosed in parentheses.

Contents

Summary of Major Changes for Release 21.7 17
Major Technical Changes 17
Major Editorial Changes 17

Guide to Utility Program Functions 19

Introduction 23
Control 23
Job Control Statements 23
Utility Control Statements 23
Continuing Utility Control Statements 24
Restrictions 24
Multiprogramming Considerations 24
System Utility Programs 24
Data Set Utility Programs 25
Independent Utility Programs 25
Executing IBCDASDI, IBCDMPRS, and IBCRCVRP 26
Executing ICAPRTBL 26

IBCDASDI Program 27
Initializing a Direct Access Volume 27
Assigning an Alternate Track 27
Input and Output 27
Control 27
Utility Control Statements 27
JOB Statement 28
MSG Statement 28
DADEF Statement 28
VLD Statement 29
VTOCD Statement 30
IPLTXT Statement 30
GETALT Statement 30
END Statement 31
LASTCARD Statement 31
IBCDASDI Examples 31

IBCDMPRS Program 33
Input and Output 33
Control 33
Utility Control Statements 33
JOB Statement 33
MSG Statement 33
DUMP Statement 33
VDRL Statement 35
RESTORE Statement 35
END Statement 36
IBCDMPRS Examples 36

IBCRCVRP Program—Class C 37
Recovering Usable Data 37
Replacing Bad Data 37
Replacement Record 37
Input and Output 37
Control 37
Utility Control Statements 37
JOB Statement 38
MSG Statement 38
RECOVER Statement 38
REPLACE Statement 39
LIST Statement 40
INSERT Statement 40
END Statement 41
IBCRCVRP Examples 41

ICAPRTBL Program 43
Input and Output 43
Control 43
Utility Control Statements 43
JOB Statement 43
DFN Statement 43
UCS Statement 43
FCB Statement 44
END Statement 44
ICAPRTBL Example 44

IEBCOMPR Program—Class C . . . 45
Input and Output . . . 46
Control . . . 46
Job Control Statements . . . 46
Restrictions . . . 46
Utility Control Statements . . . 47
COMPARE Statement . . . 47
EXITS Statement . . . 47
LABELS Statement . . . 47
IEBCOMPR Examples . . . 48

IEBCOPY Program . . . 53
Creating a Backup Copy . . . 53
Copying Data Sets . . . 53
Selecting Members to be Copied . . . 54
Replacing Identically Named Members . . . 54
Replacing Selected Members . . . 55
Renaming Selected Members . . . 55
Excluding Members from a Copy Operation . . . 55
Compressing a Data Set . . . 55
Merging Data Sets . . . 55
Recreating a Data Set . . . 55
Input and Output . . . 55
Control . . . 56
Job Control Statements . . . 56
Restrictions . . . 57
Space Allocation . . . 57
Utility Control Statements . . . 58
COPY Statement . . . 58
SELECT Statement . . . 60
EXCLUDE Statement . . . 61
IEBCOPY Examples . . . 61

IEBDG Program . . . 81
IBM-Supplied Patterns . . . 81
User-Specified Pictures . . . 82
Modification of Selected Fields . . . 82
Input and Output . . . 82
Control . . . 83
Job Control Statements . . . 83
Restrictions . . . 84
PARM Information on the EXEC Statement . . . 84
Utility Control Statements . . . 85
DSD Statement . . . 85
FD Statement . . . 85
CREATE Statement . . . 89
REPEAT Statement . . . 92
END Statement . . . 92
IEBDG Examples . . . 93

IEBEDIT Program . . . 101
Input and Output . . . 101
Control . . . 101
Job Control Statements . . . 101
Restrictions . . . 101
Utility Control Statement . . . 101
EDIT Statement . . . 101
IEBEDIT Examples . . . 103

IEBGENER Program—Class C . . . 107
Creating a Backup Copy . . . 107
Producing a Partitioned Data Set from Sequential Input . . . 107
Expanding a Partitioned Data Set . . . 107
Producing an Edited Data Set . . . 107
Reblocking or Changing Logical Record Length . . . 109
Input and Output . . . 109
Control . . . 109
Job Control Statements . . . 110
Restrictions . . . 110
Utility Control Statements . . . 111
GENERATE Statement . . . 111
EXITS Statement . . . 111
LABELS Statement . . . 112
MEMBER Statement . . . 113
RECORD Statement . . . 113
IEBGENER Examples . . . 115

IEBISAM Program . . . 123
Copying an Indexed Sequential Data Set . . . 123
Creating a Sequential Backup Copy . . . 123
Creating an Indexed Sequential Data Set from an Unloaded Data Set . . . 124
Printing the Logical Records of an Indexed Sequential Data Set . . . 124
Input and Output . . . 125
Control . . . 125
Job Control Statements . . . 126
PARM Information on the EXEC Statement . . . 126
IEBISAM Examples . . . 127

IEBPTPCH Program—Class C . . . 129
Printing or Punching a Data Set . . . 129
Printing or Punching Selected Members . . . 129
Printing or Punching Selected Records . . . 129
Printing or Punching a Partitioned Directory . . . 129
Printing or Punching an Edited Data Set . . . 130
Input and Output . . . 130
Control . . . 130
Job Control Statements . . . 130
Restrictions . . . 131
Utility Control Statements . . . 131
PRINT Statement . . . 132
PUNCH Statement . . . 134
TITLE Statement . . . 135
EXITS Statement . . . 136
MEMBER Statement . . . 136
RECORD Statement . . . 136
LABELS Statement . . . 138
IEBPTPCH Examples . . . 139

IEBTCRIN Program . . . 145
Error Records . . . 145
Error Description Word (EDW) . . . 145
Level Status (Byte 0) . . . 145
Type Status (Byte 1) . . . 146
Start-of-Record (Byte 2) . . . 146
End-of-Record (Byte 3) . . . 146
Sample Error Records . . . 146
MTDI Editing Criteria . . . 148
MTDI Editing Restrictions . . . 148
End-of-Cartridge . . . 149
Input and Output . . . 149
Control . . . 149
Job Control Statements . . . 149
Restrictions . . . 150
Utility Control Statements . . . 150
TCRGEN Statement . . . 151
EXITS Statement . . . 153
Return Codes from IEBTCRIN . . . 157
IEBTCRIN Examples . . . 157

IEBUPDAT Program . . . 159
Input and Output . . . 159
Control . . . 159
Job Control Statements . . . 159
PARM Information on the EXEC Statement . . . 159
Utility Control Statements . . . 160
Header Statement . . . 160
NUMBR Statement . . . 161
DELET Statement . . . 161
Logical Record Statement . . . 162
ALIAS Statement . . . 162
ENDUP Statement . . . 162
IEBUPDAT Examples . . . 162

IEBUPDTE Program . . . 165
Creating and Updating Symbolic Libraries . . . 165
Incorporating Changes . . . 165
Changing Data Set Organization . . . 165
Input and Output . . . 165

Control . . . 166
Job Control Statements . . . 166
Restrictions . . . 166
PARM Information on the EXEC Statement . . . 167
Utility Control Statements . . . 167
Function Statement . . . 167
Detail Statement . . . 171
Data Statement . . . 173
LABEL Statement . . . 173
ALIAS Statement . . . 174
ENDUP Statement . . . 175
IEBUPDTE Examples . . . 175

IEHATLAS Program . . . 185
Input and Output . . . 185
Control . . . 185
Job Control Statements . . . 185
Restrictions . . . 185
Utility Control Statement . . . 186
TRACK or VTOC Statement . . . 186
IEHATLAS Examples . . . 187

IEHDASDR Program . . . 189
Initialize—With Recording-Surface Analysis . . . 189
Initialize—Without Recording-Surface Analysis . . . 189
Changing the Volume Serial Number of a Direct Access Volume . . . 190
Assigning Alternate Tracks for Specified Tracks . . . 190
Creating a Backup, Transportable, or Printed Copy . . . 190
Copying Dumped Data to a Direct Access Volume . . . 191
Input and Output . . . 191
Control . . . 192
Job Control Statements . . . 192
Restrictions . . . 193
PARM Information on the EXEC Statement . . . 193
Utility Control Statements . . . 194
ANALYZE Statement . . . 194
FORMAT Statement . . . 196
LABEL Statement . . . 198
GETALT Statement . . . 198
DUMP Statement . . . 198
RESTORE Statement . . . 200
IPLTXT Statement . . . 201
IEHDASDR Examples . . . 202

IEHINITT Program . . . 209
Placing a Standard Label Set on Magnetic Tape . . . 209
Input and Output . . . 210
Control . . . 210
Job Control Statements . . . 210
Restrictions . . . 210
PARM Information on the EXEC Statement . . . 210
Utility Control Statement . . . 211
INITT Statement . . . 211
IEHINITT Examples . . . 212

IEHIOSUP Program . . . 215
Input and Output . . . 215
Control . . . 215
Job Control Statements . . . 215
Restrictions . . . 215
IEHIOSUP Examples . . . 215

IEHLIST Program . . . 217
Listing Catalog Entries . . . 217
Listing a Partitioned Data Set Directory . . . 217
Edited Format . . . 217
Unedited (Dump) Format . . . 218
Listing a Volume Table of Contents . . . 218
Edited Format . . . 218
Unedited (Dump) Format . . . 220
Input and Output . . . 220
Control . . . 220
Job Control Statements . . . 220
Restrictions . . . 221
PARM Information on the EXEC Statement . . . 221
Utility Control Statements . . . 221
LISTCTLG Statement . . . 221
LISTPDS Statement . . . 222
LISTVTOC Statement . . . 222
IEHLIST Examples . . . 223

IEHMOVE Program . . . 227
Reblocking . . . 230
Moving or Copying a Data Set . . . 230
Moving or Copying a Group of Cataloged Data Sets . . . 232
Moving or Copying a Catalog . . . 233
Moving or Copying a Volume of Data Sets . . . 233
Moving or Copying Direct Data Sets with Variable Spanned Records . . . 234
Input and Output . . . 234
Control . . . 234
Job Control Statements . . . 235
Restrictions . . . 236
PARM Information on the EXEC Statement . . . 236
Job Control Language for the Track Overflow Feature . . . 237
Utility Control Statements . . . 237
MOVE DSNAME Statement . . . 238
COPY DSNAME Statement . . . 239
MOVE DSGROUP Statement . . . 239
COPY DSGROUP Statement . . . 240
MOVE PDS Statement . . . 241
COPY PDS Statement . . . 242
MOVE CATALOG Statement . . . 243
COPY CATALOG Statement . . . 244
MOVE VOLUME Statement . . . 245
COPY VOLUME Statement . . . 245
INCLUDE Statement . . . 246
EXCLUDE Statement . . . 246
SELECT Statement . . . 247
REPLACE Statement . . . 247
IEHMOVE Examples . . . 248

IEHPROGM Program . . . 257
Scratching a Data Set or Member . . . 257
Renaming a Data Set or Member . . . 257
Cataloging or Uncataloging a Data Set . . . 257
Building or Deleting an Index . . . 258
Building or Deleting an Index Alias . . . 259
Connecting or Releasing Two Volumes . . . 259
Building and Maintaining a Generation Index . . . 260
Maintaining Data Set Passwords . . . 261
Adding Data Set Passwords . . . 262
Replacing Data Set Passwords . . . 262
Deleting Data Set Passwords . . . 262
Listing Password Entries . . . 263
Input and Output . . . 263
Control . . . 263
Job Control Statements . . . 264
Restrictions . . . 264
PARM Information on the EXEC Statement . . . 265
Utility Control Statements . . . 265
SCRATCH Statement . . . 265
RENAME Statement . . . 266
CATLG Statement . . . 266
UNCATLG Statement . . . 267
BLDX (Build Index) Statement . . . 267
DLTX (Delete Index) Statement . . . 267
BLDA (Build Index Alias) Statement . . . 268
DLTA (Delete Index Alias) Statement . . . 268
CONNECT Statement . . . 268
RELEASE (Disconnect) Statement . . . 269
BLDG (Build Generation Index) Statement . . . 269
ADD (Add a Password) Statement . . . 269
REPLACE (Replace a Password) Statement . . . 270
DELETEP (Delete a Password) Statement . . . 271
LIST (List Information from a Password) Statement . . . 272
IEHPROGM Examples . . . 272

IFHSTATR Program . . . 277
Input and Output . . . 277
Control . . . 277
Job Control Statements . . . 278
IFHSTATR Example . . . 278

Appendix A: Exit Routine Linkage . . . 279
Linkage to an Exit Routine . . . 279
Label Processing Routine Parameters . . . 279
Nonlabel Processing Routine Parameters . . . 280
Return from an Exit Routine . . . 280

Appendix B: Invoking Utility Programs from a Problem Program 283
LINK or ATTACH Macro Instruction 283
LOAD Macro Instruction 284
CALL Macro Instruction 285

Appendix C: DD Statements for Defining Mountable Devices 287
DD Statement Examples 287

Appendix D: Generation Data Groups 289
Absolute Generation and Version Numbers 289
Relative Generation Numbers 290
Building a Generation Index 290
Creating a New Generation 291
Allocating a Generation 292
Cataloging a Generation 292
Using JCL Procedures to Catalog a Generation 292
Using IEHPROGM to Catalog a Generation 292
Creating an ISAM Data Set as Part of a Generation Data Group 292
Retrieving a Generation 293
Generation Data Group Examples 293

Appendix E: Processing User Labels 297
Processing User Labels as Data Set Descriptors 297
Exiting to a User's Totaling Routine 297
Processing User Labels as Data 298

Index 299

Figures

1. Locating the Right Program 6
2. Locating the Right Example 6
3. Partitioned Directories Whose Data Sets Can Be Compared Using IEBCOMPR . 45
4. Partitioned Directories Whose Data Sets Cannot Be Compared Using IEBCOMPR 45
5. Multiple Copy Operations Within a Job Step 59
6. Copying a Partitioned Data Set—Full Copy 62
7. Copying from Three Input Partitioned Data Sets 63
8. Copy Operation with "Replace" Specified on the Data Set Level 65
9. Copying Selected Members with Reblocking and Deblocking 66
10. Selective Copy with "Replace" Specified on the Member Level 68
11. Selective Copy with "Replace" Specified on the Data Set Level 69
12. Renaming Selected Members Using IEBCOPY 71
13. Exclusive Copy with "Replace" Specified for One Input Partitioned Data Set . . 72
14. Compressing a Data Set in Place 73
15. Compress-in-Place Following Full Copy with "Replace" Specified 74
16. Multiple Copy Operations/Copy Steps 77
17. Multiple Copy Operations/Copy Steps Within a Job Step 79
18. IEBDG Actions 82
19. Defining and Selecting Fields for Output Records Using IEBDG 85
20. Field Selected from the Input Record for Use in the Output Record 86
21. Compatible IEBDG Operations 88
22. Default Placement of Fields Within an Output Record Using IEBDG 91
23. Creating Output Records with Utility Control Statements 91
24. Repetition Due to the REPEAT Statement Using IEBDG 92
25. Output Records at Job Step Completion 94
26. Output Partitioned Member at Job Step Completion 96
27. Partitioned Data Set Members at Job Step Completion 97
28. Contents of Output Records at Job Step Completion 98
29. Creating a Partitioned Data Set from Sequential Input Using IEBGENER 108
30. Expanding a Partitioned Data Set Using IEBGENER 108
31. Editing a Sequential Data Set Using IEBGENER 109
32. An Unloaded Data Set Created Using IEBISAM 124
33. Record Heading Buffer Used by IEBISAM 125
34. Tape Cartridge Reader Data Stream 147
35. Record Construction 147
36. MTDI Codes from TCR 154
37. MTST Codes from TCR 155
38. MTST Codes after Translation by IEBTCRIN with TRANS = STDCL 156
39. Format of System Status Information 170
40. Sequence Numbers and Data Statements to Be Inserted 180
41. Sequence Numbers and Seven Data Statements to Be Inserted 181
42. Direct Access Volume Initialized Using IEHDASDR 190
43. Format of a Direct Access Volume Dumped to a Printer Using IEHDASDR 191
44. IBM Standard Label Group After Volume Receives Data 209
45. Printout of INITT Statement Specifications and Initial Volume Label Information 212
46. Index Structure—Listed by IEHLIST 217
47. Sample Directory Block 217
48. Edited Partitioned Directory Entry 218
49. Sample Partitioned Directory Listing 218
50. Sample Printout of a Volume Table of Contents 219
51. Partitioned Data Set Before and After an IEHMOVE Copy Operation 231
52. Merging Two Data Sets Using IEHMOVE 232
53. Merging Three Data Sets Using IEHMOVE 232
54. Cataloging a Data Set Using IEHPROGM 258
55. Uncataloging a Data Set Using IEHPROGM 258
56. Index Structure Before and After an IEHPROGM Build Operation 259
57. Building an Index Alias Using IEHPROGM 259
58. Connecting a Volume to a Second Volume Using IEHPROGM 260
59. Connecting Three Volumes Using IEHPROGM 260
60. Building a Generation Index Using IEHPROGM 260
61. Relationship Between the Protection Status of a Data Set and Its Passwords . . 261
62. Listing of a Password Entry 263
63. Index Structure After Generation Data Sets Are Cataloged 275
64. Type 21 (ESV) Record Format 277
65. Sample Output from IFHSTATR 277
66. Typical Parameter Lists 284
67. Generation Index—Three Entries 290
68. Relative Positioning—Three Entries in the Catalog 291
69. Generation Index 291
70. System Action at OPEN, EOV, or CLOSE Time 297

Tables

1. Tasks and Utility Programs 19
1. Tasks and Utility Programs 19
2. ICAPRTBL Wait State Codes 26
3. VTOC Entries per Track 30
4. IBCDASDI Example Directory 31
5. Valid 7-Track Tape Unit Modes in IBCDMPRS 34
6. IBCDMPRS Example Directory 36
7. Valid 7-Track Tape Unit Modes in IBCRCVRP 38
8. IBCRCVRP Example Directory 41
9. IEBCOMPR Job Control Statements 46
10. IEBCOMPR Example Directory 48
11. IEBCOPY Job Control Statements 57
12. Changing Input Record Format Using IEBCOPY 57
13. IEBCOPY Example Directory 61
14. IBM-Supplied Patterns 81
15. IEBDG Job Control Statements 83
16. IEBDG Example Directory 93
17. IEBEDIT Job Control Statements 102
18. IEBEDIT Example Directory 103
19. IEBGENER Job Control Statements 110
20. IEBGENER Example Directory 115
21. IEBISAM Job Control Statements 126
22. IEBISAM Example Directory 127
23. IEBPTPCH Job Control Statements 130
24. IEBPTPCH Example Directory 139
25. IEBTCRIN Job Control Statements 149
26. Special Purpose Codes 153
27. IEBTCRIN Return Codes 157
28. IEBTCRIN Example Directory 157
29. IEBUPDAT Job Control Statements 159
30. IEBUPDAT Example Directory 162
31. IEBUPDTE Job Control Statements 166
32. NEW, MEMBER, and NAME Parameters 171
33. IEBUPDTE Example Directory 175
34. IEHATLAS Job Control Statements 186
35. IEHATLAS Example Directory 187
36. IEHDASDR Job Control Statements 192
37. IEHDASDR Example Directory 202
38. IEHINITT Job Control Statements 210
39. IEHINITT Example Directory 212
40. IEHIOSUP Job Control Statements 215
41. IEHIOSUP Example Directory 215
42. IEHLIST Job Control Statements 221
43. IEHLIST Example Directory 223
44. Move and Copy Operations—Direct Access Receiving Volume with Size Compatible with Source Volume 229
45. Move and Copy Operations—Direct Access Receiving Volume with Size Incompatible with Source Volume 229
46. Move and Copy Operations—Nondirect Access Receiving Volume 229
47. Moving and Copying Sequential and Partitioned Data Sets 231
48. Moving and Copying a Group of Cataloged Data Sets 233
49. Moving and Copying the Catalog 233
50. Moving and Copying a Volume of Data Sets 234
51. IEHMOVE Job Control Statements 235
52. IEHMOVE Example Directory 248
53. IEHPROGM Job Control Statements 264
54. IEHPROGM Example Directory 272
55. IFHSTATR Job Control Statements 278
56. Parameter Lists for Nonlabel Processing Exit Routines 280
57. Return Codes Issued by User Exit Routines 281
58. Sequence of DDNMELST Entries 284

Summary of Major Changes for Release 21.7

Following is a summary of major technical and editorial changes.

Major Technical Changes

There are no major technical changes. Minor maintenance corrections have been made where applicable.

Major Editorial Changes

Following is a summary of the major editorial changes to this publication for Release 21.7:

- The examples have been modified, where necessary, to reflect proper placement of continuation for JCL and utility control statements.
- Various examples throughout the publication have been modified to reflect usage of more recent device technologies.
- Additional examples have been included for IEHATLAS, IEHDASDR, and IEHMOVE.

Note: The following programs have been put into Class C Programming Maintenance classification as of December 15, 1972, and are marked "Class C" at the start of each of their descriptions in this publication.

IBCRCVRP	IEBISAM
IEBCOMPR	IEBPTPCH
IEBGENER	IEBUPDAT

Table 1 shows a list of tasks that the utility programs can be used to perform. The left-hand column shows tasks that you might want to perform. The middle column more specifically defines the tasks. The right-hand column shows the utility programs that can be used for each task. Notice that in some cases more than one program may be available to perform the same task.

Table 1. Tasks and Utility Programs

Operation		*Utility*
Add	a password	IEHPROGM
Analyze	tracks on direct access	IEHATLAS, IEHDASDR, IBCDASDI
Assign alternate tracks	to a direct access volume	IEHDASDR, IBCDASDI, IEHATLAS
Build	a generation index	IEHPROGM
	a generation	IEHPROGM
	an index	IEHPROGM
Catalog	a data set	IEHPROGM
	a generation data set	IEHPROGM
Change	data set organization	IEBUPDTE
	logical record length	IEBGENER
	volume serial number of direct access volume	IEHDASDR
Compare	a partitioned data set	IEBCOMPR
	sequential data sets	IEBCOMPR
Compress-in-place	a partitioned data set	IEBCOPY
Connect	volumes	IEHPROGM
Construct	records from MTST and MTDI input	IEBTCRIN
Convert to partitioned	a sequential data set created as a result of an unload	IEBCOPY
	sequential data sets	IEBUPDTE, IEBGENER
Convert to sequential	a partitioned data set	IEBUPDTE
	an indexed sequential data set	IEBISAM, IEBDG
Copy	a catalog	IEHMOVE
	a direct access volume	IEHDASDR, IEHMOVE, IBCDMPRS
	a partitioned data set	IEBCOPY, IEHMOVE
	a volume of data sets	IEHMOVE
	an indexed sequential data set	IEBISAM
	cataloged data sets	IEHMOVE
	dumped data from tape to direct access	IEHDASDR, IBCDMPRS
	job steps	IEBEDIT
	members	IEBUPDAT, IEBGENER, IEBUPDTE, IEBDG
	selected members	IEBCOPY, IEHMOVE
	sequential data sets	IEBGENER, IEHMOVE, IEBUPDTE
	to tape	IBCDMPRS
Create	a library of partitioned members	IEBUPDTE
	a member	IEBDG
	a sequential output data set	IEBDG
	an index	IEHPROGM
	an output job stream	IEBEDIT
Delete	a password	IEHPROGM
	an index structure	IEHPROGM
	records from a member	IEBUPDAT
	records in a partitioned data set	IEBUPDTE
Dump	a direct access volume	IEHDASDR, IBCDMPRS

Operation		*Utility*
Edit	MTDI input	IEBTCRIN
Edit and convert to partitioned	a sequential data set	IEBGENER, IEBUPDTE
Edit and copy	a job stream	IEBEDIT
	a sequential data set	IEBGENER, IEBUPDTE
Edit and list	error statistics by volume (ESV) records	IFHSTATR
Edit and print	a sequential data set	IEBPTPCH
Edit and punch	a sequential data set	IEBPTPCH
Enter	a procedure into a procedure library	IEBUPDTE
Exclude	a partitioned data set member from a copy operation	IEBCOPY, IEHMOVE
Expand	a partitioned data set	IEBCOPY
	a sequential data set	IEBGENER
Generate	test data	IEBDG
Get alternate tracks	on a direct access volume	IEHDASDR, IBCDASDI, IEHATLAS
Include	changes to members or sequential data sets	IEBUPDTE
	source language modifications in a symbolic library	IEBUPDAT
Initialize	a direct access volume	IEHDASDR, IBCDASDI
	a tape	IEHINITT
Insert records	into a partitioned data set	IEBUPDTE
Label	magnetic tape volumes	IEHINITT
List	a password entry	IEHPROGM
	a volume table of contents	IEHLIST
	contents of direct access volume on system output device	IEHDASDR
	number of unused directory blocks and tracks	IEBCOPY
	partitioned directories	IEHLIST
	the contents of the catalog (SYSCTLG data set)	IEHLIST
Load	a previously unloaded partitioned data set	IEBCOPY
	an indexed sequential data set	IEBISAM
	an unloaded data set	IEHMOVE
	UCS and FCB buffers of a 3211	ICAPRTBL
Merge	partitioned data sets	IEHMOVE, IEBCOPY
Modify	a partitioned or sequential data set	IEBUPDTE
Move	a catalog	IEHMOVE
	a volume of data sets	IEHMOVE
	cataloged data sets	IEHMOVE
	partitioned data sets	IEHMOVE
	sequential data sets	IEHMOVE
Number records	in a new member	IEBUPDAT, IEBUPDTE
	in a partitioned data set	IEBUPDTE
Password protect	add a password	IEHPROGM
	delete a password	IEHPROGM
	list passwords	IEHPROGM
	replace a password	IEHPROGM
Print	a sequential data set	IEBGENER, IEBUPDTE, IEBPTPCH
	partitioned data sets	IEBPTPCH
	selected records	IEBPTPCH
Punch	a partitioned data set member	IEBPTPCH
	a sequential data set	IEBPTPCH
	selected records	IEBPTPCH
Read	Tape Cartridge Reader input	IEBTCRIN
Reblock	a partitioned data set	IEBCOPY
	a sequential data set	IEBGENER, IEBUPDTE
Recover	data from defective tracks on direct access volumes	IBCRCVRP, IEHATLAS
Release	a connected volume	IEHPROGM
Rename	a partitioned data set member	IEBCOPY, IEHPROGM
	a sequential or partitioned data set	IEHPROGM
	moved or copied members	IEHMOVE

Operation		*Utility*
Renumber	logical records	IEBUPDTE, IEBUPDAT
Replace	a password	IEHPROGM
	bad data on a defective track	IBCRCVRP
	data on an alternate track	IBCRCVRP, IEHATLAS
	identically named members	IEBCOPY
	logical records	IEBUPDTE
	members	IEBUPDAT, IEBUPDTE
	records in a member	IEBUPDAT, IEBUPDTE
	records in a partitioned data set	IEBUPDTE, IEBCOPY
	selected members	IEBCOPY
	selected members in a move or copy operation	IEHMOVE, IEBCOPY
Restore	a dumped direct access volume from tape	IBCDMPRS, IEHDASDR
Retrieve	usable data from a defective track	IBCRCVRP
Scratch	a volume table of contents	IEHPROGM
	data sets	IEHPROGM
Uncatalog	data sets	IEHPROGM
Unload	a partitioned data set	IEBCOPY, IEHMOVE
	a sequential data set	IEHMOVE
	an indexed sequential data set	IEBISAM
Update	in place a partitioned data set	IEBUPDTE
	TTR entries in the supervisor call library	IEHIOSUP
Write	IPL records and a program on a direct access volume	IBCDASI, IEHDASDR

Introduction

The IBM System/360 Operating System provides utility programs to assist in organizing and maintaining data. Each utility program described in this publication falls into one of three classes of programs. The program class into which a utility program falls is determined by the function that the utility program performs and the manner in which the program is controlled. The program classes are:

- System utility programs, which are used to maintain system control data at an organizational or system level. These programs are controlled by job control statements and utility control statements.
- Data set utility programs, which are used to reorganize, change, or compare data at the data set and/or record level. These programs are controlled by job control statements and utility control statements.
- Independent utility programs, which are used to prepare devices for system use when the operating system is not available. Independent utility programs operate outside, and in support of, IBM System/360 Operating System. These programs are controlled by utility control statements.

The selection of a specific program is dependent on the nature of the job to be performed. For example, renaming a data set involves modifying system control data. Therefore, a system utility program can be used to rename the data set. In some cases, a specific function can be performed by more than one program. Table 1 in "Guide to Utility Program Functions," which immediately precedes this chapter, is provided to help you find the program that performs the function you need.

Control

System and data set utility programs are controlled by job control statements and utility control statements. Independent utility programs are controlled by utility control statements; because these programs are independent of the operating system, job control statements are not required. The job control statements and utility control statements necessary to use utility programs are provided in the major discussion of each utility program.

Job Control Statements

A system or data set utility program can be introduced to the operating system in different ways:

- Job control statements can be included in the input stream.
- Job control statements, placed in a procedure library or defined as an inline procedure, can be included by means of the EXEC job control statement.
- A utility program can be invoked by a calling program.

If job control statements are placed in a procedure library, they should satisfy the requirements for most applications of the program; a procedure, of course, can be modified or supplemented for applications that require additional parameters, data sets, or devices. The data set utility IEBUPDTE can be used to enter a procedure into a procedure library; see "IEBUPDTE Program."

Independent utility programs do not require job control statements and cannot be invoked by a calling program. For information on executing independent utility programs, see "Independent Utility Programs" below.

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 control statements for the IBM System/360 Operating System 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. It can contain from one to eight alphameric characters, the first of which must be alphabetic.

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.

Utility control statements are coded on cards or as card images and are contained in columns 1 through 71. A statement that exceeds 71 characters can be continued on one or more additional cards. The label and operation fields must appear on the first card. The operand, however, can be interrupted after any comma; a comment can be interrupted after any column. Comments can be placed on any card containing a complete or partial operand. However, when a comment is placed on a card with a partial operand, the comment cannot be continued. A nonblank character must be placed in column 72 to indicate continuation.

Continuing Utility Control Statements

The continued portion of the utility control statement must begin in column 16 of the following statement. (Job control language continuations can begin in any column from 4 through 16, and do not require a nonblank character in column 72 for continued operand fields.) Comments can be placed on any card containing a complete or partial statement. However, when a card is included for the sole purpose of continuing a comment, the continuation must begin in column 16.

Note: The IEBCOPY, IBCRCVRP, IEBPTPCH, IEBGENER, IEBCOMPR, and IEBDG utility programs permit certain exceptions to these requirements (see the applicable program description).

The utility control statements are discussed in detail, as applicable, in the remaining chapters.

Restrictions

- A substitute name that represents two or more different device types (for example, DISK, meaning 2311 and 2314) cannot be processed by a utility program.
- Unless otherwise indicated in the description of a specific utility program, a temporary data set can be processed by a utility program only if the user specifies the complete name generated for the data set by the system (for example, DSNAME = SYS68296.T000051.RP001.JOBTEMP.TEMPMOD).

Multiprogramming Considerations

In an MVT environment, a region size should be specified for each application of a system or data set utility program. The region size is determined by the number of bytes in the utility program and by the block sizes of the data sets used in the job step). A region size can be specified as a parameter in the EXEC statement specifying the utility program name. Refer to "Job Control Statements" under each utility program for the minimum region size.

A job that modifies a system data set (identified by SYS1.) must be run in a single job environment; however, a job that uses a system data set, but does not modify it, can be run in a multiprogramming environment. The operator should be informed of all jobs that modify system data sets.

DD statements should ensure that the volumes on which the data sets reside cannot be shared when update activity is being performed.

System Utility Programs

System utility programs manipulate collections of data and system control information. The system utility programs are:

- IEHATLAS, which is used to assign alternate tracks when defective tracks are indicated.
- IEHDASDR, which is used to initialize direct access volumes or to dump or restore data.
- IEHINITT, which is used to write standard labels on tape volumes.
- IEHIOSUP, which is used to update entries in the supervisor call library.
- IEHLIST, which is used to list system control data.
- IEHMOVE, which is used to move or copy collections of data.
- IEHPROGM, which is used to build and maintain system control data.
- IFHSTATR, which is used to select, format, and write information about tape errors from the IFASMFDP tape or the SYS1.MAN data set.

A system utility program is executed or invoked through the use of job control statements and utility control statements.

System utility programs can be executed as jobs or can be invoked as subroutines by a calling program. The invocation of utility programs and the linkage conventions are discussed in "Appendix B: Invoking Utility Programs from a Problem Program."

When using system utility programs, be sure that:

- Each data set to be used by programs other than IEHPROGM, IEHMOVE, and IEHLIST is defined on a DD statement specifying the data set name and DISP = OLD. When updating activity is being performed by IEHPROGM, IEHMOVE, IEHLIST, or IEHDASDR in a multiprogramming environment, other tasks should not be allowed to access the data set being updated. (Refer to "Appendix C: DD Statements for Defining Mountable Devices" for precautions to be taken.)
- DD statements defining mountable devices specify that volumes mounted on those devices cannot be shared.
- Mountable volumes are not made available to the system until the user is requested by the system to mount the specified volumes.
- A reader procedure is used that will direct input and output data sets to volumes other than those which are to be modified by a system utility program.
- When executing a SCRATCH operation, the data set or volume being scratched is not being used by a program executing concurrently.

Data Set Utility Programs

Data set utility programs manipulate partitioned, sequential, or indexed sequential data sets provided as input to the programs. Data ranging from fields within a logical record to entire data sets can be manipulated. The data set utility programs are:

- IEBCOMPR, which is used to compare records in sequential or partitioned data sets.
- IEBCOPY, which is used to copy, compress, or merge partitioned data sets, to select or exclude specified members in a copy operation, and to rename and/or replace selected members of partitioned data sets.
- IEBDG, which is used to create a test data set consisting of patterned data.
- IEBEDIT, which is used to selectively copy job steps and their associated JOB statements.
- IEBGENER, which is used to copy records from a sequential data set or to convert a data set from sequential organization to partitioned organization.
- IEBISAM, which is used to place source data from an indexed sequential data set into a sequential data set in a format suitable for subsequent reconstruction.
- IEBPTPCH, which is used to print or punch records that reside in a sequential or partitioned data set.
- IEBTCRIN, which is used to construct records from the input data stream that have been read from the IBM 2495 Tape Cartridge Reader.
- IEBUPDAT, which is used to incorporate changes to symbolic libraries.
- IEBUPDTE, which is used to incorporate changes to sequential or partitioned data sets.

Data set utility programs can be executed as jobs or can be invoked as subroutines by a calling program. The invocation of utility programs and the linkage conventions are discussed in "Appendix B: Invoking Utility Programs from a Problem Program."

Independent Utility Programs

Independent utility programs operate outside, and in support of, the IBM System/360 Operating System. They are not supported, however, by the 3066 console, which is only used with the Model 165, System/370. If the 3066 is the only console available, execute independent utilities by following step 3b "Executing IBCDASDI, IBCDMPRS, IBCRCVRP" below. The independent utility programs are:

- IBCDASDI, which is used to initialize a direct access volume and to assign alternate tracks.
- IBCDMPRS, which is used to dump and restore the data contents of a direct access volume.
- IBCRCVRP, which is used to recover usable data from a defective track, assign an alternate track, and merge replacement data with the recovered data onto the alternate track.
- ICAPRTBL, which is used to load the forms control and Universal Character Set buffers of a 3211 after an unsuccessful attempt to IPL with the 3211 printer assigned as the output portion of a composite console.

Executing IBCDASDI, IBCDMPRS, and IBCRCVRP

IBCDASDI, IBCDMPRS, and IBCRCVRP are loaded as card decks or as card images on tape. Control statements for the requested program can follow the last card or card image of the program, or can be entered on a separate input device. To execute IBCDASDI, IBCDMPRS, or IBCRCVRP:

1. Place the object program deck in the reader or mount the tape reel that contains the object program.
2. Load the object program from the reader or tape drive by setting the load selector switches and pressing the console LOAD key. When the program is loaded, the wait state is entered and the console lights display the hexadecimal value FFFF.
3. Define the control statement input device in one of the following ways:
   (a) Press the REQUEST key of the console typewriter and, in response to the message "DEFINE INPUT DEVICE", enter "INPUT = *xxxx,cuu*". The *xxxx* is the device type, *c* is the channel address, and *uu* is the unit address. The device type can be 1402, 1442, 2400, 2501, or 2540.
   (b) If the console typewriter is not available, enter at storage location 0110 (hexadecimal): 1*cuu* for a 1442 Card Read Punch; 2*cuu* for a 2400 9-track tape drive; or 0*cuu* for a 2540 Card Read Punch, 2501 card reader, 3410 tape, or 3420 tape. Press the console INTERRUPT key.
4. Control statements are printed on the message output device. At the end of the job, "END OF JOB" is printed on the message output device and the program enters the wait state.

If the job executes IBCRCVRP and the message output device is a tape, the console lights display the hexadecimal value DDDD at a normal end of the job and EEEE at an abnormal end of job. If a machine check occurs, 00E2 is displayed.

Executing ICAPRTBL

ICAPRTBL must be loaded from a card reader. Control statements must follow the last card of the program. Only one printer can be initialized each time the program is executed.

To execute ICAPRTBL:

1. Mount the correct train on the printer and ready the printer.
2. Place the object program deck and the control cards in the card reader. Ready the reader and press the end-of-file key.
3. Load the object program from the reader by setting the load selector switches and pressing the console LOAD key.

Wait state codes will be displayed in the address portion of the PSW for normal termination and for input/output, system or control card errors. Code B01 is issued for normal termination; B02 through B07 are issued for control card errors; B0A through B0C are issued for system errors; and B11 through B1D are issued for input/output errors. Table 2 shows these codes and their meanings. For a detailed discussion of the wait-state codes, see *OS Messages & Codes*, GC28-6631.

Table 2. ICAPRTBL Wait-State Codes

Code	Meaning	Code	Meaning
B01	Visually check the train image printed on the 3211.	B12	Reader not ready.
B02	Missing control card or control card out of order.	B13	Reader unit check (display low main storage location 3 for sense information).
B03	Incorrect JOB statement.	B14	Reader channel error.
B04	Incorrect DFN statement.	B15	No device end on reader.
B05	Incorrect UCS statement.	B19	Printer not online.
B06	Incorrect FCB statement.	B1A	Printer not ready.
B07	Incorrect END statement.	B1B	Printer unit check (display low main storage location 2 through 7 for sense information).
B0A	External interrupt.		
B0B	Program check interrupt.		
B0C	Machine check interrupt.	B1C	Printer channel error.
B11	Reader not online.	B1D	No device end on printer.

IBCDASDI Program

IBCDASDI is an independent utility used to initialize direct access volumes for use and to assign alternate tracks on nondrum, direct access storage volumes. (See "Introduction" for general independent utility information.) IBCDASDI jobs can be performed continuously by stacking complete sets of control statements.

IBCDASDI is not supported on MP65 with the mode switch set to MS; the mode switch must be set to 65.

Initializing a Direct Access Volume

IBCDASDI can be used to initialize a direct access volume. A volume can be initialized with or without surface analysis, a test for defective tracks; however, a surface analysis should be performed when a volume is initialized for the first time.

Note: A 2321 volume is automatically initialized with a surface analysis.

When a volume is initialized, IBCDASDI:

- Checks for tracks that have been previously designated as defective (flagged) and have had alternates assigned. This test must be suppressed when a disk is initialized with surface analysis for the first time. This test must not be suppressed when a volume is initialized without surface analysis.
- Automatically assigns alternates, if necessary, when a volume is initialized with surface analysis. Tracks that are available for disposition as alternates are checked first.
- Writes a track descriptor record (record 0) and erases the remainder of each track. When a volume is initialized with surface analysis, IBCDASDI also writes a standard home address.
- Writes IPL records on track 0 (records 1 and 2).
- Writes volume label on track 0 (record 3) and provides space for additional records, if requested.
- Constructs and writes a volume table of contents (VTOC).
- Writes an IPL program, if requested. When a volume is initialized with surface analysis, the IPL program is written on track 0 for 2301, 2305, 2314, or 2319 volumes or track 1 for 2303 or 2311 volumes. When a volume is initialized without surface analysis, the IPL program is written on track 0 for 2301, 2305, 2314, 2319, or 3330 volumes or on track 1 for 2303 or 2311 volumes.

Note: Defective tracks are flagged and alternate tracks are assigned when the 3330 storage volumes are initialized at the factory. An IBCDASDI job to initialize a 3330 will not perform a surface analysis. The quick DASDI, which can be performed on a 3330 volume, includes: (1) reading alternate tracks and decreasing the total count of the alternates by one when an alternate is found defective or assigned; (2) writing a volume label and VTOC; and (3) writing IPLTXT, if requested. Note that surface analysis is not performed and neither the home address nor record 0 is written on the primary tracks. The **BYPASS** and **FLAGTEST** options of the DADEF statement are ignored. (See "DADEF Statement" below.)

Assigning an Alternate Track

IBCDASDI can be used to: (1) test a track and, if necessary, assign an alternate or (2) bypass testing and automatically assign an alternate.

If testing is performed, an alternate track is assigned for any track found defective. If the defective track is an unassigned alternate, it is flagged to prevent its future use. The alternate track address is made known to the operator.

If a track is tested and not found to be defective, no alternate is assigned. The operator is notified by a message.

If testing is bypassed, an alternate track can be assigned for the specified track or its alternate, whether it is defective or not. If the specified track is an unassigned alternate, it is flagged to prevent its future use.

Input and Output

IBCDASDI uses as input a control data set, which consists of utility control statements.

IBCDASDI produces as output an initialized direct access volume and a message data set.

Control

IBCDASDI is controlled by utility control statements. Because IBCDASDI is an independent utility, operating system job control statements are not used.

Utility Control Statements

IBCDASDI utility control statements in the order in which they must appear are:

- JOB statement, which is used to indicate the beginning of an IBCDASDI job.

- MSG statement, which is used to define an output device for operator messages.
- DADEF statement, which is used to define the volume to be initialized.
- VLD statement, which contains information for constructing an initial volume label and for allocating space for additional labels.
- VTOCD statement, which contains information for controlling the location of the volume table of contents.
- IPLTXT statement, which is used to separate utility control statements from any IPL program text statements.
- GETALT statement, which is used to assign an alternate track on a volume.
- END statement, which is used to indicate the end of an IBCDASDI job.
- LASTCARD statement, which is used to end a series of stacked IBCDASDI jobs.

Note: An IBCDASDI job that initializes a 2321 Data Cell cannot follow one that initializes a different device type unless IBCDASDI is reloaded.

JOB Statement

The JOB statement indicates the beginning of an IBCDASDI job.

The format of the JOB statement is:

```
[label] JOB [user-information]
```

JOB must be preceded and followed by at least one blank.

MSG Statement

The MSG statement defines an output device for operator messages. It follows the JOB statement and precedes any function definition statements.

The format of the MSG statement is:

```
[label] MSG  TODEV = xxxx
            ,TODADDR = cuu
```

where:

TODEV = *xxxx*
specifies the type of output device to receive messages, for example, 1403. The devices that can be specified are 1403, 1443, 1052, 2400, 3210, and 3211.

TOADDR = *cuu*
specifies the channel number, *c*, and unit number, *uu*, of the message output device.

DADEF Statement

The DADEF statement defines the direct access volume to be initialized.

The format of the DADEF statement is:

```
[label] DADEF  TODEV = xxxx
               ,TOADDR = cuu
               [,IPL = YES]
               ,VOLID = {serial  }
                        {SCRATCH}
               [,FLAGTEST = NO]
               [,PASSES = n]
               [,BYPASS = YES]
               [,BIN = d]
               [,MODEL = n]
```

where:

TODEV = *xxxx*
specifies the type of the direct access device to be initialized, for example, 2314. Note that the 2319 disk is functionally equivalent to the 2314 disk. To use a 2319, specify 2314 in the **TODEV** parameter.

TOADDR = *cuu*
specifies channel number, *c*, and unit number, *uu*, of the device.

IPL = YES
specifies that an IPL program is to be written on the volume. An IPL initialization program must be written on a device to be used for system residence. If IPL is omitted, no IPL program is written.

VOLID =
specifies whether a volume serial number check is to be made. These values can be coded:

serial
specifies the volume serial number of the volume to be initialized. If *serial* does not match the volume serial number found on the volume to be initialized, the operator is notified and the job is terminated.

SCRATCH
specifies that no volume serial number check is to be made.

FLAGTEST = NO
specifies that no check is to be made for previously flagged tracks on a disk volume before surface analysis is performed. FLAGTEST = NO should be specified when the disk recording surface is initialized for the first time. Because no check is made for previously flagged tracks on drum volumes or on 2321 volumes, FLAGTEST = NO need not be coded when these devices are initialized.

PASSES = *n*
specifies the number of passes per track to be made in checking for defective tracks. **PASSES** is valid when surface analysis is to be performed or when a quick DASDI is to be performed on a 3330 volume. The value *n* can be 0 through 255. The 0 specification indicates that a quick DASDI is to be performed on a 3330 volume. For a 3330 volume, a value greater than 0 causes no check to be made for defective tracks. A specification of 1 through 255 indicates the number of passes to be made per track for volumes other than a 3330 volume. If **PASSES** is omitted, one pass is made per track. This parameter does not apply to 2321 volumes.

BYPASS = YES
specifies that no check is to be made for defective tracks. If **BYPASS** is omitted, tracks are checked and those found defective are automatically assigned alternates. This parameter applies only when surface analysis is not to be performed; it does not apply to 2321 volumes.

BIN = *d*
specifies the decimal number of a bin to be initialized. The value of *d* can be 0 through 9. This parameter applies only to 2321 volumes.

MODEL = *n*
specifies a decimal model number (1 or 2). This parameter corresponds to the 2305-1 and 2305-2, respectively. **MODEL** is required when a 2305 is to be initialized.

VLD Statement

The VLD statement contains information for constructing an initial volume label and for allocating space for additional labels.

The format of the VLD statement is:

```
[label] VLD  NEWVOLID = serial
             [,VOLPASS = {0 }
                         {1 }]
             [,OWNERID = xxxxxxxxxx]
             [,ADDLABEL = n]
```

where:

NEWVOLID = *serial*
specifies a one- to six-character volume serial number.

VOLPASS =
specifies the value of the volume security bit. These values can be coded:

0
specifies that the volume is not security protected. If **VOLPASS** is omitted, 0 is assumed.

1
specifies that the volume is security protected.

OWNERID = *xxxxxxxxxx*
specifies a one- to ten-character field that identifies the owner of the volume. If **OWNERID** is omitted, no identification is given.

ADDLABEL = *n*
specifies the total number of additional labels for which space is to be allocated. The value of *n* can be 1 through 7. If **ADDLABEL** is omitted, 0 is assumed.

VTOCD Statement

The VTOCD statement contains information for controlling the location of the volume table of contents.

The format of the VTOCD statement is:

```
[label] VTOCD  STRTADR = nnnnn
              ,EXTENT = nnnn
```

where:

STRTADR = *nnnnn*
specifies the one- to five-byte track address, relative to the beginning of the volume, at which the volume table of contents is to begin. The VTOC cannot occupy track 0 or any alternate track.

EXTENT = *nnnn*
specifies the length (number of tracks) of the VTOC.

Table 3 shows the number of VTOC entries per track for each device type.

Table 3. VTOC Entries per Track

Device	VTOC Entries per Track
2301	63
2314	25
2319	25
2302	22
2303	17
2311	16
2321	8
2305-1	18
2305-2	34
3330	39

IPLTXT Statement

The IPLTXT statement separates utility control statements from IPL program text statements. It is required only when IPL text is included.

The format of the IPLTXT statement is:

```
IPLTXT
```

When IPL text is included, **END** must start in column 2. See "END Statement" below.

GETALT Statement

The GETALT statement is used to assign an alternate track on a volume. Any number of alternate tracks can be assigned in a single job by including a GETALT statement for each track.

Note: A GETALT statement that applies to a 3330 device causes an alternate track to be assigned automatically without testing.

The format of the GETALT statement is:

```
[label] GETALT  TODEV = xxxx
                ,TOADDR = cuu
                ,TRACK = ccccchhhh
                ,VOLID = serial
                [,FLAGTEST = NO]
                [,PASSES = n]
                [,BYPASS = YES]
                [,BIN = d]
                [,MODEL = n]
```

where:

TODEV = *xxxx*
specifies the device type of the direct access device.

TOADDR = *cuu*
specifies the channel number, *c*, and unit number, *uu*, of the direct access device.

TRACK = *ccccchhhh*
specifies the address of the track for which an alternate is requested, where cccc is the cylinder number and *hhhh* is the head number.

VOLID = *serial*
specifies the volume serial number of the volume to which an alternate track is to be assigned. If *serial* does not match the volume serial number found on this volume, the operator is notified and the job is terminated.

FLAGTEST = NO
specifies that no check is to be made for a previously flagged track before a surface analysis for a disk volume is performed on this track. This parameter is used when testing before assigning an alternate.

PASSES = *n*
specifies the number of passes, *n*, to be made when performing a surface analysis on this track. The value of *n* can be 1 through 255. If **PASSES** is omitted, one pass is made. (If, however, the GETALT statement applies to a 3330 volume, an alternate track is assigned without testing; the **PASSES** parameter is ignored.) This parameter is used when testing before assigning an alternate.

BYPASS = YES
specifies that no check for a defective track is to be made. If **BYPASS** is omitted, the program assigns an alternate only if it finds that the specified track is defective.

BIN = *d*
specifies the decimal number of a bin to be initialized. The value of *d* can be 0 through 9. This parameter applies only to 2321 volumes.

MODEL = *n*
specifies a decimal model number (1 or 2). This parameter corresponds to the 2305-1 and 2305-2, respectively. **MODEL** is required when a 2305 is to be initialized.

The GETALT function should not be used immediately after a RESTORE operation that did not complete successfully. Before using GETALT in such a case, reinitialize the volume, if possible.

END Statement

The END Statement denotes the end of job. It appears after the last function definition statement.

The format of the END statement is:

```
[label] END [user-information]
```

END must be preceded and followed by at least one blank. When IPL text is included, END must start in column 2.

LASTCARD Statement

The LASTCARD statement is required only when an IBCDASDI job or a series of stacked IBCDASDI jobs is followed by other statements on the control statement input device. The LASTCARD statement must follow the last END statement applying to an IBCDASDI job.

The format of the LASTCARD statement is:

```
LASTCARD
```

IBCDASDI Examples

The examples that follow illustrate some of the uses of IBCDASDI. Table 4 can be used as a quick reference guide to IBCDASDI examples. The numbers in the "Example" column point to examples that follow.

Table 4. IBCDASDI Example Directory

Operation	Comments	Example
Initialize	A 2305 volume is to be initialized with surface analysis.	1
Initialize	A 2305 volume is to be initialized without surface analysis.	2
Initialize	A 3330 volume to be used as the system residence volume is to be initialized. An IPL program is included in TXT format.	3
Assign alternate tracks	Three alternate tracks are to be assigned on a 3330 volume.	4

IBCDASDI Example 1

In this example, a 2305 volume is initialized for the first time. A surface analysis is performed with the initialization.

The example follows:

```
INIT  JOB  'INITIALIZE 2305'
        MSG TODEV=1403,TOADDR=00E
      DADEF TODEV=2305,TOADDR=140,VOLID=SCRATCH,FLAGTEST=NO
        VLD NEWVOLID=111111
      VTOCD STRTADR=50,EXTENT=10
        END
```

The control statements are discussed below:

- JOB initiates the IBCDASDI job.
- MSG defines the 1403 on channel 0, unit 0E, as the output message device.

- DADEF specifies that a 2305 volume on channel 1, unit 40, is to be initialized. Because the volume is being initialized for the first time, no check is to be made for previously flagged tracks.
- VLD specifies 111111 as the volume serial number of the volume to be initialized.
- VTOCD specifies the starting address and length in tracks of the volume table of contents.

IBCDASDI Example 2

In this example, a 2305 volume is initialized for the first time. No surface analysis is performed with the initialization.

The example follows:

```
INIT  JOB  INITIALIZE 2305
        MSG TODEV=1403,TOADDR=00E
      DADEF TODEV=2305,TOADDR=140,VOLID=SCRATCH,BYPASS=YES
        VLD NEWVOLID=230500
      VTOCD STRTADR=1,EXTENT=3
        END
```

The control statements are discussed below:

- DADEF specifies that a 2305 volume is to be initialized and specifies the channel and unit number. No check is to be made for the volume serial number or for defective tracks.
- VLD specifies the volume serial number of the volume to be initialized.
- VTOCD specifies that the volume table of contents is to begin on track 1 and is to extend over three tracks.
- END specifies the end of the IBCDASDI job.

IBCDASDI Example 3

In this example, a 3330 volume is initialized for later use as a system residence volume. An IPL program is included in standard TXT format.

The example follows:

```
INIT       JOB  'INITIALIZE 3330'
             MSG TODEV=1403,TOADDR=00E
           DADEF TODEV=3330,TOADDR=150,IPL=YES
             VLD NEWVOLID=P00001,OWNERID=BROWN,ADDLABEL=2
           VTOCD STRTADR=2,EXTENT=9
          IPLTXT
(IPL program text statements)
 END
```

The control statements are discussed below:

- DADEF specifies that a 3330 volume is to be initialized and specifies the channel number and unit number. An IPL program is to be included.
- VLD specifies a volume serial number and owner identification for the volume to be initialized. It also specifies that space is to be allocated for two additional labels.
- VTOCD specifies that the volume table of contents is to begin on track 2 and is to extend over nine tracks.
- IPLTEXT specifies the beginning of IPL program text statements.
- END specifies the end of IPL program text statements. Because IPL text is included, **END** begins in column 2.

IBCDASDI Example 4

In this example, three alternate tracks are assigned to a 3330 volume, without reinitialization of the volume. The check for a defective track is bypassed when the first two of the three tracks are assigned.

The example follows:

```
ALTRK      JOB  ASSIGN ALTERNATE TRACKS ON 3330
             MSG TODEV=1052,TOADDR=009
STMT1    GETALT TODEV=3330,TOADDR=150,VOLID=P00002,          ⟪C
               BYPASS=YES,TRACK=006F0001
STMT2    GETALT TODEV=3330,TOADDR=150,VOLID=P00002,          ⟪C
               BYPASS=YES,TRACK=00910004
STMT3    GETALT TODEV=3330,TOADDR=150,                       ⟪C
               TRACK=004B0007,VOLID=P00002
             END
```

The control statements are discussed below:

- The first and second GETALT statements bypass the check for defective tracks.
- The third GETALT statement causes the check for a defective track to be made because **BYPASS** is not included.

IBCDMPRS Program

IBCDMPRS is an independent utility used to dump and restore data on direct access volumes. (See "Introduction" for general independent utility information.)

The data contents of a direct access volume (all data except the home address) can be *dumped* to 2311, 2314, 2319, 2305, 3330, or tape volumes and restored to a direct access volume that resides on the same type of device as the source volume. Both the source volume and the volume to which data is to be restored must have been initialized to IBM System/360 Operating System specifications. IBCDMPRS is useful for preparing transportable copies and backup copies of direct access volumes.

IBCDMPRS is not supported on MP65 with the mode switch set to MS; the mode switch must be set to 65.

Input and Output

IBCDMPRS uses as input:

- A control data set, which contains utility control statements.
- A data set to be dumped to tape or to be restored to a direct access volume.

IBCDMPRS produces as output:

- A data set dumped to tape or a data set restored to a direct access volume.
- A message data set.

Control

IBCDMPRS is controlled by utility control statements. Because IBCDMPRS is an independent utility, operating system job control statements are not used.

Utility Control Statements

IBCDMPRS utility control statements are:

- JOB statement, which is used to begin an IBCDMPRS job.
- MSG statement, which is used to define an output device for operator messages.
- DUMP statement, which is used to identify the volume to be dumped and the receiving volume.
- VDRL Statement, which is used to specify the upper and lower track limits of a partial dump.
- RESTORE statement, which is used to identify the source volume whose data is to be restored and the receiving volume.
- END statement, which is used to indicate the end of an IBCDMPRS job.

JOB Statement

The JOB statement indicates the beginning of a job.

The format of the JOB statement is:

```
[label] JOB [user-information]
```

JOB must be preceded and followed by at least one blank.

MSG Statement

The MSG statement defines an output device for operator messages. It follows the JOB statement and precedes any function definition statements.

The format of the MSG statement is:

```
[label] MSG   TODEV = xxxx
              ,TOADDR = cuu
```

where:

TODEV = *xxxx*
specifies the type of the output device to receive messages, for example, 1403. The devices that can be specified are 1403, 1443, 1052, 2400, 3210, and 3211.

TOADDR = *cuu*
specifies the channel number, *c*, and unit number, *uu*, of the message output device.

DUMP Statement

The DUMP statement is used to identify both the source volume whose contents are to be dumped and the receiving volume. The data contents of the entire source volume are dumped, including any data on alternate tracks. If both the source and receiving volumes reside on 2311, 2314, 2319, or 3330 volumes, the receiving volume is an exact replica of the source volume.

The format of the DUMP statement is:

```
[label] DUMP  FROMDEV = xxxx
              ,FROMADDR = cuu
              ,TODEV = xxxx
              ,TOADDR = cuu
              [,VOLID = serial[,serial]]
              [,MODE = mm]
              [,BIN = d]
              [,MODEL = n]
```

where:

FROMDEV = *xxxx*
specifies the type of the source device, for example, 3330.

FROMADDR = *cuu*
specifies channel number, c, and unit number, *uu*, of the source device.

TODEV = *xxxx*
specifies the type of the receiving device, for example, 2400. If the receiving device is a tape drive and no **MODE** parameter is specified, the data is written at the highest density supported by the device. (For 7-track tape, the default mode is 93.)

TOADDR = *cuu*
specifies the channel number, *c*, and unit number, *uu*, of the receiving device.

VOLID = *serial*[,*serial*]...
specifies the volume serial numbers of the receiving volumes to which data is to be dumped. **VOLID** is required when the receiving volume has been initialized to operating system specifications. If *serial* does not match the volume serial number found on the receiving volume, the operator is notified and the job is terminated. If **VOLID** is not specified and the receiving volume contains a volume serial number, the operator is notified.

MODE = *mm*
specifies the bit density for data written onto the receiving magnetic tape volume. This parameter is applicable to 7-track tape drives and to 9-track tape drives with density selections of 800 and 1600 bits per inch. Valid modes for 7-track tape are shown in Table 5. (Only those modes that set the data converter on are accepted.) For 9-track tape with density selections of 800 and 1600 bits per inch, the mode settings are CB and C3, respectively. If the receiving device is not a tape drive, the **MODE** parameter is ignored. If the receiving device is a tape drive but no mode is specified, the data is written at the highest density supported by the device.

BIN = *d*
specifies the decimal number of a bin to be dumped. The value of *d* can be 0 through 9. This parameter applies only to 2321 volumes. When a 2321 volume is to be dumped to a 2311, 2314, 2319, or 3330 volume, *d* must be 0.

MODEL = *n*
specifies a decimal model number (1 or 2) for a 2305. This parameter is applicable only when a 2305 is specified. If **MODEL** is omitted, 2305-1 is assumed.

Note: The 2319 disk is functionally equivalent to the 2314 disk. To use the 2319, specify 2314.

Dump time can be minimized by selecting devices assigned to different channels. For example:

```
DUMP FROMDEV=3330,FROMADDR=150,TODEV=2400,TOADDR=282
```

Table 5 shows valid modes for 7-track tape that can be entered for the **MODE** parameter.

Table 5. Valid 7-Track Tape Unit Modes in IBCDMPRS

Mode (mm)	*Density (bits-per-inch)*	*Translator*	*Data Converter*	*Parity*
13	200	Off	On	Odd
53	556	Off	On	Odd
93	800	Off	On	Odd

VDRL Statement

The VDRL (volume dump/restore limits) statement is used to specify the upper and lower limits of a partial dump. If a track within these limits has had an alternate assigned to it, the data on the alternate track is included in the dump. When the VDRL statement is used, it must be preceded by a DUMP statement and must be followed by an END statement.

The format of the VDRL statement is:

```
[label] VDRL  BEGIN = nnnnn
              [,END = nnnnn]
```

where:

BEGIN = *nnnnn*
specifies a one- to five-byte relative track address that identifies the first track to be dumped.

END = *nnnnn*
specifies the relative track address of the last track to be dumped. If only one track is to be dumped, this address is the same as the beginning address. If **END** is omitted, the last track of the volume, excluding those tracks reserved as alternates, is assumed to be the upper limit.

RESTORE Statement

The RESTORE statement is used to identify both the source volume whose data contents are to be restored and the receiving volume.

Note: IBCDMPRS can be used to restore a tape created by IEHDASDR. Conversely, IEHDASDR can be used to restore a tape created by IBCDMPRS.

The format of the RESTORE statement is:

```
[label] RESTORE  FROMDEV = xxxx
                 ,FROMADDR = cuu
                 ,TODEV = xxxx
                 ,TOADDR = cuu
                 ,VOLID = serial
                 [,MODE = mm]
                 [,BIN = d]
                 [,MODEL = n]
```

where:

FROMDEV = *xxxx*
specifies the type of the source device, for example, 2400.

FROMADDR = *cuu*
specifies the channel number, *c*, and unit number, *uu*, of the source device.

TODEV = *xxxx*
specifies the type of the receiving device, for example, 3330. This device type must be the same as the device containing the volume originally dumped.

TOADDR = *cuu*
specifies the channel number, *c*, and unit number, *uu*, of the receiving device.

VOLID = *serial*
specifies the volume serial number of the receiving volume. If *serial* does not match the volume serial number found on the receiving volume, the operator is notified and the job is terminated.

MODE = *mm*
specifies the bit density for data written to the receiving tape volume. This parameter must match the mode specified when data was written to the source volume. **MODE** should not be specified if the source and receiving volumes are not tape or if **MODE** was not specified when data was written to the source volume. Valid modes are shown earlier in Table 5. (Only those modes that set the data converter on are accepted.) For 9-track tape drives with density selections of 800 and 1600 bits per inch, the mode settings are CB and C3, respectively.

BIN = *d*
specifies the decimal number of a bin to be restored. This parameter applies only to 2321 volumes. The value of *d* can be 0 through 9. When a 2311, 2314, 2319, 2301, 2302, 2303, 2305, or 3330 volume is to be restored to a 2321 volume, *d* must be 0.

MODEL = *n*
specifies a decimal model number (1 or 2) for a 2305. If **MODEL** is omitted, 2305-1 is assumed.

Restore time can be minimized by selecting devices assigned to different channels. For example:

```
RESTORE FROMDEV = 2400,FROMADDR = 282,TODEV = 3330,TOADDR = 150
```

END Statement

The END statement marks the end of job. It appears after the last function definition statement.

The format of the END statement is:

```
[label] END [user-information]
```

END must be preceded and followed by at least one blank.

IBCDMPRS Examples

The examples that follow illustrate some of the uses of IBCDMPRS. Table 6 can be used as a quick reference guide to the examples. The numbers in the "Example" column point to examples that follow.

Table 6. IBCDMPRS Example Directory

Operation	Comments	Example
DUMP	A direct access volume is to be dumped to a tape volume.	1
RESTORE	A data set dumped to tape is to be restored to a direct access volume.	2

IBCDMPRS Example 1

In this example, a direct access volume is dumped to a tape volume.

The example follows:

```
DUMP       JOB   DUMP 3330 ONTO TAPE
            MSG   TODEV=1052,TOADDR=009
           DUMP   FROMDEV=3330,FROMADDR=150,                    ℛC
                  TODEV=2400,TOADDR=280
 END
```

IBCDMPRS Example 2

In this example, dumped data is restored to a direct access volume.

The example follows:

```
RESTORE    JOB   RESTORE 3330 FROM TAPE
            MSG   TODEV=1052,TOADDR=009
      RESTORE   FROMDEV=2400,FROMADDR=280,TODEV=3330,           ℛC
                TOADDR=150,VOLID=PZ1111
 END
```

IBCRCVRP is an independent utility used to retrieve usable data from a defective track, to assign an alternate track, and to merge the usable data with replacement data on the alternate track. (See "Introduction" for general independent utility information.) IBCRCVRP will perform the recovery function on only the following devices: 2302, 2303, 2311, 2314, 2319, 2321.

IBCRCVRP is not supported on MP65 with the mode switch set to MS; the mode switch must be set to 65.

Note: IEHATLAS, a system utility program, can be used to perform these operations under control of the operating system.

Recovering Usable Data

IBCRCVRP can be used to retrieve data from a defective track, write this data on a receiving tape, and list the bad records on the message output device.

Replacing Bad Data

IBCRCVRP can be used to merge data recovered from a defective track with replacement data and write the result on an assigned alternate track. (Alternate tracks must be assigned manually on drum volumes.)

Replacement Record

A replacement record is an 80-byte card image that contains replacement data for bad fields.

A replacement record must be supplied if a key or data field is found to be bad. The replacement record is described by column, as follows:

- Columns 1 through 8 contain either "**I/D** = *xxx*" or "**I/D = LAST**". The value of I/D must be the same as that of the RECORD parameter of the associated INSERT statement. See "INSERT Statement" below.
- Columns 9 and 10 are blank.
- Columns 11 through 80 contain replacement data in hexadecimal. The number of bytes must be the same as that specified in the COUNT parameter of the INSERT statement. See "INSERT Statement" below.

Replacement records can be continued. The continuation records must start in column 11 with the continued replacement data.

When the same device is used to read both control statements and replacement records, a replacement record must follow the INSERT statement that describes it.

Input and Output

IBCRCVRP uses as input:

- A control data set, which contains utility control statements.
- A data set from which usable data on a defective track is to be recovered or a data set on which bad data is to be replaced.
- Replacement data if bad data is to be replaced on a data set.

IBCRCVRP produces as output:

- A data set from which usable data has been recovered or a data set on which bad data has been replaced.
- A message data set.

Control

IBCRCVRP is controlled by utility control statements. Because IBCRCVRP is an independent utility, operating system job control statements are not used.

Utility Control Statements

IBCRCVRP utility control statements are:

- JOB statement, which is used to begin an IBCRCVRP job.
- MSG statement, which is used to define an output device for operator messages.
- RECOVER statement, which is used to identify the direct access volume that contains the defective track, the defective track, and a receiving volume.
- LIST statement, which is used to request that the contents of the defective track be listed.
- REPLACE statement, which is used to identify the tape device on which the volume containing recovered data resides and the direct access volume on which recovered data is to be merged with replacement data.
- INSERT, which identifies the device on which the replacement record volume resides.
- END statement, which is used to indicate the end of an IBCRCVRP job.

JOB Statement

The JOB statement indicates the beginning of an IBCRCVRP job.

The format of the JOB statement is:

```
[label] JOB [user information]
```

JOB must be preceded and followed by at least one blank.

MSG Statement

The MSG statement defines an output device for operator messages. It follows the JOB statement and precedes any function definition statements.

The format of the MSG statement is:

```
[label] MSG  TODEV = xxxx
             ,TOADDR = cuu
             [,MODE = mm]
```

where:

TODEV = *xxxx*
specifies the type of the output device to receive messages, for example, 1403. The devices that can be specified are 1403, 1443, 1052, and 2400.

TOADDR = *cuu*
specifies the channel number, *c*, and unit number, *uu*, of the message output device.

MODE = *mm*
specifies the mode in which the message output tape is to be written. This parameter is valid only when the message output device is 7-track tape. Valid modes are shown in Table 7. If **MODE** is omitted, the following assumptions (6B) are made: density is 556 bits per inch, translator is on, data converter is off, and the parity is even.

Table 7 shows the values that can be entered for the **MODE** parameter.

Table 7. Valid 7-Track Tape Unit Modes in IBCRCVRP

Mode (mm)	*Density (bits per inch)*	*Translator*	*Data Converter*	*Parity*
13	200	Off	On	Odd
23	200	Off	Off	Even
33	200	Off	Off	Odd
2B	200	On	Off	Even
3B	200	On	Off	Odd
53	556	Off	On	Odd
63	556	Off	Off	Even
73	556	Off	Off	Odd
6B	556	On	Off	Even
7B	556	On	Off	Odd
93	800	Off	On	Odd
A3	800	Off	Off	Even
B3	800	Off	Off	Odd
AB	800	On	Off	Even
BB	800	On	Off	Odd

RECOVER Statement

The RECOVER statement identifies: (1) the direct access volume that contains the defective track, (2) the defective track, and (3) a receiving tape. A RECOVER statement is required for each defective track from which data is to be recovered. The RECOVER statement must precede any associated LIST or INSERT statements when IBCRCVRP is used to recover data.

The format of the RECOVER statement is:

```
[label] RECOVER  FROMDEV = xxxx
                 ,FROMADDR = cuu
                 ,TODEV = xxxx
                 ,TOADDR = cuu
                 ,VOLID = serial
                 ,TRACK = bbbbccchhhh
                 [,MODE = mm]
```

where:

FROMDEV = *xxxx*
specifies the type of direct access device that contains the defective track, for example, 2311.

FROMADDR = *cuu*
specifies the channel number, *c*, and unit number, *uu*, of the direct access device that contains the defective track.

TODEV = *xxxx*
specifies the type of the receiving tape volume. If is not specified, the data is written at the highest density supported by the device.

TOADDR = *cuu*
specifies the channel number, *c*, and unit number, *uu*, of the receiving tape volume. This tape must be different from other receiving tapes in the same job. If this volume has no label or a if this volume is unlabeled or non standard labeled, the RECOVER routine writes a tape mark preceding the data.

VOLID = *serial*
specifies the volume serial number of the direct access volume that contains the defective track. If *serial* does not match the volume serial number found on the specified volume, the operator is notified and the job is terminated.

TRACK = *bbbbccchhhh*
specifies the hexadecimal bin, cylinder, and head addresses of the defective track. If the specified track is one for which an alternate has been assigned, data is recovered from the alternate, and a message identifying both tracks is issued.

MODE = *mm*
specifies the bit density for data written to the receiving tape volume. This parameter is applicable to 7-track tape drives and to 9-track tape drives with density selections of 800 and 1600 bits per inch. Valid 7-track modes are shown earlier in Table 7. (Only those modes that set the data converter on are accepted.) For 7-track tape, the default mode is 93. For 9-track tape drives with 800 and 1600 bits per inch density selections, the mode settings are CB and C3, respectively. If no mode is specified, the data is written at the highest density supported by the device.

REPLACE Statement

The REPLACE statement identifies both the tape device containing recovered data (recover tape) and the direct access volume on which recovered data is merged with new replacement data.

The format of the REPLACE statement is:

```
[label] REPLACE  FROMDEV = xxxx
                ,FROMADDR = cuu
                ,TODEV = xxxx
                ,TOADDR = cuu
                ,VOLID = serial
                ,TRACK = bbbbccchhhh
                [,MODE = mm]
```

where:

FROMDEV = *xxxx*
specifies the type of the device on which the recover tape is mounted, for example, 2400. If **MODE** is not specified in this statement, it is assumed that the recover tape was written at maximum density.

FROMADDR = *cuu*
specifies the channel number, *c*, and the unit number *uu* of the tape device on which the recover tape is mounted.

TODEV = *xxxx*
specifies the device type of the direct access device on which recovered data is to be merged with replacement data.

TOADDR = *cuu*
specifies the channel number, *c*, and unit number, *uu*, of the direct access device on which recovered data is to be merged with replacement data.

VOLID = *serial*
specifies the volume serial number of the direct access volume on which recovered data is to be merged with replacement data.

TRACK = *bbbbccchhhh*
specifies the hexadecimal bin, cylinder, and head addresses of the defective primary track from which data was recovered.

MODE = *mm*
specifies the bit density at which data was written onto the source magnetic tape volume. **MODE** should not be specified if it was not specified when data was written onto the source volume. Valid 7-track modes are shown earlier in Table 7. (Only those modes that set the data converter on are accepted.) For 9-track tape drives with 800 and 1600 bits per inch density selections, the mode settings are CB and C3, respectively.

LIST Statement

The LIST statement specifies that the entire contents of a defective track be printed when data is being recovered; it specifies that both recovered data and replacement records are to be listed after they are merged when data is being replaced. If the LIST statement is omitted, only bad records are printed when data is being recovered; only replacement records are listed when data is being replaced.

The format of the LIST statement is:

```
[label] LIST  TODEV = xxxx
              ,TOADDR = cuu
              [,MODE = nn]
```

where:

TODEV = *xxxx*
specifies the type of the list device, for example, 1403.

TOADDR = *cuu*
specifies the channel number, *c*, and unit number, *uu*, of the list device.

MODE = *mm*
specifies the mode in which the list tape is to be written when the list device is 7-track tape. Valid modes are shown earlier in Table 7. If **MODE** is not specified and the list device is different from the message output device, MODE = 93 is assumed. For 9-track tape drives with 800 and 1600 bits per inch density selections, the mode settings are CB and C3, respectively.

Tape volumes must have either a standard label or a tape mark in place of a label. The label or tape mark must be written in the same mode as the data.

If the list device and the message output device are the same, the list mode will be the same as the message mode.

Neither the list device nor the message output device can be the same as the tape device containing recovered data.

INSERT Statement

The INSERT statement identifies the device that contains each replacement record and describes the count field of that record. INSERT statements and corresponding data must be in sequence by record number (for example, if records 3 and 5 are bad, the INSERT statement and replacement data for record 3 must precede the INSERT statement and data for record 5).

The format of the INSERT statement is:

```
[label] INSERT  [FROMDEV = xxxx]
                [,FROMADDR = cuu]
                {,RECORD = nnn  }
                {,RECORD = LAST }
                ,COUNT = ccchhhhrrkkdddd
                [,MODE = mm]
                [,OVERFLOW = YES]
```

where:

FROMDEV = *xxxx*
specifies the device type of the device that contains replacement data. **FROMDEV** may be omitted if the bad record did not contain key or data fields.

FROMADDR = *cuu*
specifies the channel number, *c*, and unit number, *uu*, of the device that contains replacement data. **FROMADDR** may be omitted if the bad record did not contain key or data fields.

RECORD = *nnn*
indicates the decimal record number of the original bad record. (This number is obtained from message IBC305I.)

RECORD = LAST
specifies that this replacement record is to be the last physical record written on the alternate track. Records can be added after this record if the track capacity is not exceeded. With this feature, records near the end of a defective track that has missing address markers (and, thus, could not be recovered) can still be replaced.

COUNT = *ccchhhhrrkkdddd*
specifies in hexadecimal the count field for the replacement record, where *cccc* is the cylinder number, *hhhh* is the head number, *rr* is the physical record number, *kk* is the key length, in bytes, and *dddd* is the data length (excluding the key length), in bytes.

MODE = *mm*
specifies the mode in which the input tape was written when the replacement data is on 7-track tape. Valid modes are shown earlier in Table 7. This tape volume must have either a standard label or a tape mark in place of a label. The label or tape mark must be written in the same mode as the data. If **MODE** is omitted, 93 is assumed. For 9-track tape drives with 800 and 1600 bits per inch density selections, the mode settings are CB and C3, respectively.

OVERFLOW = YES
specifies that the bad record, which is being replaced, was a segment, other than the last segment, of an overflow record. The replacement record will be either the last record or the only record on the assigned alternate track. Six lines per inch are to be printed. Channel 1 is assigned to line 4, channel 2 is assigned to line 10, channel 3 is assigned to line 16, etc.

END Statement

The END Statement denotes the end of job. It appears after the last function definition statement.

The format of the END statement is:

[*label*] **END** [*user-information*]

END must be preceded and followed by at least one blank.

IBCRCVRP Examples

The examples that follow illustrate some uses of IBCRCVRP. Table 8 can be used as a quick reference guide to IBCRCVRP examples. The numbers in the "Example" column point to the examples that follow.

Table 8. IBCRCVRP Example Directory

Operation	*Comments*	*Example*
RECOVER	Data is to be recovered from defective tracks on 2314 volumes.	1
REPLACE	Bad data on a 2314 volume is to be replaced.	2

IBCRCVRP Example 1

In this example data is recovered from defective tracks on 2314 volumes 123456 and 222222. The entire contents of these tracks are listed on a 1403 printer—channel 0, unit 0E. Note that column 1 is blank.

The example follows:

```
JOB       'RECOVER 2314 TRACKS'
MSG       TODEV=1403,TOADDR=00E
RECOVER   FROMDEV=2314,FROMADDR=190,TODEV=2400,                ¢C
             VOLID=123456,TRACK=0000005E0008,TOADDR=280
LIST      TODEV=1403,TOADDR=00E
RECOVER   FROMDEV=2314,FROMADDR=191,TODEV=2400,                ¢C
             TOADDR=281,VOLID=222222,TRACK=000000110005
LIST      TODEV=1403,TOADDR=00E
END
```

IBCRCVRP Example 2

In this example, bad data is replaced on 2314 volume 123456. Replacement records 001 and 003 contain 32 (20 in hexadecimal) bytes and 40 bytes (including an 8-byte key) of replacement data, respectively. Record 003 must be continued on an additional card image because the replacement data exceeds 35 bytes. Note that column 1 is blank for all but the replacement record statements.

The example follows:

```
 JOB           'REPLACE 2314 TRACK'
  MSG       TODEV=1403,TOADDR=00E
  REPLACE   FROMDEV=2400,FROMADDR=280,TODEV=2311,                      C
               TOADDR=190,VOLID=123456,TRACK=0000005E0008
  LIST      TODEV=1403,TOADDR=00E
  INSERT    FROMDEV=1442,FROMADDR=00C,RECORD=001,                      C
               COUNT=005E000801000020
I/D=001    ABCDEF0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0123456789
  INSERT    FROMDEV=1442,FROMADDR=00C,RECORD=003,                      C
               COUNT=005E000803080020
I/D=003    FFFFFFFFFFFFFFFFDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
          DDDDDDDDDD
 END
```

ICAPRTBL Program

ICAPRTBL is an independent utility used to load the Universal Character Set (UCS) buffer and the forms control buffer (FCB) for an IBM 3211 Printer. (See "Introduction" for general independent utility information.)

ICAPRTBL is used when the 3211 is assigned as the output portion of a composite console and an unsuccessful attempt has been made to initialize the operating system because the UCS and FCB buffers contain improper bit patterns. ICAPRTBL is used to properly load the buffers so the operating system can be initialized.

Note: When an operable console printer-keyboard is available, the buffers are loaded under the control of the operating system.

Input and Output

ICAPRTBL uses as input utility control statements that contain images to be loaded into the Universal Character Set and/or forms control buffer. ICAPRTBL produces as output properly loaded UCS and FCB buffers.

Control

ICAPRTBL is controlled by utility control statements. Because ICAPRTBL is an independent utility, operating system job control statements are not used.

Utility Control Statements

ICAPRTBL utility control statements are:

- JOB statement, which is used to indicate the beginning of an ICAPRTBL job.
- DFN statement, which is used to define the address of the 3211.
- UCS statement, which contains an image of the characters to be loaded into the UCS buffer.
- FCB statement, which defines the image to be loaded into the FCB.
- END statement, which is used to indicate the end of an ICAPRTBL job.

JOB Statement

The JOB statement indicates the beginning of an ICAPRTBL job.

The format of the JOB statement is:

```
[label] JOB [user-information]
```

JOB must be preceded and followed by at least one blank.

DFN Statement

The DFN statement is used to define the address of the 3211 and to specify lowercase letters are to be printed in uppercase when the lowercase print train is not available.

The format of the DFN statement is:

```
DFN ADDR = cuu,FOLD = {Y}
                      {N}
```

where:

ADDR = *cuu*
specifies the channel number, *c*, and unit number, *uu*, of the 3211.

FOLD =
specifies whether lowercase letters are to be printed as uppercase letters when the lowercase print train is not available. These values can be coded:

Y
specifies that lowercase letters are to be printed as uppercase letters when the lowercase print train is not available.

N
specifies that lowercase letters are not to be printed as uppercase letters.

UCS Statement

The UCS statement contains an image to be loaded into the UCS buffer.

The format of the UCS statement is:

```
ucsname UCS      ucs-image
```

where:

ucsname
is a one- to eight-character alphameric name. This name is printed on the printer to serve as a reference to the print train being used.

ucs-image
specifies characters to be loaded into the UCS buffer. The characters must be contained in columns 16 through 71. The first UCS statement contains the first 56 characters; subsequent statements contain continuations of the image to be loaded into the UCS buffer.

FCB Statement

The FCB statement defines the image to be loaded into the forms control buffer. The FCB statement may precede or follow the UCS statement.

The format of the FCB statement is:

```
[fcbname] FCB  LPI = {6 }
                     {8 }
                  ,LNCH = ((l,c)[,(l,c,)...])
                  ,FORMEND = x
```

where:

fcbname
specifies a one- to eight-character name of the image loaded into the forms control buffer. The actual image loaded into the buffer is not affected by this name, but to serve as a meaningful reference when printed on the printer, *fcbname* should be the same as the FCB image being used.

LPI =
specifies the number of lines per inch that will be printed on the document. These values can be coded:

6
specifies that six lines per inch will be printed.

8
specifies that eight lines per inch will be printed.

LNCH =
specifies the channels of the FCB image. Each set of parentheses must contain the line number (1-280), a comma, and the channel number (1-12) to be assigned to that line. One or all of the 12 channels may be assigned in any order. Each set must be separated by commas and the entire group surrounded by parentheses.

FORMEND = *x*
specifies the number of lines (maximum 180) on the printer form. For an 11 inch form, spacing six lines per inch, *x* must be 66.

END Statement

The END statement signals the end of the ICAPRTBL job.

The format of the END statement is:

```
[label] END [user-information]
```

END must be preceded and followed by at least one blank.

ICAPRTBL Example

In this example, an A11 UCS image and an FCB image are loaded into the UCS and FCB buffers.

The example follows:

```
        JOB LOAD A11 IMAGE
        DFN ADDR=002,FCLD=N
A11     UCS       1<.=IHGFEDCBA*$-RQPONMLKJ%,&ZYXWVUTS/@#0987654321<.=IHGF
                  EDCBA*$-RQPONMLKJ%,&ZYXWVUTS/@#0987654321<.=IHGFEDCBA*$-
                  RQPONMLKJ%,&ZYXWVUTS/@#0987654321<.=IHGFEDCBA*$-RQPONMLK
                  J%,&ZYXWVUTS/@#0987654321<.=IHGFEDCBA*$-RQPONMLKJ%,&ZYXW
                  VUTS/@#0987654321<.=IHGFEDCBA*$-RQPONMLKJ%,&ZYXWVUTS/@#0
                  987654321<.=ihgfedcba*$-rqponmlkj%,&zyxwvuts/23098765432
                  1<.=IHGFEDCBA*$-RQPONMLKJ%,&ZYXWVUTS/@#0987654321<.=IHGH
                  EDCBA*$-RQPONMLKJ%,&ZYXWVUTS/@#098765432
STD2 FCB LPI=6,                                                          C
         LNCH=((4,1),(10,2),(16,3),(22,4),(28,5),(34,6),(40,7),          C
         (46,8),(52,10),(58,11),(64,12),(66,9))
         END
```

The control statements are discussed below:

- DFN specifies the channel and unit number of the 3211 and specifies that lowercase letters are not to be printed as uppercase letters when the lowercase print train is not available.
- UCS specifies the characters to be loaded into the UCS buffer.
- FCB specifies the values to be loaded into the forms control buffer.

IEBCOMPR is a data set utility used to compare two sequentially organized or two partitioned data sets at the logical record level to verify a backup copy. Fixed, variable, or undefined records from blocked or unblocked data sets or members can be compared. (See "Introduction" for general data set utility information.)

Two sequential data sets are considered *equal*, that is, are considered to be identical, if:

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

If these conditions are not met, an unequal comparison results. 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 terminate the job step unless a user routine is provided to handle error conditions.

Two partitioned data sets are considered equal if:

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

If these conditions are not met, an unequal comparison results. 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. After ten successive unequal comparisons, processing continues with the next member unless a user routine is provided to handle error conditions.

Partitioned data sets 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 identified by these entries and corresponding user data.

Figure 3 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.

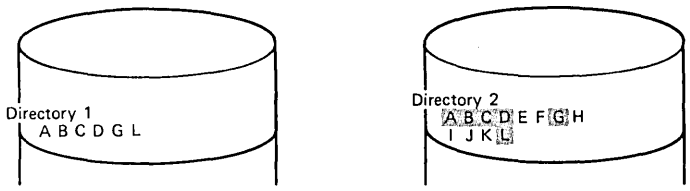


Figure 3. Partitioned Directories Whose Data Sets Can Be Compared Using IEBCOMPR

Figure 4 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 is terminated.

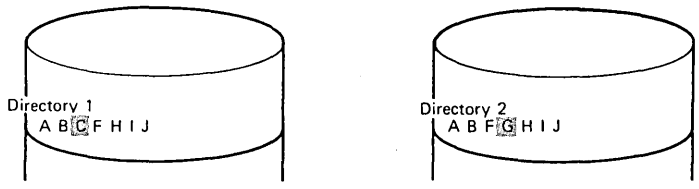


Figure 4. Partitioned Directories Whose Data Sets Cannot Be Compared Using IEBCOMPR

User exits are provided for optional user routines to process user labels, handle error conditions, and modify source records. See "Appendix A: Exit Routine Linkage" for a discussion of the linkage conventions to be followed when user routines are used.

At the completion or termination of IEBCOMPR, the highest return code encountered within the program is passed to the calling program.

Input and Output

IEBCOMPR uses the following input:

- Two sequential or two partitioned data sets 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 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.

IEBCOMPR provides a return code to indicate the results of program execution. The return codes and their meanings are:

- 00, which indicates successful completion.
- 08, which indicates an unequal comparison. Processing continues.
- 12, which indicates an unrecoverable error. The job step is terminated.
- 16, which indicates that a user routine passed a return code of 16 to IEBCOMPR. The job step is terminated.

Control

IEBCOMPR is controlled by job control statements and utility control statements. The job control statements are required to execute or invoke 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 or partitioned), to identify any user routines that may be provided, and to indicate whether user labels are to be treated as data.

Job Control Statements

Table 9 shows the job control statements necessary for using IEBCOMPR.

Table 9. IEBCOMPR Job Control Statements

Statement	*Use*
JOB	Initiates 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 message data set, which can be written to a system output device, a tape volume, or a direct access volume.
SYSUT1 DD	Defines an input data set to be compared.
SYSUT2 DD	Defines an input data set 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 provided. The control data set normally resides in the input stream; however, it can be defined as a member within a library of partitioned members.

The minimum region size that can be specified for IEBCOMPR is 14K + 2b, where b is the largest block size in the job step, rounded to the next higher 2K.

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

Input data sets residing on different device types can be compared. Input data sets with a sequential organization written at different densities can be compared.

Restrictions

- The SYSPRINT DD statement must be present for each use of IEBCOMPR.
- The SYSIN DD statement is required.
- The logical record lengths of the input data sets must be identical; otherwise, unequal comparisons result. The block sizes of the input data sets can differ; however, block sizes must be multiples of the logical record length.
- The block size specified in the SYSPRINT DD statement must be a multiple of 121. The block size specified in the SYSIN DD statement must be a multiple of 80.
- When the input/output data set has fixed length, variable length, or variable spanned records, the BLKSIZE, RECFM, and LRECL are required. When the data set has undefined length records, only BLKSIZE is required.

Utility Control Statements

The utility control statements used to control IEBCOMPR are:

- COMPARE statement, which is used to indicate the organization of a data set.
- EXITS statement, which is used to identify user exit routines to be used.
- LABELS statement, which is used to indicate whether user labels are to be treated as data by IEBCOMPR.

COMPARE Statement

The COMPARE statement is used to indicate the organization of data sets to be compared.

The format of the COMPARE statement is:

```
[label] COMPARE [TYPORG = {PS}
                          {PO}]
```

where:

TYPORG =
specifies the organization of the input data sets. If **TYPORG** is omitted, input data sets are assumed to be sequentially organized. The values that can be coded are:

PS
specifies that the input data sets are sequential data sets. If nothing is specified, PS is assumed.

PO
specifies that the input data sets are partitioned data sets.

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

EXITS Statement

The EXITS statement is used to identify any user exit routines to be used.

The format of the EXITS statement is:

```
[label] EXITS  [INHDR = routinename]
               [,INTLR = routinename]
               [,ERROR = routinename]
               [,PRECOMP = routinename]
```

where:

INHDR = *routinename*
specifies the symbolic name of a routine that processes user input header labels.

INTLR = *routinename*
specifies the symbolic name of a routine that processes user input trailer labels.

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

PRECOMP = *routinename*
specifies the symbolic name of a routine that processes logical records (physical blocks in the case of VS or VBS records longer than 32K bytes) from either or both of the input data sets before they are compared.

The EXITS statement is required if a user exit routine is to be used. If more than one valid EXITS statement is included, all but the last EXITS statement are ignored. For a discussion of the processing of user labels as data set descriptors, see "Appendix E: Processing User Labels."

LABELS Statement

The LABELS statement specifies whether user labels are to be treated as data by IEBCOMPR. For a discussion of this option, refer to "Processing User Labels as Data" in "Appendix E: Processing User Labels."

The format of the LABELS statement is:

```
[label] LABELS [DATA = {YES  }
                       {NO   }
                       {ALL  }
                       {ONLY }]
```

where:

DATA =
specifies whether user labels 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. If no value is entered, **YES** is assumed.

NO
specifies that user labels are not to be treated as data.

ALL
specifies that user labels are to be treated as data regardless of any return code. A return code of 16 causes IEBCOMPR to complete processing of the remainder of the group of user labels and to terminate 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 terminates upon return from the OPEN routine.

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

If more than one valid LABELS statement is included, all but the last LABELS statement is ignored.

IEBCOMPR Examples

The examples that follow illustrate some of the uses of IEBCOMPR. Table 10 can be used as a quick reference guide to IEBCOMPR examples. The numbers in the "Example" column point to examples that follow.

Table 10. IEBCOMPR Example Directory

Operation	*Data Set Organization*	*Devices*	*Comments*	*Example*
COMPARE	Sequential	9-track tape	No user routines. Blocked input.	1
COMPARE	Sequential	7-track tape	No user routines. Blocked input.	2
COMPARE	Sequential	7-track and 9-track tape	User routines. Blocked input. Different density tapes.	3
COMPARE	Sequential	Card Reader, 9-track tape	No user routines. Blocked input.	4
COMPARE	Partitioned	2314 Disk	No user routines. Blocked input.	5
COPY (using IEBGENER) and COMPARE	Sequential	9-track tape	No user routines. Blocked input. Two job steps; data sets are passed to second job step.	6
COPY (using IEBCOPY) and COMPARE	Partitioned	2311 Disk	User routine. Blocked input. Two job steps; data sets are passed to second job step.	7

IEBCOMPR Example 1

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

The example follows:

```
//TAPETAPE JOB  09#660,SMITH
//         EXEC PGM=IEBCOMPR
//SYSPRINT DD   SYSOUT=A
//SYSUT1   DD   UNIT=2400,LABEL=(,NL),
//              DCB=(RECFM=FB,LRECL=80,BLKSIZE=2000),
//              DISP=(OLD,KEEP),VOLUME=SER=001234
//SYSUT2   DD   UNIT=2400,LABEL=(,NL),DISP=(OLD,KEEP),
//              DCB=(RECFM=FB,LRECL=80,BLKSIZE=1040),
//              VOLUME=SER=001235
//SYSIN    DD   DUMMY
/*
```

Because no user routines are to be used and the input data sets have a sequential organization, utility control statements are not used.

The control statements are discussed below:

- SYSUT1 DD defines an input data set, which resides on an unlabeled, 9-track tape volume. The blocked data set was originally written at 800 bits per inch density.
- SYSUT2 DD defines an input data set, which resides on an unlabeled, 9-track tape volume. The blocked data set was originally written at 800 bits per inch density.
- SYSIN DD defines a dummy data set.

IEBCOMPR Example 2

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

The example follows:

```
//TAPETAPE JOB  09#660,SMITH
//         EXEC PGM=IEBCOMPR
//SYSPRINT DD   SYSOUT=A
//SYSUT1   DD   DSNAME=SET1,LABEL=(2,SUL),DISP=(OLD,KEEP),
//              VOL=SER=001234,DCB=(DEN=2,RECFM=FB,LRECL=80,
//              BLKSIZE=2000,TRTCH=C),UNIT=2400-2
//SYSUT2   DD   DSNAME=SET1,LABEL=(,SUL),DISP=(OLD,KEEP),
//              VOL=SER=001235,DCB=(DEN=2,RECFM=FB,LRECL=80,
//              BLKSIZE=2000,TRTCH=C),UNIT=2400-2
//SYSIN    DD   *
 COMPARE  TYPORG=PS
 LABELS   DATA=ONLY
/*
```

The control statements are discussed below:

- SYSUT1 DD defines an input data set, which resides on a labeled, 7-track tape volume. The blocked data set was originally written at 800 bits per inch density with the data converter on.
- SYSUT2 DD defines an input data set, which is the first or only data set on a labeled, 7-track tape volume. The blocked data set was originally written at 800 bits per inch density with the data converter on.
- SYSIN DD defines the control data set, which follows in the input stream.
- COMPARE specifies that the input data sets are sequentially organized.
- LABELS specifies that only user header labels are to be compared.

IEBCOMPR Example 3

In this example, two sequential data sets written at different densities on different device types are to be compared.

The example follows:

```
//TAPETAPE JOB  09#660,SMITH
//         EXEC PGM=IEBCOMPR
//SYSPRINT DD   SYSOUT=A
//SYSUT1   DD   DSNAME=SET1,LABEL=(,SUL),DISP=(OLD,KEEP),
//              VOL=SER=001234,DCB=(DEN=1,RECFM=FB,LRECL=80,
//              BLKSIZE=320,TRTCH=C),UNIT=2400-2
//SYSUT2   DD   DSNAME=SET2,LABEL=(,SUL),DISP=(OLD,KEEP),
//              DCB=(RECFM=FB,LRECL=80,BLKSIZE=640),
//              VOLUME=SER=001235,UNIT=2400
//SYSIN    DD   *
 COMPARE  TYPORG=PS
 EXITS    INHDR=HDRS,INTLR=TLRS
 LABELS   DATA=NO
/*
```

The control statements are discussed below:

- SYSUT1 DD defines an input data set, which is the first or only data set on a labeled, 7-track tape volume. The blocked data set was originally written at 556 bits per inch density with the data converter on.
- SYSUT2 DD defines an input data set, which is the first or only data set on a labeled, 9-track tape volume. The blocked data set was originally written at 800 bits per inch density.
- SYSIN DD defines the control data set, which follows in the input stream.
- COMPARE 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 specifies that the user input header and trailer labels are not to be compared.

IEBCOMPR Example 4

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

The example follows:

```
//CARDTAPE JOB  09#660,SMITH
//         EXEC PGM=IEBCOMPR
//SYSPRINT DD   SYSOUT=A
//SYSIN    DD   DUMMY
//SYSUT2   DD   UNIT=2400,VOLUME=SER=001234,LABEL=(,NL),
//              DCB=(RECFM=FB,LRECL=80,BLKSIZE=2000),
//              DISP=(OLD,KEEP)
//SYSUT1   DD   DATA
```

(input card data set)

```
/*
```

The control statements are discussed below:

- 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 used.
- SYSUT2 DD defines an input data set, which resides on an unlabeled, 9-track tape volume. The blocked data set was originally written at 800 bits per inch density.
- SYSUT1 DD defines an input data set (card input).

IEBCOMPR Example 5

In this example, two partitioned data sets are to be compared.

The example follows:

```
//DISKDISK JOB  09#660,SMITH
//         EXEC PGM=IEBCOMPR
//SYSPRINT DD   SYSOUT=A
//SYSUT1   DD   DSNAME=PDSSET,UNIT=2314,DISP=(OLD,KEEP),
//              DCB=(RECFM=FB,LRECL=80,BLKSIZE=2000),
//              VOLUME=SER=111112
//SYSUT2   DD   DSNAME=PDSSET,UNIT=2314,DISP=(OLD,KEEP),
//              DCB=(RECFM=FB,LRECL=80,BLKSIZE=2000),
//              VOLUME=SER=111113
//SYSIN    DD   *
     COMPARE   TYPORG=PO
/*
```

The control statements are discussed below:

- SYSUT1 DD defines an input partitioned data set. The blocked data set resides on a 2314 volume.
- SYSUT2 DD defines an input partitioned data set. The blocked data set resides on a 2314 volume.
- SYSIN DD defines the control data set, which follows in the input stream. The data set consists of one utility control statement.

IEBCOMPR Example 6

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

The example follows:

```
//TAPETAPE JOB 09#660,SMITH
//STEPA  EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=A
//SYSUT1   DD DSN=COPYSET,UNIT=2400,DISP=(OLD,PASS),
//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=640),
//            LABEL=(,SL),VOLUME=SER=001234
//SYSUT2   DD DSNAME=COPYSET,DISP=(,PASS),LABEL=(,SL),
//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=640),
//            VOLUME=SER=001235,UNIT=2400
//SYSIN    DD DUMMY
/*
//STEPB    EXEC PGM=IEBCOMPR
//SYSPRINT DD SYSOUT=A
//SYSUT1   DD DSNAME=*.STEPA.SYSUT1,DISP=(OLD,KEEP)
//SYSUT2   DD DSNAME=*.STEPA.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 below:

- SYSUT1 DD defines an input data set passed from the preceding job step. The data set resides on a labeled, 9-track tape volume. The blocked data set was originally written at 800 bits per inch density.
- SYSUT2 DD defines an input data set passed from the preceding job step. The data set, which was created in the preceding job step, resides on a labeled, 9-track tape volume. The blocked data set was originally written at 800 bits per inch density.
- 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.

IEBCOMPR Example 7

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

The example follows:

```
//DISKDISK JOB 09#660,SMITH
//STEPA    EXEC PGM=IEBCOPY
//SYSPRINT DD SYSOUT=A
//SYSUT1   DD DSNAME=OLDSET,UNIT=2311,DISP=(OLD,PASS),
//            VOLUME=SER=111112,DCB=(RECFM=FB,LRECL=80,
//            BLKSIZE=640)
//SYSUT2   DD DSNAME=NEWMEMS,UNIT=2311,DISP=(,PASS),
//            VOLUME=SER=111113,SPACE=(TRK,(10,5,5)),
//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=640)
//SYSUT3   DD UNIT=2311,SPACE=(TRK,(1))
//SYSUT4   DD UNIT=2311,SPACE=(TRK,(1))
//SYSIN    DD *
 COPY OUTDD=SYSUT2,INDD=SYSUT1
 SELECT MEMBER=(A,B,D,E,F)
/*
//STEPB    EXEC PGM=IEBCOMPR
//SYSPRINT DD SYSOUT=A
//SYSUT1   DD DSNAME=OLDSET,DISP=(OLD,KEEP)
//SYSUT2   DD DSNAME=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 below:

- SYSUT1 DD defines a blocked input data set that is passed from the preceding job step. The data set resides on a 2311 volume.
- SYSUT2 DD defines a blocked input data set that is passed from the preceding job step. The data set resides on a 2311 volume.
- SYSIN DD defines the control data set, which contains a COMPARE statement and an EXITS statement.
- COMPARE specifies partitioned organization.
- EXITS specifies that a user 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.

IEBCOPY Program

IEBCOPY[1] is a data set utility used to copy one or more partitioned data sets or to merge partitioned data sets. (See "Introduction" for general data set utility information.) Specified members of partitioned data sets can be selected for, or excluded from, a copy process.

IEBCOPY can be used to:

- Create a backup copy.
- Copy data sets.
- Select members, from one or more data sets, to be copied.
- Replace identically named members on data sets.
- Replace selected data set members.
- Rename selected members.
- Exclude members, from one or more data sets, from being copied.
- Compress a data set in place.
- Merge data sets.
- Re-create a data set that has exhausted its primary, secondary, or directory space allocation.

In addition, IEBCOPY automatically lists the number of unused directory blocks and the number of unused tracks available for member records in the output partitioned data set. The names of copied members can be listed by input partitioned data set.

When copying members that have aliases, the following should be noted:

- When the main member and its aliases are copied, they exist on the output partitioned data set in the same relationship they had on the input partitioned data set.
- When one alias is copied without its main member, it becomes a main member.
- When two or more aliases are copied without the main member, the lowest alias (in alphameric collating sequence) becomes the main member; any remaining aliases become aliases of the *new* main member. Note that if an *old* main member name is present in an alias entry, it remains there.

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

At the completion or termination of the program, the highest return code encountered within the program is passed to the calling program.

Creating a Backup Copy

IEBCOPY can be used to copy a partitioned data set, totally or in part, 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, the data set is compressed in place.

Note: The copied members are not reordered; that is, they are copied in the order in which they exist on the original data set. If the members are to be collated, IEHMOVE can be used for the copy operation. See the chapter "IEHMOVE Program" for a discussion of the IEHMOVE program.

Copying Data Sets

IEBCOPY can be used to copy more than one input partitioned data set, totally or in part, from one or more direct access volumes to a single direct access volume. See "COPY Statement" below for a discussion of how to specify more than one input partitioned data set. The input partitioned data sets are copied in the order in which they are specified.

---

[1] This is a description of the version of IEBCOPY available on Release 21.7 of IBM System/360 Operating System. The program is designed to accept the job and control statements written for the version available on releases prior to Release 20. However, it is recommended that any future user applications be written to the specifications discussed in this chapter.

Selecting Members to be Copied

Members can be selected from one or more input partitioned data sets. Selected members are searched for in a low-to-high (a to z) collating sequence, regardless of the order in which they are specified; however, they are copied in the same physical sequence in which they appear on the input partitioned data set.

When selecting members from an input partitioned data set, remember that once a member is found it is not searched for on any subsequent input partitioned data set. Similarly, when all of the selected members are found, the copy step is terminated although all of the input partitioned data sets may not have been searched. For example, if members A and B are specified and A is found on the first of three input partitioned data sets, it is not searched for again; if B is found on the second input partitioned data set, the copy operation is successfully terminated after the second input partitioned data set has been processed, although both A and B may also exist on the third input partitioned data set.

However, if the first member name is not found on the first input partitioned data set, the second selected member is searched for; if it is not found, the third is searched for, and so on. This process continues until there are no more members to be searched for in this input partitioned data set. All the members that were found on the input partitioned data set are then processed for copying onto the output partitioned data set. This process is repeated for the second input partitioned data set (except that the members that were found on the first input partitioned data set are not searched for again).

Replacing Identically Named Members

In many copy operations, the output partitioned data set may contain members that have names identical to the names of the input partitioned data set members to be copied. When this occurs, the user may specify that the identically named members are to be copied from the input partitioned data set to replace existing members. The replace option allows an input member to override an existing member on the output partitioned data set with the same name.

If the replace option is not specified, input members are not copied when they have the same name as a member on the output partitioned data set.

The replace option can be specified on the data set or member level. The level is specified on a utility control statement.

When replace is specified on the data set (specified on a COPY or on the INDD statement), the input data is copied as follows:

- In a full copy process, all members on an input partitioned data set are copied onto an output partitioned data set; members whose names already exist on the output partitioned data set are replaced by the members copied from the input partitioned data set.
- In a selective copy process, all selected members on an input partitioned data set are copied to an output partitioned data set; all selected members *found* are copied and members whose names already exist on the output partitioned data set are replaced by the *found* members copied from the input partitioned data set.
- In an exclusive copy process, all nonexcluded members on input partitioned data sets are copied to an output partitioned data set; nonexcluded input members whose names already exist on the output partitioned data set replace those identically named members on the output partitioned data set.

When replace is specified on the member level (specified on a SELECT statement), only selected members on the input partitioned data sets are copied, and identically named members on the output partitioned data set are replaced.

Differences between full, selective, and exclusive copy processing should be remembered when specifying the replace option when multiple data sets contain member names common to some or all of the input partitioned data sets being copied. These differences are:

- When a full copy is performed, the output partitioned data set contains the replacing members that were on the last input partitioned data set copied.
- When a selective copy is performed, the output partitioned data set contains the selected replacing members which were *found* on the earliest input partitioned data set searched. Once a selected member is found, it is not searched for again; therefore, once found, a selected member is copied, and if the same member exists on another input partitioned data set it is not searched for, and hence, not copied.
- When an exclusive copy is performed, the output partitioned data set contains the nonexcluded replacing members that were on the last input partitioned data set copied.

Replacing Selected Members

The user may specify the replace option on either the data set or the member level when members are being selected for copying.

If the replace option is specified on the data set level, all selected members found on the designated input partitioned data sets replace identically named members on the output partitioned data set. This is limited by the fact that once a selected member is found it is not searched for again.

If the replace option is specified on the member level, the specified members on the input partitioned data set replace identically named members on the output partitioned data set. Once a member is found it is not searched for again. (See "Replacing Identically Named Members" earlier in this chapter.)

Renaming Selected Members

Selected members on input partitioned data sets can be copied and renamed on the output partitioned data set. However, if the new name is identical to a member name on the output partitioned data set, the input member is not copied unless the replace option is also specified. See "SELECT Statement" below for information on renaming selected members.

Note: Renaming is not physically done to the input partitioned data set directory entry. However, after the member is copied onto the output partitioned data set, the new name is entered into the output partitioned data set directory.

Excluding Members from a Copy Operation

Members from one or more input partitioned data sets can be excluded from a copy operation. The excluded member is searched for on every input partitioned data set in the copy operation and is always omitted from the copy.

The replace option can be specified on the data set level in an exclusive copy, in which case, nonexcluded members on the input partitioned data set replace identically named members on the output partitioned data set. See "Replacing Identically Named Members" earlier in this chapter for more information on the replace option.

Compressing a Data Set

A compressed data set is one that does not contain embedded unused space. After copying one or more input partitioned data sets to a *new* output partitioned data set (by means of a selective, exclusive, or full copy that does not involve replacing members), the output partitioned data set contains no embedded unused space.

To make unused space available, either the entire data set must be scratched or it must be compressed in place. A compressed version can be created by specifying the same data set for both the input and the output parameters in a full copy step. A backup copy of the partitioned data set to be compressed in place should be kept until successful completion of an in-place compression is indicated (by an end-of-job message and a return code of 00).

Note: An in-place compression does not release extents assigned to the data set.

Merging Data Sets

A merged data set is one to which an additional member is copied. It is created by copying the additional members to an existing output partitioned data set; the merge operation—the ordering of the output partitioned data set's directory—is automatically performed by IEBCOPY.

Note: If there is a question about whether or not enough directory blocks are allocated to the output partitioned data set to which an input partitioned data set is being merged, the output partitioned data set should be *re-created* prior to the merge operation.

Re-creating a Data Set

A data set can be recreated by copying it and allocating a larger amount of space than was allocated for the original data set. This application of IEBCOPY is especially useful if insufficient directory space was allocated to a data set. Space cannot be allocated in this manner for an existing data set into which members are being merged.

Input and Output

IEBCOPY uses the following input:

- An input data set, which contains the members to be copied or merged into a partitioned data set.
- A control data set, which contains utility control statements. The control data set is required if selected members are to be copied, merged into a partitioned data set, or omitted from the copy or merge operation.

If the control data set is null, a full copy is attempted from the input partitioned data set to the output partitioned data set. In this case, SYSUT1 and SYSUT2 are required ddnames for the input partitioned data set and output partitioned data set, described under "Job Control Statements" below, respectively.

Note: When merging into or compressing libraries, do not specify DISP = SHR. The results of a merge into or compress of the current SYS1.LINKLIB or SYS1.SVCLIB would be unpredictable.

IEBCOPY produces the following output:

- An output data set, which contains the copied or merged data. The output data set is either a new data set (from a copy operation) or an old data set (from a merge or compress-in-place).
- A message data set, which contains informational messages (for example, the names of copied members) and error messages, if applicable.
- Spill data sets, which are temporary data sets used to provide space when not enough main storage is available for the input and/or output partitioned data set directories. These data sets are opened only when needed.

All input, output, and utility data sets must be on direct access devices. The following devices may be used:

- 2311 Disk Storage Drive
- 2314 Direct Access Storage Facility
- 2319 Direct Access Storage Facility
- 2301 Disk Storage
- 2302 Drum Storage
- 2303 Drum Storage
- 2305 Fixed Head Storage
- 2321 Data Cell Drive
- 3330 Disk Storage

Any combination of these devices is acceptable to IEBCOPY.

Note: Refer to *OS Storage Estimates*, GC28-6551, to determine when spill data sets are required; see "Space Allocation" below for a description of how to determine the amount of space to allocate.)

IEBCOPY produces a return code to indicate the results of program execution. The return codes and their meanings are:

- 00, which indicates successful completion.
- 04, which indicates a condition from which recovery may be possible.
- 08, which indicates an unrecoverable error. The job step is terminated.

Control

IEBCOPY is controlled by job control statements and utility control statements.

Job Control Statements

Table 11 shows the job control statements necessary for using IEBCOPY.

The minimum region size that can be specified for IEBCOPY is 28K + 2b, where b is the largest block size in the job step, rounded to the next higher 2K. For additional information, see *OS Storage Estimates*, GC28-6551.

Fixed or variable records can be reblocked. Reblocking or deblocking is done if the block size of the input partitioned data set is not equal to the block size of the output partitioned data set. Reblocking or deblocking cannot be done 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. (Earlier versions allowed reblocking or deblocking with track overflow output records.)

Table 12 shows how input record formats can be changed. In addition, any record format can be changed to the undefined format (in terms of its description in the DSCB).

System data sets should not be compressed in place in a multiprogramming environment unless the subject partitioned data set is made *non-sharable*. The libraries in which IEBCOPY resides (SYS1.LINKLIB and SYS1.SVCLIB) must not be compressed by IEBCOPY unless IEBCOPY is first transferred to a JOBLIB.

Refer to *OS Data Management Services Guide*, GC26-3746, for information on estimating space allocations.

Table 11. IEBCOPY Job Control Statements

Statement	*Use*
JOB	Initiates the job.
EXEC	Specifies the program name (PGM = IEBCOPY) or, if the job control statements reside in the procedure library, the procedure name.
SYSPRINT DD	Defines the sequential message data set used for listing statements and messages. This data set can be written onto a system output device, a tape volume, or a direct access volume.
anyname1 DD	Defines an input partitioned data set. The data set can be defined by a data set name, as a cataloged data set, or as a data set passed from a previous job step.
anyname2 DD	Defines an output partitioned data set.
SYSUT3 DD	Defines a spill data set on a direct access device. SYSUT3 is used when there is no space in main storage for some or all of the current input partitioned data set's directory entries. SYSUT3 may also be used when not enough space is available in main storage for retaining information during table sorting.
SYSUT4 DD	Defines a spill data set on a direct access device. SYSUT4 is used when there is no space in main storage for the current output partitioned data set's merged directory and the output partitioned data set is not new.
SYSIN DD	Defines the control data set. The control data set normally resides in the input stream; however, it can reside on a system input device, a tape volume, or a direct access volume.

Table 12. Changing Input Record Format Using IEBCOPY

Input	*Output*
Fixed	Fixed Blocked
Fixed Blocked	Fixed
Variable	Variable Blocked
Variable Blocked	Variable

Refer to *OS Storage Estimates*, GC28-6551, to determine when spill data sets are required; see "Space Allocation" below for a description of how to determine the amount of space to allocate.

Restrictions

- SYSPRINT and SYSIN are mandatory DD statements. The block size for the SYSPRINT data set must be a multiple of 121. The block size for the SYSIN data set must be a multiple of 80. Any blocking factor may be specified for these data sets, with a maximum allowable block size of 32,767 bytes.
- The SYSPRINT DD statement must define a data set with fixed blocked or fixed records.
- At least one INPUT DD statement is required; there must be one INPUT DD statement for each unique part in the data set used for input in the job step.
- Input data sets cannot be concatenated.
- There must be an OUTPUT DD statement for each unique partitioned data set used for output in the job step.
- The SYSIN DD statement must define a data set with fixed block or fixed records.

Space Allocation

Sometimes it is necessary to allocate space on spill data sets (SYSUT3 and SYSUT4). To conserve space on the direct access volume, an initial quantity and a secondary quantity for space allocation may be used, as shown in the following SPACE parameter:

SPACE = (c,(x,y))

The c value should be a block length of 80 for SYSUT3 and of 256 for SYSUT4. The x value is the number of blocks in the primary allocation, and the y value is the number of blocks in a secondary allocation.

For SYSUT3, x + 15y must be equal to or greater than the number of entries in the largest input partitioned data set in the copy operation, multiplied by 1.05.

For SYSUT4, x + 15y must be equal to or greater than the number of blocks allocated to the largest output partitioned data set directory in the IEBCOPY job step.

For example, if there are 700 members on the largest input partitioned data set, space could be allocated for SYSUT3 as follows:

```
SPACE = (80,(60,45))
```

However, the total amount of space required for SYSUT3 in the worst case is used only if needed. If space is allocated in this manner for SYSUT4, the user must specify in his SYSUT4 DD statement:

```
DCB = KEYLEN = 8
```

Note that IEBCOPY ignores all other DCB information specified for SYSUT3 and/or SYSUT4. Multivolume SYSUT3 and SYSUT4 data sets are not supported.

Utility Control Statements

IEBCOPY is controlled by the following utility control statements:

- COPY statement, which indicates the beginning of a COPY operation.
- SELECT statement, which specifies which members in the input data set are to be copied.
- EXCLUDE statement, which specifies members in the input data set to be excluded from the copy step.

In addition, when **INDD**, a COPY statement parameter, appears on a card other than the COPY statement, it is referred to as an INDD statement; it can function as a control statement in this context.

Utility control statements may be continued on subsequent cards provided that all the data is contained in columns 2 through 71. Control statement operation and keyword parameters can be abbreviated to their initial letters; for example, COPY can be abbreviated to C.

COPY Statement

The COPY statement is required to initiate all IEBCOPY copy operations. Any number of COPY statements can appear within a single job step.

A COPY statement must precede a SELECT or EXCLUDE statement when members are selected for or excluded from a copy 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 to which it applies. If one or more INDD statements are immediately followed by the /* card or another COPY statement, a full copy is invoked onto the most recent output partitioned data set previously specified.

IEBCOPY uses a copy operation/copy step concept. The unit of work starting with a COPY statement and continuing until another COPY statement or until the end of the control data set is found is called a copy operation. 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, then the operation consists of only one copy step.

Figure 5 shows the copy operation/copy step concept. Two copy operations are shown in the figure: the first begins with the statement containing the name COPOPER1, and the second begins with the statement containing the name COPOPER2.

There are two copy steps within the first copy operation shown in Figure 5: 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. There are two copy steps within the second copy operation: the first begins with the COPY statement and continues through the SELECT statement; the second begins with the INDD statement immediately following the SELECT statement and ends with the same /* (delimiter) statement that ended the copy operation.

The format of the COPY statement is:

```
[label] COPY  OUTDD = ddname
              [,INDD = {ddname1[,ddname2]...                }
                       {ddname1[,ddname2][,(ddname2,R)]...  }
                       {((ddname1,R)[,ddname2]...)          }]
              [,LIST = NO]
```

where:

OUTDD = *ddname*
specifies the name of the output partitioned data set. One ddname is required for each copy operation; the ddname used must be specified on a DD statement.

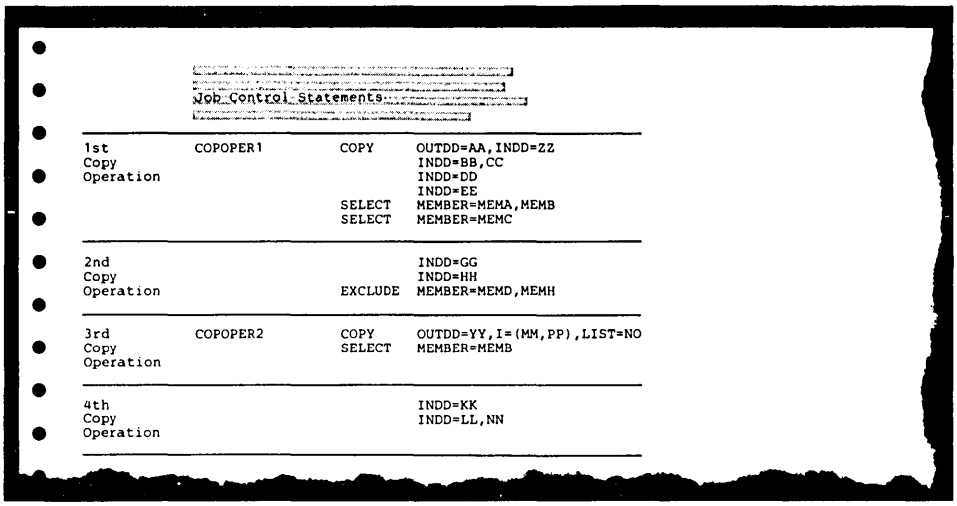


Figure 5. Multiple Copy Operations Within a Job Step

INDD =
specifies the names of the input partitioned data sets. **INDD** may, optionally, be placed on a separate card following a COPY statement containing the **OUTDD** parameter, another INDD statement, a SELECT statement, or an EXCLUDE statement. These values can be coded:

ddname
specifies the ddname, which is specified on a DD statement, of an input partitioned data set. If more than one ddname is specified, the input data sets are processed in the same sequence as the ddnames.

R
specifies that all members to be copied from this input partitioned 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 partitioned data set are copied as usual.) When this option is specified with the **INDD** parameter, it does not have to appear with the MEMBER parameter (discussed in "SELECT Statement" in this chapter) in a selective copy operation. 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 = NO
specifies that the names of copied members are not to be listed on SYSPRINT at the end of each input data set.

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

Only one **INDD** and one **OUTDD** keyword may be placed on a single card. **OUTDD** must appear on the COPY statement. When **INDD** appears on a separate card, no other operands may be specified on that card.

INDD may appear on a separate card; if this option is selected, **INDD** is not preceded by a comma.

If there are no keywords on the COPY card, compatibility with the previous version is implied. In this case, comments may not be placed on this card.

If more than one ddname is specified, the input partitioned data sets are processed in the same sequence as that in which the ddnames are specified.

A full copy is invoked only by specifying different input and output ddnames; that is, by omitting the SELECT or EXCLUDE statement from the copy step.

The compress-in-place function is normally invoked by specifying the same ddname (with the same dsname and volume serial number specified on the DD statement) for both the **OUTDD** and **INDD** parameters of a COPY statement. If multiple entries are made on the INDD statement, a compress in place will occur if one of the input

ddnames is the same as the ddname specified by the **OUTDD** parameter of the COPY statement, provided that SELECT or EXCLUDE is not specified.

The compress-in-place operation cannot be performed for the following:

- A data set with track overflow records.
- A data set with keyed records.
- A data set for which reblocking is specified in the DCB parameter.
- An *unmovable* data set.

When a compression is invoked by specifying the same ddname for the **INDD** and **OUTDD** parameters, and the DD statement specifies a block size that differs from the block size specified in the DSCB, the DSCB block size is overridden; however, no physical reblocking or deblocking is done by IEBCOPY.

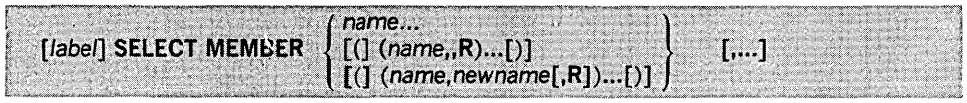
SELECT Statement

The SELECT statement specifies members to be selected from input partitioned data sets. This statement is also used to rename and/or replace selected members on the output partitioned 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 that includes an **INDD** parameter or one or more INDD statements. A SELECT statement cannot appear with an EXCLUDE statement in the same copy operation.

When a selected member is found on an input partitioned data set, it is not searched for again, regardless of whether it is copied. A selected member will not replace an identically named member on the output partitioned data set unless the replace option is specified on either the data set or member level. (For a description of replacing identically named members see "Replacing Identically Named Data Set Members," and "Replacing Selected Members" in this chapter.) In addition, a renamed member will not replace a member on the output partitioned data set that has the same new name as the renamed member, unless the replace option is specified.

The format of the SELECT statement is:



where:

MEMBER =
specifies the members to be selected from the input partitioned data set. The values that can be coded are:

name
specifies the name of a member that is to be selected in a copy step. Each member name specified within one copy step must be unique; that is, duplicate names cannot be specified as either old names, or new names, or both, under any circumstances.

newname
specifies a new name for a selected member. The member is copied onto the output partitioned data set using its new name. If the name already appears on the output partitioned data set, the member is not copied unless replacement (R) is also specified.

R
specifies that the input member is to replace any identically named member that exists on the output partitioned data set.

Note: The control statement operation and keyword parameter can be abbreviated to their initial letters; **SELECT** can be abbreviated to **S** and **MEMBER** can be abbreviated to **M**.

To rename a member, the old member name is specified in the SELECT statement, followed by the new name and, optionally, the **R** parameter. When this option is specified, the *old* member name and *new* member name must be enclosed in a set of 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.

EXCLUDE Statement

The EXCLUDE statement specifies members to be excluded from the copy step. Unlike the selective copy, an exclusive copy causes all specified members on each input partitioned data set to be excluded from the copy.

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 that includes an **INDD** parameter or one or more INDD statements. An EXCLUDE statement cannot appear with a SELECT statement in the same copy step; however, both may be used in a single copy operation.

The format of the EXCLUDE statement is:

[label] **EXCLUDE MEMBER** = [(]*membername1*[,*membername2*]...[)]

where:

MEMBER = [(]*membername1*[,*membername2*]...[)]
specifies members on the input partitioned data sets that are not to be copied to the output partitioned data set. The members are not deleted from the input partitioned data set unless the entire data set is deleted. (This can be done by specifying DISP = DELETE in the operand field of the input DD job control statement.) Each member name specified within one copy step must be unique.

Note: The control statement operation and keyword parameter can be abbreviated to their initial letters; **EXCLUDE** can be abbreviated to **E** and **MEMBER** can be abbreviated to **M.**

IEBCOPY Examples

The following examples illustrate some of the uses of IEBCOPY. Table 13 can be used as a quick reference guide to IEBCOPY examples. The numbers in the "Example" column point to examples that follow.

Table 13. IEBCOPY Example Directory

Operation	*Device*	*Comments*	*Example*
COPY	2314 Disk	Full Copy.	1
COPY	2311 Disk, 2301 Drum, 2302 Disk	Multiple input partitioned data sets. Fixed-blocked and fixed record formats.	2
COPY	2302 Disk, 2314 or 2319 Disk[1]	All members are to be copied Identically named members on the output data set are to be replaced.	3
COPY	2302 Disk, 2314 Disk, 2311 Disk	Selected members are to be copied. Variable blocked data set is to be created. Record formats are variable-blocked and variable.	4
COPY	2302 Disk, 2311 Disk	Selected members are to be copied. One member is to replace an identically named member on the output data set.	5
COPY	2301 Drum, 2314 or 2319 Disk,[1] 2311 Disk	Selected members are to be copied. Members found on first input data set replace identically named members on the output data set.	6
COPY	2311 Disk	Selected members are to be copied. Two members are to be renamed. One renamed member is to replace an identically named member on the output data set.	7
COPY	2314 Disk	Exclusive Copy. Fixed blocked and fixed record formats.	8
COPY	2314 or 2319 Disk[1]	Compress-in-place.	9
COPY	2311 Disk, 2314 Disk	Full copy to be followed by a compress-in-place of the output data set. Replace specified for one input data set.	10
COPY	2314 or 2319 Disk	Multiple copy operations.	11
COPY	2311 Disk	Multiple copy operations.	12

[1] The 2319 disk is functionally equivalent to the 2314 disk; to use the 2319, specify 2314 in the control statement.

IEBCOPY Example 1

In this example, a partitioned data set (DATASET5) is to be copied from one disk volume to another. Figure 6 shows the input and output data sets before and after processing.

The example follows:

```
//COPY      JOB   06#990,MCEWAN
//JOBSTEP   EXEC  PGM=IEBCOPY
//SYSPRINT  DD    SYSOUT=A
//INOUT4    DD    DSNAME=DATASET4,UNIT=2314,VOL=SER=111112,
//                DISP=(NEW,KEEP),SPACE=(TRK,(5,1,2))
//INOUT5    DD    DSNAME=DATASET5,UNIT=2314,VOL=SER=111113,
//                DISP=OLD
//SYSUT3    DD    UNIT=2314,SPACE=(TRK,(1))
//SYSUT4    DD    UNIT=2314,SPACE=(TRK,(1))
//SYSIN     DD    *
COPYOPER    COPY  OUTDD=INOUT4,INDD=INOUT5
/*
```

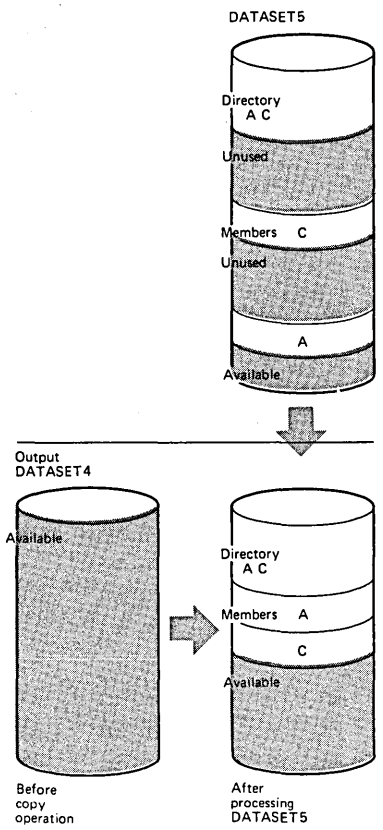


Figure 6. Copying a Partitioned Data Set—Full Copy

The control statements are discussed below:

- INOUT4 DD defines a partitioned data set (DATASET4). This data set is new and is to be kept after the copy operation. Five tracks are allocated for the data set on a 2314 volume. Two blocks are allocated for directory entries.
- INOUT5 DD defines a partitioned data set (DATASET5), which resides on a 2314 volume and contains two members (A and C).
- SYSUT3 DD defines a temporary spill data set. One track is allocated on a 2314 volume.
- SYSUT4 DD defines a temporary spill data set. One track is allocated on a 2314 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 absence of a SELECT or EXCLUDE statement causes a default to a full copy. The **OUTDD** parameter specifies INOUT4 as the DD statement for the output data set (DATASET4); the

INDD parameter specifies INOUT5 as the DD statement for the input data set. After the copy operation is finished, the output data set (DATASET4) will contain the same members that are on the input data set (DATASET5); however, there will be no embedded unused space on DATASET4.

The temporary spill data sets may or may not be opened, depending on the amount of main storage available; therefore, it is suggested that the SYSUT3 and SYSUT4 DD statements always appear in the job stream.

IEBCOPY Example 2

In this example, members are to be 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 7 shows the input and output data sets before and after processing.

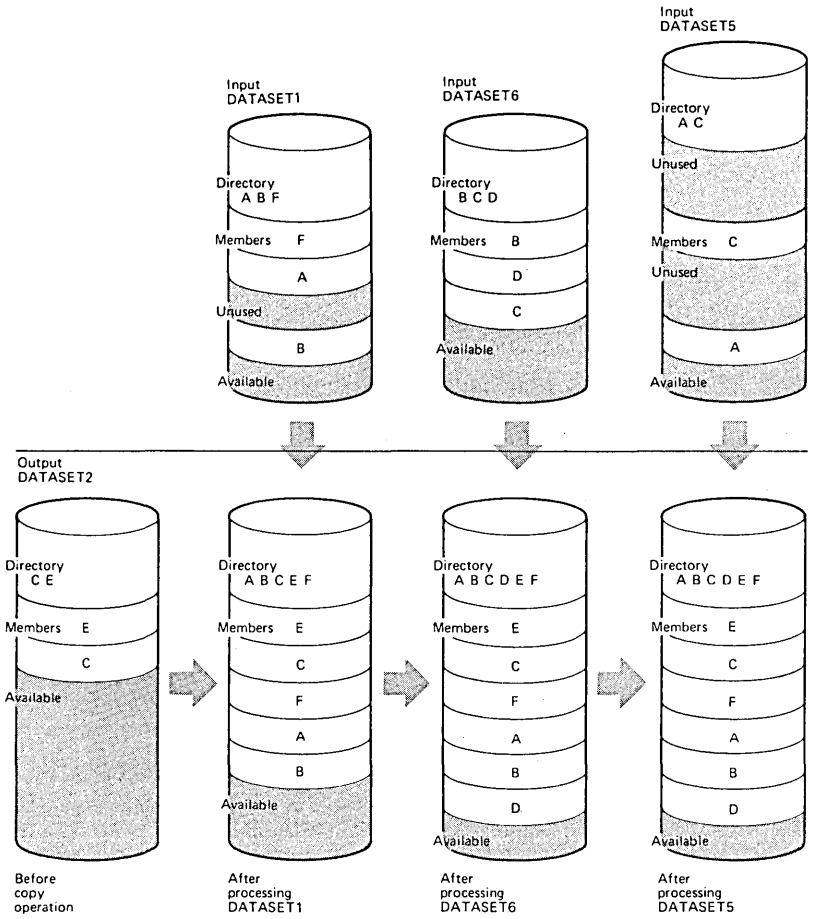


Figure 7. Copying from Three Input Partitioned Data Sets

The example follows:

```
//COPY     JOB   06#990,MCEWAN
//JOBSTEP  EXEC  PGM=IEBCOPY
//SYSPRINT DD    SYSOUT=A
//INOUT1   DD   DSNAME=DATASET1,UNIT=2311,VOL=SER=111112,
//              DISP=(OLD,KEEP)
//INOUT5   DD    DSNAME=DATASET5,UNIT=2301,VOL=SER=111114,
//              DISP=OLD
//INOUT2   DD    DSNAME=DATASET2,UNIT=2302,VOL=SER=111115,
//              DISP=(OLD,KEEP)
//INOUT6   DD    DSNAME=DATASET6,UNIT=2301,VOL=SER=111117,
//              DISP=(OLD,DELETE)
//SYSUT3   DD    UNIT=2311,SPACE=(TRK,(1))
//SYSUT4   DD    UNIT=2311,SPACE=(TRK,(1))
//SYSIN    DD    *
COPYOPER  COPY   OUTDD=INOUT2
              INDD=INOUT1
              INDD=INOUT6
              INDD=INOUT5
/*
```

The control statements are discussed below:

- INOUT1 DD defines a partitioned data set (DATASET1). This data set, which resides on a 2311 volume, 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.
- INOUT5 DD defines a partitioned data set (DATASET5), which resides on a 2301 volume. 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.
- INOUT2 DD defines a partitioned data set (DATASET2), which resides on a 2302 volume. This data set contains two members (C and E) in fixed blocked format. The members have a logical record length of 80 bytes and a block size of 240 bytes.
- INOUT6 DD defines a partitioned data set (DATASET6), which resides on a 2301 volume. This data set contains three members (B, C, and D) in fixed blocked 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 DD defines a temporary spill data set. One track is allocated on a 2311 volume.
- SYSUT4 DD defines a temporary spill data set. One track is allocated on a 2311 volume.
- 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 absence of a SELECT or EXCLUDE statement causes a default to a full copy. The **OUTDD** parameter specifies INOUT2 as the DD statement for the output data set (DATASET2).
- The first INDD statement specifies INOUT1 as the DD statement for the first input data set (DATASET1) to be processed. All members (A, B, and F) are copied to the output data set (DATASET2).
- The second INDD statement specifies INOUT6 as the DD statement for the second input data set (DATASET6) to be processed. Processing occurs, as follows: (1) members B and C, which already exist on DATASET2, are not copied to the output data set (DATASET2), (2) member D is copied to the output data set (DATASET2), and (3) all members on DATASET6 are lost when the data set is deleted.
- The third INDD statement specifies INOUT5 as the DD statement for the third input data set (DATASET5) to be processed. No members are copied to the output data set (DATASET2) because all of them exist on DATASET2.

The temporary spill data sets may or may not be opened, depending on the amount of main storage available; therefore, it is suggested that the SYSUT3 and SYSUT4 DD statements always appear in the job stream.

IEBCOPY Example 3

In this example, members are to be copied from an input partitioned data set (DATASET6) to an existing output partitioned data set (DATASET2). In addition, all copied members are to replace identically named members on the output partitioned data set. Figure 8 shows the input and output data sets before and after processing.

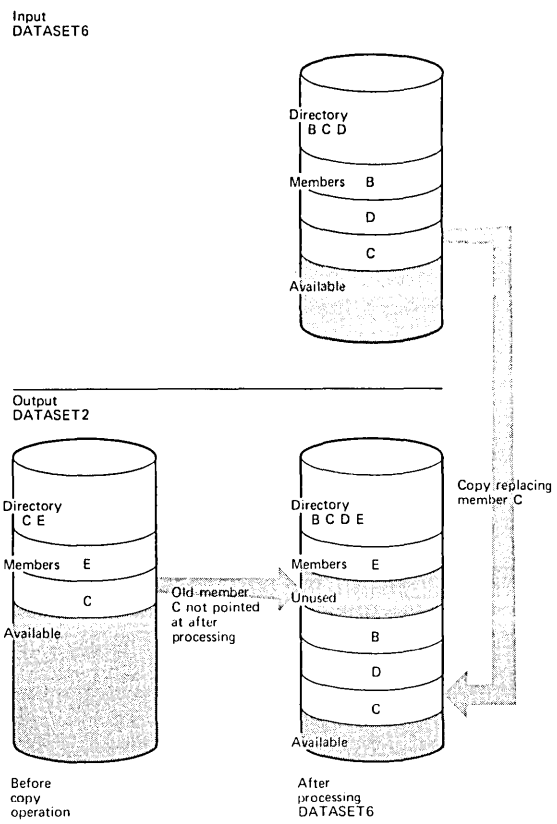


Figure 8. Copy Operation with "Replace" Specified on the Data Set Level

The example follows:

```
//COPY      JOB   06#990,MCEWAN
//JOBSTEP   EXEC  PGM=IEBCOPY
//SYSPRINT  DD    SYSOUT=A
//INOUT2    DD    DSNAME=DATASET2,UNIT=2314,VOL=SER=111113,
//              DISP=OLD
//INOUT6    DD    DSNAME=DATASET6,UNIT=2302,VOL=SER=111117,
//              DISP=(OLD,KEEP)
//SYSUT3    DD    UNIT=2311,SPACE=(TRK,(1))
//SYSUT4    DD    UNIT=2311,SPACE=(TRK,(1))
//SYSIN     DD    *
COPYOPER    COPY  OUTDD=INOUT2,INDD=((INOUT6,R))
/*
```

The control statements are discussed below:

- INOUT2 DD defines a partitioned data set (DATASET2), which resides on a 2314 volume. This data set contains two members (C and E).
- INOUT6 DD defines a partitioned data set (DATASET6), which resides on a 2302 volume. This data set contains three members (B, C, and D).
- SYSUT3 DD defines a temporary spill data set. One track is allocated on a 2311 volume.
- SYSUT4 DD defines a temporary spill data set. One track is allocated on a 2311 volume.
- SYSIN DD defines the control data set, which follows in the input stream. The data set contains a COPY statement and an INDD statement.
- COPY indicates the start of the copy operation. The absence of a SELECT or EXCLUDE statement causes a default to a full copy. The **OUTDD** parameter specifies INOUT2 as the DD statement for the output data set (DATASET2). The **INDD** parameter specifies INOUT6 as the DD statement for the input data set (DATASET6). Members B, C, and D are copied to the output data set (DATASET2). The pointer in the output data set directory is changed to point to the new (copied) member C; thus, the space occupied by the old member C is embedded unused space. Member C is copied even though the output data set already contains a

member named "C" because the replace option is specified for all identically named members on the input data set; that is, the replace option is specified on the data set level.

The temporary spill data sets may or may not be opened, depending on the amount of main storage available; therefore, it is suggested that the SYSUT3 and SYSUT4 DD statements always appear in the job stream.

IEBCOPY Example 4

In this example, five members (A, C, D, E, and G) are to be selected from two input partitioned data sets (DATASET6 and DATASET2) to be copied to a new output partitioned data set (DATASET4). Figure 9 shows the input and output data sets before and after processing.

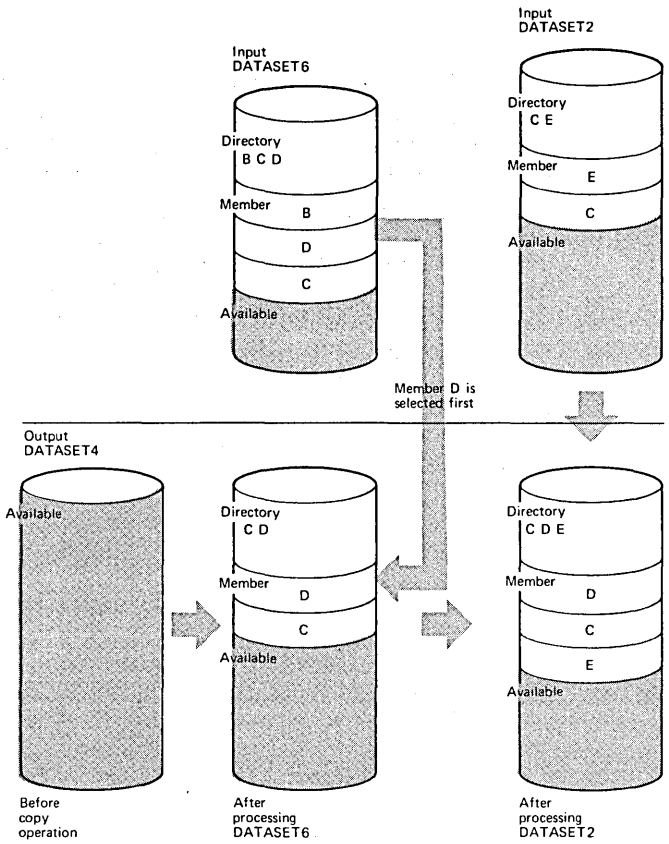


Figure 9. Copying Selected Members with Reblocking and Deblocking

The example follows:

```
//COPY      JOB      06#990,MCEWAN
//JOBSTEP   EXEC     PGM=IEBCOPY
//SYSPRINT  DD       SYSOUT=A
//INOUT2    DD       DSNAME=DATASET2,UNIT=2314,VOL=SER=111114,
//                DISP=(OLD,DELETE)
//INOUT6    DD       DSNAME=DATASET6,UNIT=2302,VOL=SER=111117,
//                DISP=(OLD,KEEP)
//INOUT4    DD       DSNAME=DATASET4,UNIT=2311,VOL=SER=111116,
//                DISP=(NEW,KEEP),SPACE=(TRK,(5,,2)),
//                DCB=(RECFM=VB,LRECL=96,BLKSIZE=300)
//SYSUT3    DD       UNIT=2311,SPACE=(TRK,(1))
//SYSUT4    DD       UNIT=2311,SPACE=(TRK,(1))
//SYSIN     DD       *
COPYOPER    COPY     OUTDD=INOUT4
                INDD=INOUT6
                INDD=INOUT2
 SELECT   MEMBER=C,D,E,A,G
/*
```

The control statements are discussed below:

- INOUT2 DD defines a partitioned data set (DATASET2), which resides on a 2314 volume. This data set contains two members (C and E) in variable blocked format with a logical record length of 96 bytes and a block size of 500 bytes. This data set is to be deleted when processing is completed.

- INOUT6 DD defines a partitioned data set (DATASET6), which resides on a 2302 volume. This data set contains three members (B, C, and D) in variable format with a logical record length of 96 bytes and a block size of 100 bytes.
- INOUT4 DD defines a partitioned data set (DATASET4). This data set is new and is to be kept after the copy operation. Five tracks are allocated for the data set on a 2311 volume. Two blocks are allocated for directory entries. In addition, records are to be copied to this data set in variable blocked format with a logical record length of 96 bytes and a block size of 300 bytes.
- SYSUT3 DD defines a temporary spill data set. One track is allocated on a 2311 volume.
- SYSUT4 DD defines a temporary spill data set. One track is allocated on a 2311 volume.
- SYSIN DD defines the control data set, which follows in the input stream. The data set contains a COPY statement, two INDD statements, and a SELECT statement.
- COPY indicates the start of the copy operation. The presence of a SELECT statement causes a selective copy. The **OUTDD** parameter specifies INOUT4 as the DD statement for the output data set (DATASET4).
- The first INDD statement specifies INOUT6 as the DD statement for the first input data set (DATASET6) to be processed. The members specified on the SELECT statement are searched for. The found members (C and D) are copied to the output data set (DATASET4) in the order in which they reside on the input data set, that is, in TTR order. In this case, member D is copied first, and then member C is copied.
- The second INDD statement specifies INOUT2 as the DD statement for the second input data set (DATASET2) to be processed. The members specified on the SELECT statement and not found on the first input data set are searched for. The found member (E) is copied onto the output data set (DATASET4). All members on DATASET2 are lost when the data set is deleted.
- SELECT specifies the members to be selected from the input data sets (DATASET6 and DATASET2) to be copied to the output data set (DATASET4).

The temporary spill data sets may or may not be opened, depending on the amount of main storage available; therefore, it is suggested that the SYSUT3 and SYSUT4 DD statements always appear in the job stream.

IEBCOPY Example 5

In this example, two members (A and B) are to be selected from two input partitioned data sets (DATASET5 and DATASET6) to be copied to an existing output partitioned data set (DATASET1). Member B is to replace an identically named member that already exists on the output data set. Figure 10 shows the input and output data sets before and after processing.

The example follows:

```
//COPY     JOB      06#990,MCEWAN
//JOBSTEP  EXEC     PGM=IEBCOPY
//SYSPRINT DD       SYSOUT=A
//INOUT1   DD       DSNAME=DATASET1,UNIT=2311,VOL=SER=111112,
//              DISP=(OLD,KEEP)
//INOUT6   DD       DSNAME=DATASET6,UNIT=2302,VOL=SER=111115,
//              DISP=OLD
//INOUT5   DD       DSNAME=DATASET5,UNIT=2311,VOL=SER=111116,
//              DISP=(OLD,KEEP)
//SYSUT3   DD       UNIT=2311,SPACE=(TRK,(1))
//SYSUT4   DD       UNIT=2311,SPACE=(TRK,(1))
//SYSIN    DD       *
COPYOPER   COPY     OUTDD=INOUT1
               INDD=INOUT5,INOUT6
 SELECT  MEMBER=((B,,R),A)
/*
```

The control statements are discussed below:

- INOUT1 DD defines a partitioned data set (DATASET1). This data set resides on a 2311 volume and contains three members (A, B, and F).
- INOUT6 DD defines a partitioned data set (DATASET6). This data set resides on a 2302 volume and contains three members (B, C, and D).
- INOUT5 DD defines a partitioned data set (DATASET5). This data set resides on a 2311 volume and contains two members (A and C).
- SYSUT3 DD defines a temporary spill data set. One track is allocated on a 2311 volume.

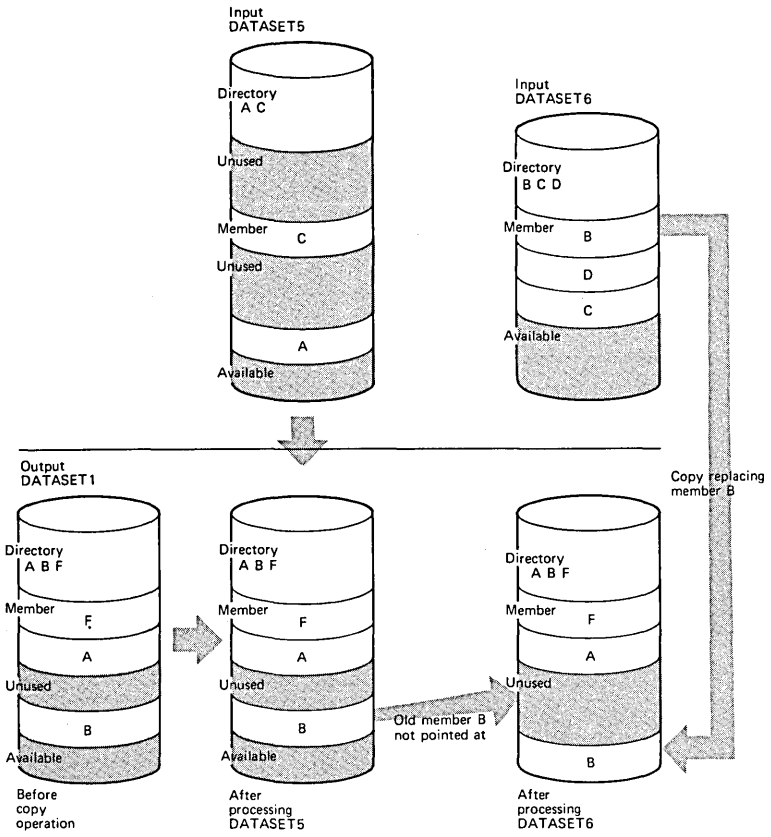


Figure 10. Selective Copy with "Replace" Specified on the Member Level

- SYSUT4 DD defines a temporary spill data set. One track is allocated on a 2311 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 presence of a SELECT statement causes a selective copy. The **OUTDD** parameter specifies INOUT1 as the DD statement for the output data set (DATASET1).
- INDD specifies INOUT5 as the DD statement for the first input data set (DATASET5) to be processed and INOUT6 as the DD statement for the second input data set (DATASET6) 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 the output data set because it already exists on DATASET2 and the replace option is not specified, (3) selected members not found on DATASET5 are searched for on DATASET6, and (4) member B is found and copied to the output data set (DATASET1), even though a member named B already exists on the output data set, because the replace option is specified for member B on the member level. The pointer in the output data set 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).

The temporary spill data sets may or may not be opened, depending on the amount of main storage available; therefore, it is suggested that the SYSUT3 and SYSUT4 DD statements always appear in the job stream.

IEBCOPY Example 6

In this example, two members (A and B) are to be selected from two input partitioned data sets (DATASET5 and DATASET6) to be copied to an existing output partitioned data set (DATASET1). All members found on DATASET5 are to replace identically named members on DATASET1. Figure 11 shows the input and output data sets before and after processing.

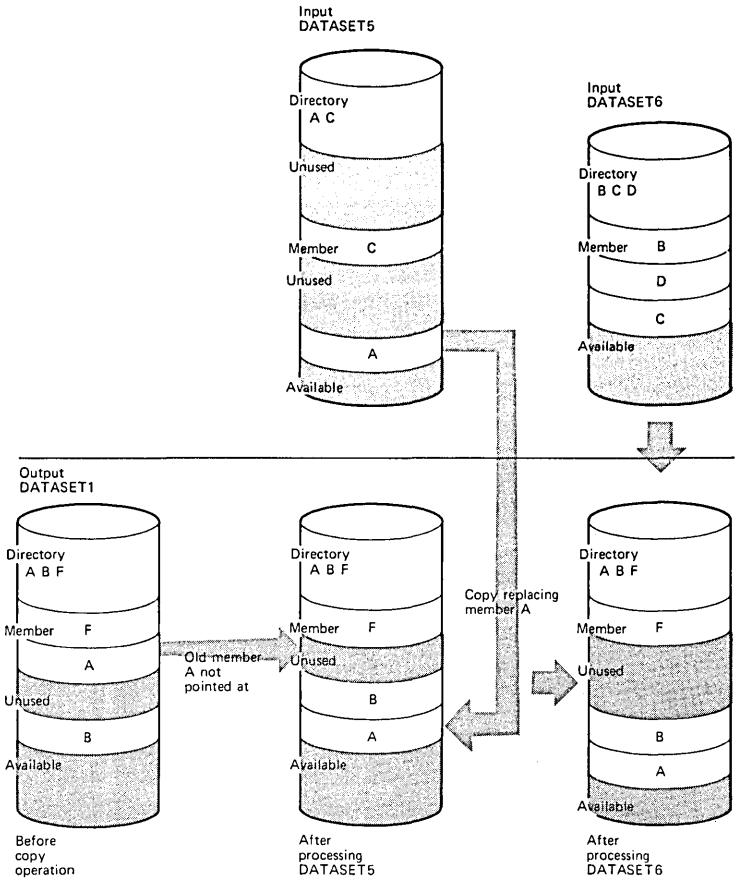


Figure 11. Selective Copy with "Replace" Specified on the Data Set Level

The example follows:

```
//COPY      JOB      06#990,MCEWAN
//JOBSTEP   EXEC     PGM=IEBCOPY
//SYSPRINT  DD       SYSOUT=A
//INOUT1    DD       DSNAME=DATASET1,UNIT=2311,VOL=SER=111112,
//                DISP=(OLD,KEEP)
//INOUT5    DD       DSNAME=DATASET5,UNIT=2314,VOL=SER=111114,
//                DISP=(OLD,DELETE)
//INOUT6    DD       DSNAME=DATASET6,UNIT=2301,VOL=SER=111115,
//                DISP=(OLD,KEEP)
//SYSUT3    DD       UNIT=2311,SPACE=(TRK,(1))
//SYSUT4    DD       UNIT=2311,SPACE=(TRK,(1))
//SYSIN     DD       *
COPYOPER    COPY     OUTDD=INOUT1
                  INDD=((INOUT5,R),INOUT6)
 SELECT   MEMBER=(A,B)
/*
```

The control statements are discussed below:

- INOUT1 DD defines a partitioned data set (DATASET1). This data set resides on a 2311 volume and contains three members (A, B, and F).
- INPUT5 DD defines a partitioned data set (DATASET5). This data set contains two members (A and C) and can reside on either a 2314 or 2319 volume. This data set is to be deleted when processing is completed.
- INOUT6 DD defines a partitioned data set (DATASET6). This data set contains three members (B, C, and D) and resides on a 2301 volume.
- SYSUT3 DD defines a temporary spill data set. One track is allocated on a 2311 volume.
- SYSUT4 DD defines a temporary spill data set. One track is allocated on a 2311 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 presence of a SELECT statement causes a selective copy. The OUTDD operand specifies INOUT1 as the DD statement for the output data set (DATASET1).
- INDD specifies INOUT5 as the DD statement for the first input data set (DATASET5) to be processed and INOUT6 as the statement for the second input data set (DATASET6) to be processed. Processing occurs, as follows: (1) selected members are searched for on DATASET5, (2) member A is found and copied to the output data set (DATASET1) because the replace option was specified on the data set level for DATASET5, (3) member B, which was not found on DATASET5 is searched for and found on DATASET6, (4) member B is not copied because DATASET1 already contains a member called member B and the replace option is not specified for DATASET6. The pointer in the output data set directory is changed to point to the new (copied) member A; thus, the space occupied by the old member A 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).

The temporary spill data sets may or may not be opened, depending on the amount of main storage available; therefore, it is suggested that the SYSUT3 and SYSUT4 DD statements always appear in the job stream.

IEBCOPY Example 7

In this example, four members (A, B, C, and D) are to be selected from an input partitioned data set (DATASET6) to be copied to an existing output partitioned data set (DATASET3). Member B is to be renamed H; member C is to be renamed J; and member D is to be renamed K. In addition, member C (renamed J) is to replace the identically named member (J) on the output partitioned data set. Figure 12 shows the input and output data sets before and after processing.

The example follows:

```
//COPY     JOB      #990,MCEWAN
//JOBSTEP  EXEC     PGM=IEBCOPY
//SYSPRINT DD       SYSOUT=A
//INOUT3   DD       DSNAME=DATASET3,UNIT=2311,VOL=SER=111114,
//              DISP=(OLD,KEEP)
//INOUT6   DD       DSNAME=DATASET6,UNIT=2311,VOL=SER=111117,
//              DISP=(OLD,DELETE)
//SYSUT3   DD       UNIT=2311,SPACE=(TRK,(1))
//SYSUT4   DD       UNIT=2311,SPACE=(TRK,(1))
//SYSIN    DD       *
COPYOPER   COPY     OUTDD=INOUT3
               INDD=INOUT6
 SELECT MEMBER=((B,H),(C,J,R),A,(D,K))
/*
```

The control statements are discussed below:

- INOUT3 DD defines a partitioned data set (DATASET3). This data set contains four members (D, G, H, and J).
- INOUT6 DD defines a partitioned data set (DATASET6). This data set contains three members (B, C, and D) and resides on a 2311 volume. DATASET6 is to be deleted when processing is completed; thus, all members on this data set are lost.
- SYSUT3 DD defines a temporary spill data set. One track is allocated on a 2311 volume.
- SYSUT4 DD defines a temporary spill data set. One track is allocated on a 2311 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 presence of a SELECT statement causes a selective copy. The **OUTDD** parameter specifies INOUT3 as the DD statement for the output data set (DATASET3).
- INDD specifies INOUT6 as the DD statement for the input data set (DATASET6). Processing occurs, as follows: (1) selected members are searched for on DATASET6, (2) member B is found, but is not copied to DATASET3 because its intended new name (H) is identical to the name of a member (H), which already exists on the output data set, and replace is not specified, (3) member C is found and copied to the output data set (DATASET3), although its new name (J) is

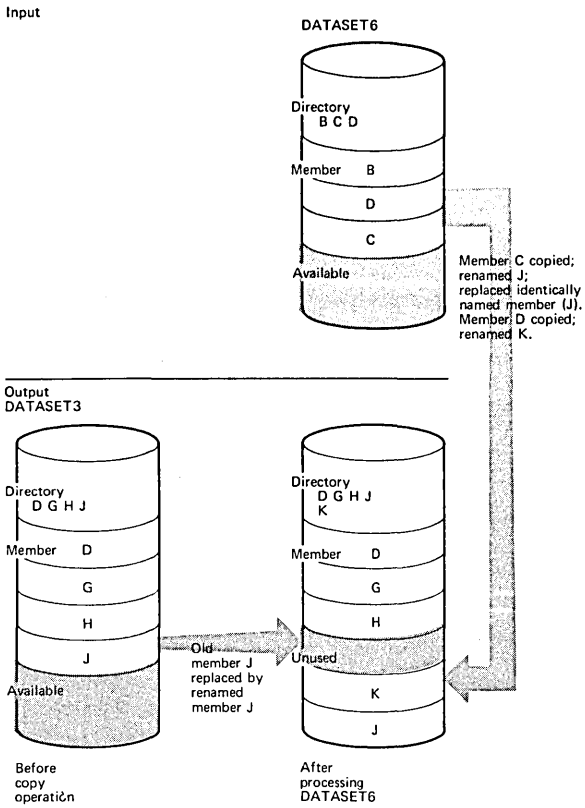


Figure 12. Renaming Selected Members Using IEBCOPY

identical to the name of a member (J), which already exists on the output data set, because the replace option is specified for the renamed member, and (4) member D is copied onto the output data set (DATASET3) because its new name (K) does not already exist there.

- SELECT specifies the members to be selected from the input data set (DATASET6) to be copied to the output data set (DATASET3).

The temporary spill data sets may or may not be opened, depending on the amount of main storage available; therefore, it is suggested that the SYSUT3 and SYSUT4 DD statements always appear in the job stream.

IEBCOPY Example 8

In this example, five members (A, B, C, J, and L) are to be excluded from the copy operation when each of the input partitioned data sets (DATASET1, DATASET3, and DATASET6) is processed. In addition, replace is specified for the last input partitioned data set (DATASET6) to be processed; thus, with the exception of the members specified on the EXCLUDE statement, all members on DATASET6 will replace any identically named members on the output partitioned data set (DATASET4). Figure 13 shows the input and output data sets before and after processing.

The example follows:

```
//COPY     JOB       06#990,MCEWAN
//JOBSTEP  EXEC      PGM=IEBCOPY
//SYSPRINT DD        SYSOUT=A
//INOUT1   DD        DSNAME=DATASET1,UNIT=2314,VOL=SER=111112,
//                DISP=(OLD,KEEP)
//INOUT3   DD        DSNAME=DATASET3,UNIT=2314,VOL=SER=111114,
//                DISP=OLD
//INOUT4   DD        DSNAME=DATASET4,UNIT=2314,VOL=SER=111115,
//                DISP=(NEW,KEEP),SPACE=(TRK,(5,1,2)),
//                DCB=(LRECL=100,RECFM=FB,BLKSIZE=400)
//INOUT6   DD        DSNAME=DATASET6,UNIT=2314,VOL=SER=111116,
//                DISP=OLD
//SYSUT3   DD        UNIT=2314,SPACE=(TRK,(1))
//SYSUT4   DD        UNIT=2314,SPACE=(TRK,(1))
//SYSIN    DD        *
COPYOPER COPY   OUTDD=INOUT4,INDD=INOUT1,INOUT3,(INOUT6,R)
 EXCLUDE  MEMBER=A,J,B,L,C
/*
```

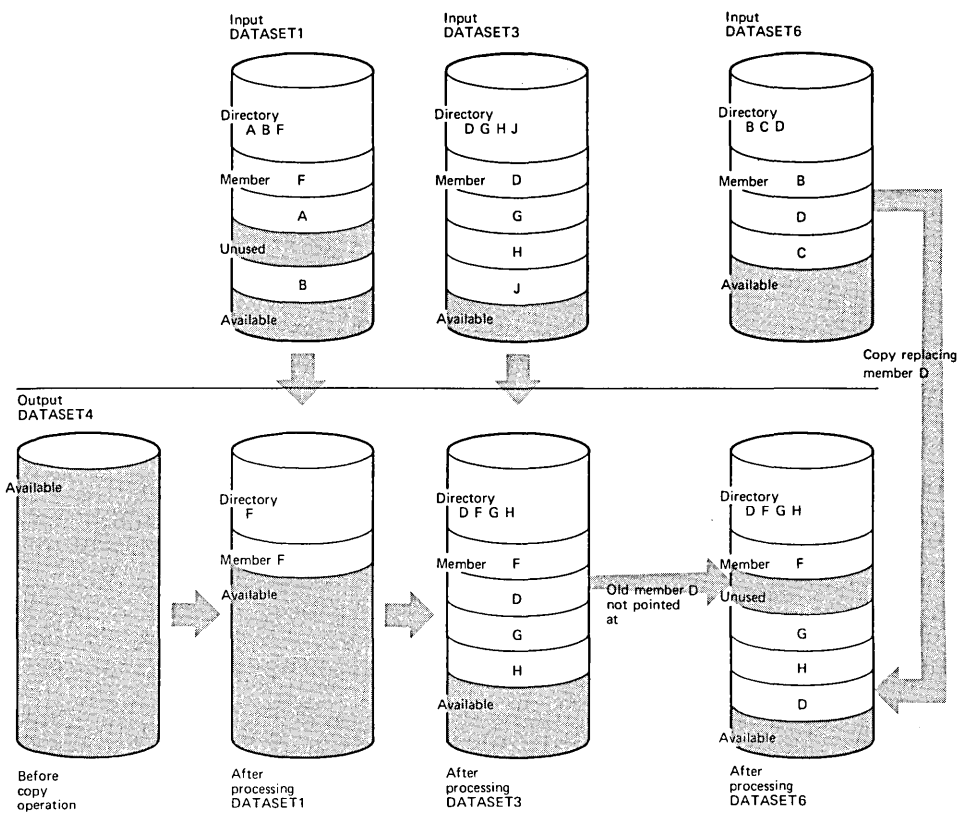


Figure 13. Exclusive Copy with "Replace" Specified for One Input Partitioned Data Set

The control statements are discussed below:

- INOUT1 DD defines a partitioned data set (DATASET1). This data set contains three members (A, B, and F) and resides on a 2314 volume. The record format is fixed blocked with a logical record length of 100 bytes and a block size of 400 bytes.
- INOUT3 DD defines a partitioned data set (DATASET3), which resides on a 2314 volume. This data set contains four members (D, G, H, and J) in fixed blocked format with a logical record length of 100 bytes and a block size of 600 bytes.
- INOUT4 DD defines a new partitioned data set (DATASET4). Five tracks are allocated for the copied members on a 2314 volume. Two blocks are allocated for directory entries. In addition records are to be copied to this data set in fixed blocked format with a logical record length of 100 bytes and a block size of 400 bytes.
- INOUT6 DD defines a partitioned data set (DATASET6). This data set contains three members (B, C, and D) in fixed format. The records have a logical record length of 100 bytes and a block size of 100 bytes. This data set resides on a 2314 volume.
- SYSUT3 DD defines a temporary spill data set. One track is allocated on a 2314 volume.
- SYSUT4 DD defines a temporary spill data set. One track is allocated on a 2314 volume.
- SYSIN DD defines the control data set, which follows in the input stream. The data set contains a COPY statement and an EXCLUDE statement.
- COPY indicates the start of the copy operation. The presence of an EXCLUDE statement causes an exclusive copy. The **OUTDD** parameter specifies INOUT4 as the DD statement for the output data set (DATASET4). The **INDD** parameter specifies INOUT1 as the DD statement for the first input data set (DATASET1) to be processed, INOUT3 as the DD statement for the second input data set (DATASET3) to be processed, and INOUT6 as the DD statement for the last input data set (DATASET6) to be processed. Processing occurs, as follows: (1) member F, which is not named on the EXCLUDE statement, is copied from DATASET1, (2) members D, G, and H, which are not named on the EXCLUDE statement, are copied

from DATASET3, and (3) member D is copied from DATASET6 because the replace option is specified for nonexcluded members. The pointer in the output data set directory is changed to point at the new (copied) member D; thus, the space occupied by the *old* member D (copied from DATASET3) is unused.

- EXCLUDE specifies the members to be excluded from the copy operation. The named members are excluded from all of the input partitioned data sets specified in the copy operation.

The temporary spill data sets may or may not be opened, depending on the amount of main storage available; therefore, it is suggested that the SYSUT3 and SYSUT4 DD statements always appear in the job stream.

IEBCOPY Example 9

In this example, a partitioned data set (DATASET5) is to be compressed in place. Figure 14 shows the input and output data sets before and after processing.

The example follows:

```
//COPY      JOB   06#990,MCEWAN
//JOBSTEP   EXEC  PGM=IEBCOPY
//SYSPRINT  DD    SYSOUT=A
//INOUT5    DD    DSNAME=DATASET5,UNIT=2314,VOL=SER=111113,
//                DISP=(OLD,KEEP)
//SYSUT3    DD    UNIT=2311,SPACE=(TRK,(1))
//SYSUT4    DD    UNIT=2311,SPACE=(TRK,(1))
//SYSIN     DD    *
COPYOPER    COPY  OUTDD=INOUT5,INDD=INOUT5
/*
```

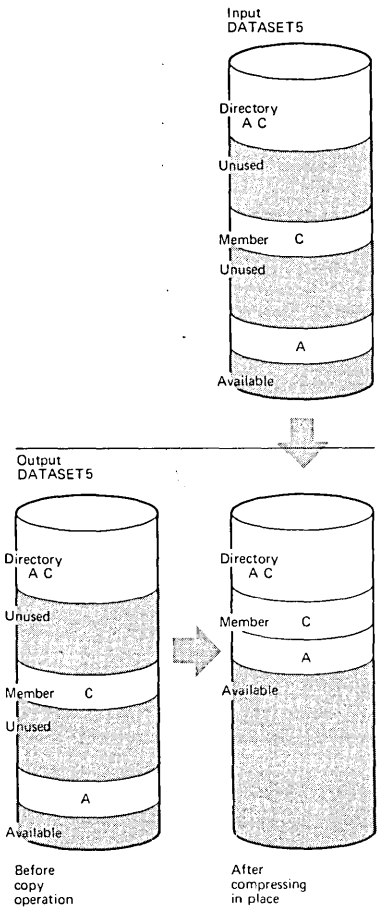


Figure 14. Compressing a Data Set in Place

The control statements are discussed below:

- INOUT5 DD defines a partitioned data set (DATASET5). This data set contains two members (A and C) and can reside on either a 2314 or 2319 volume.
- SYSUT3 DD defines a temporary spill data set. One track is allocated on a 2311 volume.

- SYSUT4 DD defines a temporary spill data set. One track is allocated on a 2311 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 absence of a SELECT or EXCLUDE statement causes a default to a full copy; however, the same DD statement is specified for both the **INDD** and **OUTDD** parameters, causing a compress in place of the specified data set. The **OUTDD** parameter specifies INOUT5 as the DD statement for the output data set (DATASET5). The **INDD** parameter also specifies INOUT5 as the DD statement for the input data set (DATASET5).

The temporary spill data sets may or may not be opened, depending on the amount of main storage available; therefore, it is suggested that the SYSUT3 and SYSUT4 DD statements always appear in the job stream.

IEBCOPY Example 10

In this example, two input partitioned data sets (DATASET5 and DATASET6) are to be copied to an existing output partitioned data set (DATASET1). In addition, all members on DATASET6 are to be 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 to be compressed in place. Figure 15 shows the input and output data sets before and after processing.

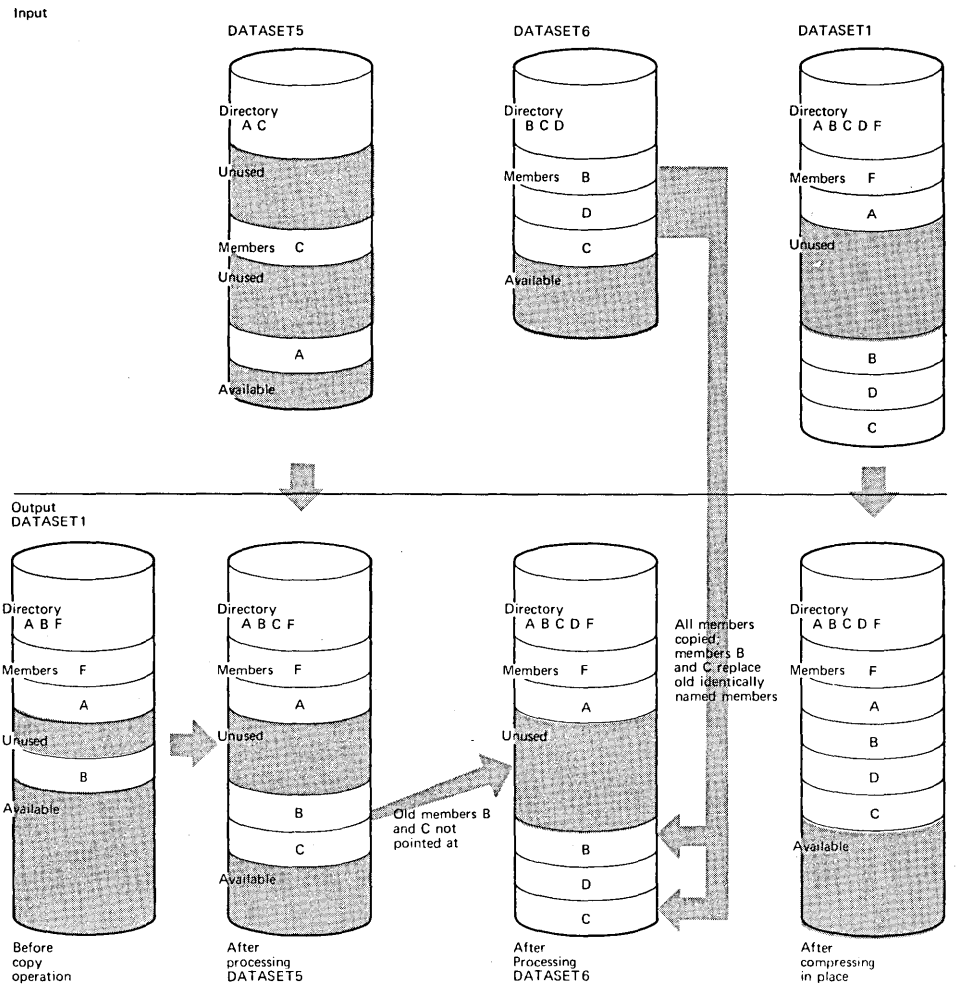


Figure 15. Compress-in-Place Following Full Copy with "Replace" Specified

The example follows:

```
//COPY      JOB   06#990,MCEWAN
//JOBSTEP   EXEC  PGM=IEBCOPY
//SYSPRINT  DD    SYSOUT=A
//INOUT1    DD    DSNAME=DATASET1,UNIT=2314,VOL=SER=111112,
//                DISP=(OLD,KEEP)
//INOUT5    DD    DSNAME=DATASET5,UNIT=2311,VOL=SER=111114,
//                DISP=OLD
//INOUT6    DD    DSNAME=DATASET6,UNIT=2311,VOL=SER=111115,
//                DISP=(OLD,KEEP)
//SYSUT3    DD    UNIT=2311,SPACE=(TRK,(1))
//SYSUT4    DD    UNIT=2311,SPACE=(TRK,(1))
//SYSIN     DD    *
COPYOPER    COPY  OUTDD=INOUT1
                  INDD=INOUT5,(INOUT6,R),INOUT1
/*
```

The control statements are discussed below:

- INOUT1 DD defines a partitioned data set (DATASET1). This data set contains three members (A, B, and F) and resides on a 2314 or 2319 volume.
- INOUT5 DD defines a partitioned data set (DATASET5). This data set contains two members (A and C) and resides on a 2311 volume.
- INOUT6 DD defines a partitioned data set (DATASET6). This data set contains three members (B, C, and D) and resides on a 2311 volume.
- SYSUT3 DD defines a temporary spill data set. One track is allocated on a 2311 volume.
- SYSUT4 DD defines a temporary spill data set. One track is allocated on a 2311 volume.
- SYSIN DD defines the control data set, which follows in the input stream. The data set contains a COPY statement and an INDD statement.
- COPY indicates the start of the copy operation. The OUTDD operand specifies INOUT1 as the DD statement for the output data set (DATASET1). The absence of a SELECT or EXCLUDE statement causes a default to a full copy.
- INDD specifies INOUT5 as the DD statement for the first input data set (DATASET5) to be processed. It then specifies INOUT6 as the DD statement for the second input data set (DATASET6) to be processed; in addition, the replace option is specified for all members copied from DATASET6. Finally, it specifies INOUT1 as the DD statement for the last input data set (DATASET1) to be processed; this causes a compress in place of DATASET1 because it is also specified as the output data set. Processing occurs, as follows: (1) member A is not copied from DATASET5 onto the output data set (DATASET1) because it already exists on DATASET1 and the replace option was not specified for DATASET5, (2) member C is copied from DATASET5 to the output data set (DATASET1), occupying the first available space, and (3) all members are copied from DATASET6 to the output data set (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 the output data set 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.

The temporary spill data sets may or may not be opened, depending on the amount of main storage available; therefore, it is suggested that the SYSUT3 and SYSUT4 DD statements always appear in the job stream.

IEBCOPY Example 11

In this example, members are to be selected, excluded, and copied from input partitioned data sets onto an output partitioned data set. This example is designed to illustrate multiple copy operations. Figure 16 shows the input and output data sets before and after processing.

The example follows:

```
//COPY     JOB       06#990,MCEWAN
//JOBSTEP  EXEC      PGM=IEBCOPY
//SYSPRINT DD        SYSOUT=A
//INOUTA   DD        DSNAME=DATASETA,UNIT=2314,VOL=SER=111113,
//              DISP=OLD
//INOUTB   DD        DSNAME=DATASETB,UNIT=2314,VOL=SER=111115,
//              DISP=(OLD,KEEP)
//INOUTC   DD        DSNAME=DATASETC,UNIT=2314,VOL=SER=111114,
//              DISP=(OLD,KEEP)
//INOUTD   DD        DSNAME=DATASETD,UNIT=2314,VOL=SER=111116,
//              DISP=OLD
//INOUTE   DD        DSNAME=DATASETE,UNIT=2314,VOL=SER=111117,
//              DISP=OLD
//INOUTX   DD        DSNAME=DATASETX,UNIT=2314,VOL=SER=111112,
//              DISP=(NEW,KEEP),SPACE=(TRK,(5,1,2))
//SYSUT3   DD        UNIT=2314,SPACE=(TRK,(1))
//SYSUT4   DD        UNIT=2314,SPACE=(TRK,(1))
//SYSIN    DD        *
COPERST1   COPY      O=INOUTX,I=INOUTA
 COPY      OUTDD=INOUTA,INDD=INOUTA
              INDD=INOUTB
 COPY      O=INOUTA
              INDD=INOUTD
 EXCLUDE   MEMBER=MM
              INDD=INOUTC
 SELECT    MEMBER((ML,MD,R))
              INDD=INOUTE
/*
```

The control statements are discussed below:

- INOUTA DD defines a partitioned data (DATASETA). This data set contains eight members (MA, MB, MC, MD, ME, MF, MG, and MH) and resides on either a 2314 or a 2319 volume.
- INOUTB DD defines a partitioned data set (DATASETB). This data set resides on either a 2314 or a 2319 volume and contains two members (MA and MJ).
- INOUTC DD defines a partitioned data set (DATASETC), which resides on either a 2314 or a 2319 volume. The data set contains four members (MF, ML, MM, and MN).
- INOUTD DD defines a partitioned data set (DATASETD). This data set resides on either a 2314 or a 2319 volume and contains two members (MM and MP).
- INOUTE DD defines a partitioned data set (DATASETE). This data set contains four members (MD, ME, MF, and MT) and resides on either a 2314 or a 2319 volume.
- INOUTX DD defines a partitioned data set (DATASETX). This data set is new and is to be kept after the copy operation. Five tracks are allocated for the data set on either a 2314 or a 2319 volume. Two blocks are allocated for directory entries.
- SYSUT3 DD defines a temporary spill data set. One track is allocated on either a 2314 or a 2319 volume.
- SYSUT4 DD defines a temporary spill data set. One track is allocated on either a 2314 or a 2319 volume.
- 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 back-up copy of DATASETA, which is subsequently compressed in place.

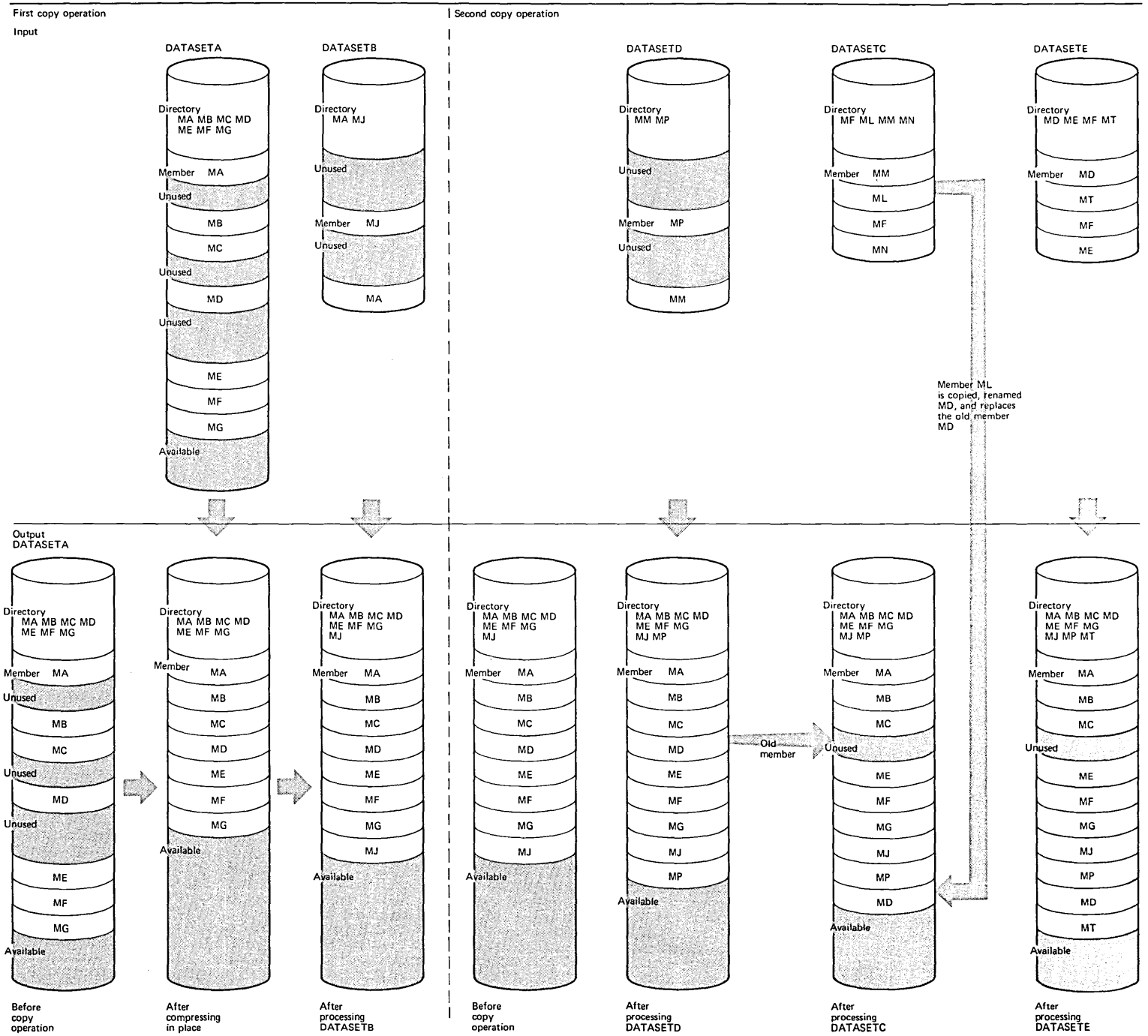


Figure 16. Multiple Copy Operations/Copy Steps

- The second COPY statement indicates the start of another copy operation. The absence of a SELECT or EXCLUDE statement causes a default to a full copy; however, the same DD statement, INOUTA, is specified for both the **INDD** and **OUTDD** parameters, causing a compress in place of the specified data set.
- INDD specifies INOUTB as the DD statement for the input data set (DATASETB) to be copied. Only member MJ is copied because member MA already exists on the output data set.
- The third COPY statement indicates the start of the third copy operation. The **OUTDD** parameter specifes INOUTA as the DD statement for the output data set (DATASETA). This copy operation contains more than one copy step.
- The first INDD statement specifies INOUTD as the DD statement for the first input data set (DATASETD) to be processed. Only member MP is copied to the output data set (DATASETA) because member MM is specified on the EXCLUDE statement.
- EXCLUDE specifies the member 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 INOUTC as the DD statement for the second input data set (DATASETC) to be processed. Member ML is searched for, found, and copied to the output data set (DATASETA). Member ML is copied even though its new name (MD) is identical to the name of a member (MD) that already exists on the output data set, because the replace option is specified for the renamed member.
- SELECT specifies the member to be selected from the input data set (DATASETC) to be copied to the output partitioned data set.
- The third INDD statement marks the beginning of the third copy step for this copy operation and specifies INOUTE as the DD statement for the last data set (DATASETE) to be copied. Only member MT is copied because the other members already exist on the output data set. Because the INDD statement is not followed by an EXCLUDE or SELECT statement, a full copy is performed.

The temporary spill data sets may or may not be opened, depending on the amount of main storage available; therefore, it is suggested that the SYSUT3 and SYSUT4 DD statements always appear in the job stream.

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

IEBCOPY Example 12

In this example, members are to be 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 17 shows the input and output data sets before and after processing.

The example follows:

```
//COPY     JOB      06#990,MCEWAN
//JOBSTEP  EXEC     PGM=IEBCOPY
//SYSPRINT DD       SYSOUT=A
//INOUTA   DD       DSNAME=DATASETA,UNIT=2311,VOL=SER=111113,
//               DISP=OLD
//INOUTB   DD       DSNAME=DATASETB,VOL=SER=111115,
//               DISP=(OLD,KEEP),UNIT=2311
//INOUTC   DD       DSNAME=DATASETC,VOL=SER=111114,
//               DISP=(OLD,KEEP),UNIT=2311
//INOUTD   DD       DSNAME=DATASETD,VOL=SER=111116,
//               DISP=OLD,UNIT=2311
//INOUTE   DD       DSNAME=DATASETE,VOL=SER=111117,
//               DISP=OLD,UNIT=2311
//SYSUT3   DD       UNIT=2311,SPACE=(TRK,(1))
//SYSUT4   DD       UNIT=2311,SPACE=(TRK,(1))
//SYSIN    DD       *
 COPY     OUTDD=INOUTA
               INDD=INOUTE
 SELECT   MEMBER=MA,MJ
               INDD=INOUTC
 EXCLUDE  MEMBER=MM,MN
 COPY     O=INOUTB,INDD=INOUTD
               I=((INOUTC,R),INOUTB)
 COPY     O=INOUTD,I=((INOUTB,R ))
 SELECT   MEMBER=MM
/*
```

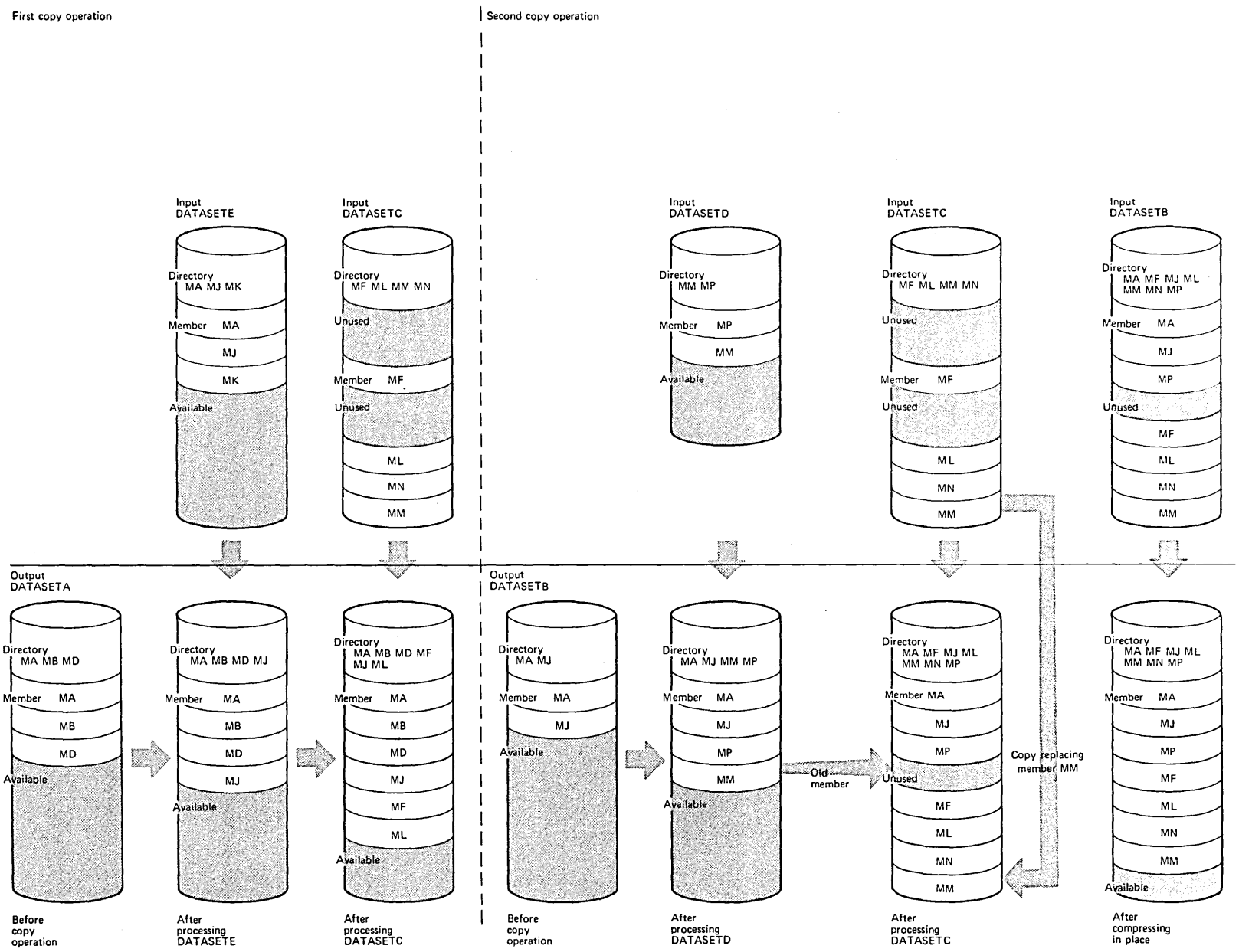


Figure 17. Multiple Copy Operations/Copy Steps Within a Job Step

The control statements are discussed below:

- INOUTA DD defines a partitioned data set (DATASETA). This data set contains three members (MA, MB, and MD) and resides on a 2311 volume.
- INOUTB DD defines a partitioned data set (DATASETB). This data set resides on a 2311 volume and contains two members (MA and MJ).
- INOUTC DD defines a partitioned data set (DATASETC), which resides on a 2311 volume. This data set contains four members (MF, ML, MM, and MN).
- INOUTD DD defines a partitioned data set (DATASETD). This data set resides on a 2311 volume and contains two members (MM and MP).
- INOUTE DD defines a partitioned data set (DATASETE), which resides on a 2311 volume. This data set contains three members (MA, MJ and MK).
- SYSUT3 DD defines a temporary spill data set. One track is allocated on a 2311 volume.
- SYSUT4 DD defines a temporary spill data set. One track is allocated on a 2311 volume.
- SYSIN DD defines the control data set, which follows in the input stream. The data set contains three COPY statements, SELECT and EXCLUDE statements, and several INDD statements.
- The first COPY statement indicates the start of a copy operation. The OUTDD operand specifies INOUTA as the DD statement for the output data set (DATASETA).
- The first INDD statement specifies INOUTE as the DD statement for the first input data set (DATASETE) to be processed. Processing occurs, as follows: (1) member MA is searched for and found, but is not copied because the replace option is not specified, and (2) member MJ is searched for, found, and copied to the output data set. Members are not searched for again after they are found.
- SELECT specifies the members (MA and MJ) to be selected from the input data set (DATASETE) to be copied.
- 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 INOUTC as the DD statement for the second input data set (DATASETC) to be processed. Members MF and ML, which are not named on the EXCLUDE statement, are copied because neither exists on the output data set.
- EXCLUDE specifies the members (MM and MN) to be excluded from the second copy operation.
- The second COPY statement indicates the start of another copy operation. The absence of a SELECT or EXCLUDE statement causes a default to a full copy. The **O (OUTDD)** parameter specifies INOUTB as the output data set (DATASETB). The **INDD** parameter specifies INOUTD as the first input data set (DATASETD) to be processed. Members MP and MM are copied to the output data set.
- INDD(I) specifies INOUTC as the DD statement for the second input data set (DATASETC) and INOUTB as the DD statement for the third input data set (DATASETB) to be processed. Members MF, ML, MM, and MN are copied from DATASETC. Member MM is copied, although it already exists on the output partitioned data sets, because the replace option is specified. Because DATASETB is also the data set specified in the **OUTDD** PARAMETER, a compress in place takes place. (The pointer in the output set directory is changed to point to the new (copied) member MM; thus the space occupied by the replaced member MM is embedded unused space.)
- The third COPY statement indicates the start of another copy operation. The **O (OUTDD)** parameter specifies INOUTD as the DD statement for the output data set (DATASETD). The **I (INDD)** parameter specifies INOUTB as the DD statement for the input data set (DATASETB).
- SELECT specifies the member (MM) to be selected from the input partitioned data set (DATASETB) to be copied. The replace option is specified on the data set level.

The temporary spill data sets may or may not be opened, depending on the amount of main storage available; therefore, it is suggested that the SYSUT3 and SYSUT4 DD statements always appear in the job stream.

Data sets used as input data sets in one copy operation can be used as output data sets in another copy operation, and vice versa.

IEBDG is a data set utility used to provide a *pattern* of test data to be used as a programming debugging aid. (See "Introduction" for general data set utility information.)

An output data set, containing records of any format, can be created through the use of utility control statements, with or without input data. An optional user exit is provided to pass control to a user routine to monitor each output record before it is written. Sequential, indexed sequential, and partitioned data sets can be used for input or output.

To generate test data, the user constructs a pattern of data that he can analyze quickly for predictable results. Test data is generated through the use of utility control statements.

When the user defines the contents of a field, he decides:

- What type of pattern—IBM-supplied or user-supplied—he wishes to place initially in the defined field.
- What action, if any, is to be performed to alter the contents of the field after it is selected for each output record.

IBM-Supplied Patterns

IBM supplies seven patterns: alphameric, alphabetic, zoned decimal, packed decimal, binary number, collating sequence, and random number. The user may choose one of them when he defines the contents of a field. All patterns except the binary 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:

ABCDEFGHIJKLMNOPQRSTUVWXYZ ABCDEFG. . .

Table 14 shows the IBM-supplied patterns.

Table 14. IBM-Supplied Patterns

Type	*Expressed in Hexadecimal*	*Expressed in Printable Characters*
Alphameric	C1 C2 . . .E9 F0. . .F9	ABC . . . Z 0 . . . 9
Alphabetic	C1 C2 . . . E9	ABC . . . Z
Zoned Decimal	F0F0 . . . F0F1	00 . . . 01
Packed Decimal	0000 . . . 001C (Positive pattern) 0000 . . . 001D (Negative pattern)	Not applicable
Binary Number	00 . . . 01 (Positive pattern) FF . . . FF (Negative pattern)	Not applicable
Collating Sequence	40 . . . F9	bc.<(+ | & !$*);¬-/,%_>?:#@' = " A . . . Z 0 . . . 9
Random Number	Random hexadecimal digits	Not applicable

Note: A packed decimal or binary number is right aligned in the defined field.

The user can specify a starting character when defining an alphameric, alphabetic, or collating sequence field. For example, a ten-byte alphabetic field for which "H" is specified as the starting character would appear as:

HIJKLMNOPQ

The same ten-byte alphabetic field with no specified starting character would appear as:

ABCDEFGHIJ

The user 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 Pictures

Instead of selecting an IBM-supplied pattern, the user can specify a picture to be placed in the defined field. The user can provide:

- An EBCDIC character string.
- A decimal number to be converted to packed decimal by IEBDG.
- A decimal number to be converted to binary by IEBDG.

When the user supplies a picture, he must specify a picture length that is equal to or less than the specified field length. An EBCDIC picture is left aligned in a defined field; a decimal number that is converted to packed decimal or to binary is right aligned in a defined field.

The user can initially load (fill) a defined field with either an EBCDIC character or a hexadecimal digit. For example, the 10-byte picture "BADCFEHGJI" is to be placed in a 15-byte field. An EBCDIC "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.) Remember that the fill character, if specified, is written in each byte of the defined field prior to the inclusion of an IBM-supplied pattern or user-supplied picture.

Modification of Selected Fields

IEBDG can be used to change the contents of a field in a specified manner. One of eight actions can be selected to change a field after its inclusion in each applicable output record. These actions are ripple, shift left, shift right, truncate left, truncate right, fixed, roll, and wave.

Figure 18 shows the effects of each of the actions on a six-byte alphabetic field. Note that the roll and wave actions are applicable only when a user pattern is supplied. In addition, the result of a ripple action depends on which type of pattern—IBM-supplied or user-supplied—is present.

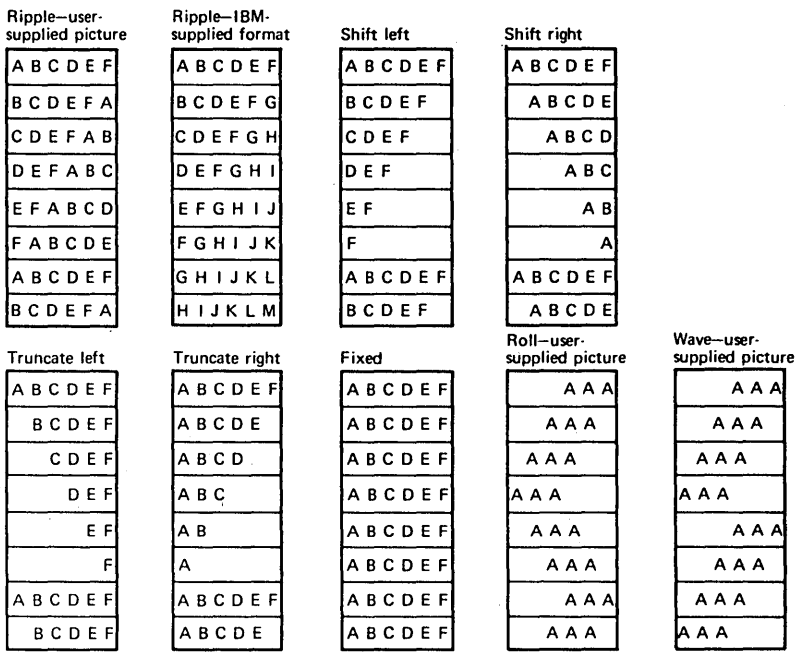


Figure 18. IEBDG Actions

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, which contains records that are to be used in the construction of an output data set or partitioned data set member. The input data sets are optional; that is, output records can be created entirely from utility control statements.
- A control data set, which contains any number of sets of utility control statements.

IEBDG produces the following output:

- An output data set, which 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, which contains informational messages, the contents of applicable utility control statements, and any error messages.

Note that input and output data sets may be sequential, indexed sequential, or partitioned data set members.

BDAM is not supported.

IEBDG produces a return code to indicate the results of program execution. The return codes and their meanings are:

- 00, which indicates successful completion.
- 04, which indicates that a user routine returned a code of 16 to IEBDG. The job step is terminated at the user's request.
- 08, which indicates that 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, which indicates that an error occurred while processing an input or output data set. The job step is terminated.
- 16, which indicates that an error occurred from which recovery is not possible. The job step is terminated.

Control

IEBDG is controlled by job control statements and utility control statements. The job control statements are used to execute or invoke 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 15 shows the job control statements necessary for using IEBDG.

Table 15. IEBDG Job Control Statements

Statement	*Use*
JOB	Initiates the job.
EXEC	Specifies the program name (PGM = IEBDG) or, if the job control statements reside in a procedure library, the procedure name. Additional information can be specified in the EXEC statement; see "PARM Information on the EXEC Statement" below.
SYSPRINT DD	Defines a sequential message data set. The data set can be written on a system output device, a tape volume, or a direct access volume.
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.
seqinset DD	Defines an optional sequential or indexed sequential data set used as input to IEBDG. The data set can reside on a tape volume or on a direct access volume. 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. Each DD statement is subsequently referred to by a DSD utility control statement.
parinset DD	Defines an optional input partitioned data set member residing on a direct access volume. 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. The "parinset" DD statement is referred to by a DSD utility control statement.
seqout DD	Defines an output (test) sequential or indexed sequential data set. Any number of "seqout" DD statements can be included per job step; however, only one "seqout" statement is applicable per set of utility control statements.
parout DD	Defines an optional output partitioned data set member to be created and placed on a direct access volume. Any number of "parout" DD statements (eachDD statement referring to the same or to a different data set) can be included per job step; however, only one "parout" statement is applicable per set of utility control statements.

The minimum region size required for using IEBDG varies with its use; see *OS Storage Estimates*, GC28-6551, for information on calculating the region size required for a particular application.

The SYSPRINT data set and the SYSIN data set can have any blocking factor.

Both input and output data sets can contain fixed, variable, or undefined records.

Refer to *OS Data Management Services Guide*, GC26-3746, for information on estimating space allocations.

The "seqinset" DD statement can be entered:

```
//seqinset DD  DSNAME = setname,UNIT = xxxx,DISP = (OLD,KEEP),
//             VOLUME = SER = xxxxxx,LABEL = (...,...),
//             DCB(applicable subparameters)
```

The LABEL parameter is included only for a magnetic tape volume. If the input data set has an indexed sequential organization, DSORG = IS should be coded in the DCB parameter.

The "parinset" DD statement can be entered:

```
//parinset DD  DSNAME = setname(membername),UNIT = xxxx,DISP = (OLD,
//             KEEP),VOLUME = SER = xxxxxx,
//             DCB = (applicable subparameters)
```

The "seqout" DD statement can be entered:

```
//seqout DD  DSNAME = setname, UNIT = xxxx,
//           DISP = (,KEEP),VOLUME = SER = xxxxxx,
//           DCB = (applicable subparameters)
```

The LABEL parameter is included for magnetic tape; the SPACE parameter is included for direct access.

The "parout" DD statement can be entered:

```
//parout DD  DSNAME = setname(membername),UNIT = xxxx,
//           VOLUME = SER = xxxxxx,DCB = (applicable
//           subparameters),DISP = (,KEEP),
//           SPACE = (applicable subparameter)
```

The SPACE parameter is included on the *parout* DD statement when creating the first member to be placed in a partitioned data set.

Restrictions

- The input data set record type must agree with the output data set record type.
- The DSORG subparameter must be included in the DCB subparameters if the input or output data set has an indexed sequential organization (DSORG = IS). If members of a partitioned data set are used, DSORG = PS or DSORG = PO may be coded. If the DSORG subparameter is not coded, DSORG = PS is assumed.
- If the SYSPRINT DD statement is omitted, no messages are written.
- On an MVT system, the ddname of the "seqout" DD statement should not be SYSPRINT.
- For an indexed sequential data set, the key length must be specified in the DCB.

PARM Information on the 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, coded as follows:

```
PARM = LINECNT = nnnn
```

The nnnn is a four-digit decimal number that specifies the number of lines (0000 to 9999) to be printed per page of output listing.

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).

Note: If IEBDG is invoked, the line-count option can be passed in a parameter list that is referred to by a subparameter of the LINK or ATTACH macro instruction. In addition, a page count can be passed in a six-byte parameter list that is referred to by a suparameter of the LINK or ATTACH macro instruction. For a discussion of linkage conventions, refer to "Appendix B: Invoking Utility Programs from a Problem Program."

Utility Control Statements

IEBDG is controlled by the following utility control statements:

- DSD statement, which specifies the ddnames of the input and output data sets. One DSD statement must be included for each *set* of utility control statements.
- FD statement, which defines the contents and lengths of fields to be used in creating output records.
- CREATE statement, which defines the contents of output records.
- REPEAT statement, which specifies the number of times a CREATE statement or a group of CREATE statements are to be used in generating output records.
- END statement, which 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.

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 format of the DSD statement is:

```
[label] DSD  OUTPUT = (ddname)
             [,INPUT = (ddname,...)]
```

where:

OUTPUT = (*ddname*)
specifies the ddname of the DD statement defining the output data set.

INPUT = (*ddname,...*)
specifies the ddname of a DD statement defining a data set 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 card, they must begin in column four.

Note: The ddname SYSIN must not be coded in the **INPUT** parameter. 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 19 shows how fields defined in FD statements are placed in buffer areas so that subsequent CREATE statements can assign selected fields to specific output records.

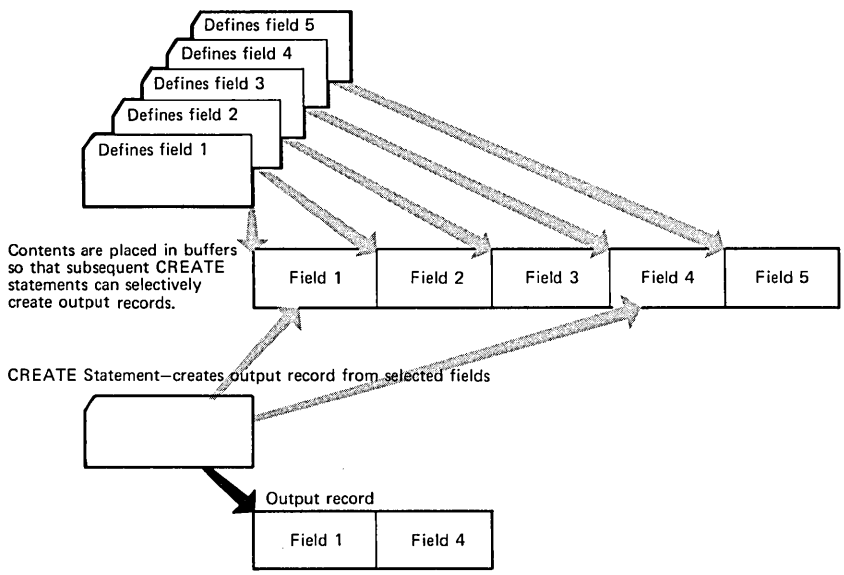


Figure 19. Defining and Selecting Fields for Output Records Using IEBDG

Figure 20 shows how the FD statement is used to specify a field in an input record to be used in output records. The left-hand side of the figure shows that a field in the input record beginning at byte 50 is selected for use in the output record. The right-hand side of the figure shows that the field is to be placed at byte 20 in the output record.

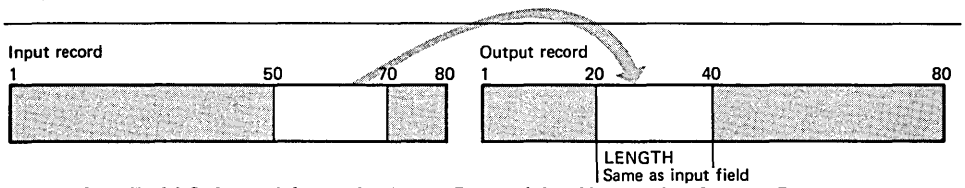


Figure 20. Field Selected from the Input Record for Use in the Output Record

The format of the FD statement is:

```
[label] FD  NAME = name
          ,LENGTH = length-in-bytes
          [,STARTLOC = starting-byte-location]
          [,FILL = {'character'                }]
                   {X'2-hexadecimal-digits'}
          [,FORMAT = pattern[,,CHARACTER = character]]
          [,PICTURE = length,{'character-string'   }
                             {P'decimal-number' }
                             {B'decimal-number' }]
          [,SIGN = sign]
          [,ACTION = action]
          [,INDEX = number[,CYCLE = number][,RANGE = number]]
          [,INPUT = ddname]
          [,FROMLOC = number]
```

where:

NAME = *name*
specifies the name of the field defined by this FD statement.

LENGTH = *length-in-bytes*
specifies the length in bytes of the defined field. For variable records, four bytes of length descriptor are added.

STARTLOC = *starting-byte-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, etc. If **STARTLOC** is omitted, 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 =
specifies an EBCDIC character or hexadecimal digits to be placed in each byte of the defined field prior to any other operation in the construction of a field. If **FILL** is omitted, binary zeros are placed in the field. These values can be coded:

'character'
specifies an EBCDIC character to be placed in the defined field.

X'*2-hexadecimal-digits***'**
specifies two hexadecimal digits (for example, FILL = X'40' or FILL = X'FF') to be placed in each byte of the defined field.

FORMAT =
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:

AN
specifies an alphameric pattern.

ZD
specifies a zoned decimal pattern.

PD
specifies a packed decimal pattern.

CO
specifies a collating-sequence pattern.

BI
specifies a binary pattern.

AL
specifies an alphabetic pattern.

RA
specifies a random binary pattern.

CHARACTER = *character*
specifies the starting character of a field. If **CHARACTER** is omitted, the starting character is as described under "IBM-Supplied Patterns" earlier.

PICTURE =
specifies the length and contents of a user-supplied field picture. **PICTURE** must not be used when **FORMAT** is used. These values can be coded:

length
specifies the number of characters in the FD picture.

'*character-string*'
specifies an EBCDIC character string that is to be placed in the defined field. The character string is left aligned in the field. A character string may be broken in column 71 and must be continued in column 4. Double quotation marks must not be coded to represent a single quotation mark within a character string.)

P'*decimal-number*'
specifies a decimal number that is to be converted to packed decimal and placed right aligned in the defined field.

B'*decimal-number*'
specifies a decimal number that is to be converted to binary and placed right aligned in the defined field. In all cases, the number of characters within the quotation marks must equal the number specified in the *length* subparameter.

SIGN = *sign*
specifies a mathematical sign (+ or -), which is used when defining a packed decimal or binary field. If **SIGN** is omitted, the sign is assumed to be positive.

ACTION = *action*
specifies that the contents of a defined field are to be altered after the field's inclusion in an output record. These values can be coded:

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.

RO
specifies that the contents of a defined field are to be rolled after the field's inclusion in an output record. **RO** can be specified only for a user-defined field.

WV
specifies that the contents of a defined field are to be waved after the field's inclusion in an output record. **WV** can be specified only for a user-defined field.

FX
specifies that the contents of a defined field are to be fixed after the field's inclusion in an output record. If **ACTION** is omitted, **FX** is assumed.

RP
specifies that the contents of a defined field are to be rippled after the field's inclusion in an output record.

INDEX = *number*
specifies a number to be added to this field whenever a specified number of records have been written. If **INDEX** is omitted, no indexing is performed. These additional values can be coded:

CYCLE = *number*
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 might appear as 111 222 333 444 etc. This parameter can be coded only when **INDEX** is coded. 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.

RANGE = *number*
specifies an absolute value which the contents of this field can never exceed. If an index operation attempts to exceed the specified absolute value, the contents of the field as of the previous index operation are used.

INPUT = *ddname*
specifies the ddname for the input data set.

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 field begins at the first byte. (For variable records, significant data begins on the first byte after the four-byte length descriptor.)

Some of the FD keywords do not apply when certain patterns or pictures are selected by the user; for example, the **INDEX**, **CYCLE**, **RANGE**, and **SIGN** parameters are used only with numeric fields. Figure 21 shows which IEBDG keywords can be used with the applicable pattern or picture chosen by the user. Each keyword should appear no more than once on any FD statement.

FORMAT/PICTURE — Compatible Operations

Format	Action
AL AN CO	SL SR TL TR FX RP
Format	
ZD PD BI	Index Cycle Range Sign*
Picture	
PD BI	Index Cycle Range Sign
Picture	**Action**
EBCDIC	SL SR TL TR FX RP WV RO

*Zoned decimal numbers (ZD) do not include a sign.

Figure 21. Compatible IEBDG Operations

The CREATE statement defines the contents of a record (or records) to be made available to a user routine or to be written directly as an output record (or records).

The format of the CREATE statement is:

```
[label] CREATE {[QUANTITY = number]}
               {[,FILL =  {'character'}
                          {X'2-hexadecimal-digits'}]}
               {[,INPUT =   {ddname }
                          {SYSIN[(cccc)] }]}
               {[,PICTURE = length,startloc, {'character-string'  }
                                             {P'decimal-number' }
                                             {B'decimal-number' }]}
               {[,NAME =  {name }
                          {(name1,namen...)}
                          {(name,(COPY = name1,namen...))}]}
               {[,EXIT = routinename]}
```

where:

QUANTITY = *number*
specifies the number of records that this CREATE statement is to generate; each record is specified by the other parameters. 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, 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.

FILL =
specifies a value that is to be placed in each byte of the output record before any other operation in the construction of record. If **FILL** is not coded, binary zeros are placed in the output record. These values can be coded:

'character'
specifies an EBCDIC character that is to be placed in each byte of the output record.

X'*2-hexadecimal-digits***'**
specifies two hexadecimal digits (for example, FILL = X'40', or FILL = X'FF') to be placed in each byte of the output record.

INPUT =
defines an input data set whose records are to be used in the construction of output records. If **INPUT** is not coded, the output records are created entirely from utility control statements. 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. These values can be coded:

ddname
specifies the ddname of a DD statement defining an input data set.

SYSIN[(cccc)]
specifies that the SYSIN data set (input stream) 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). When INPUT = SYSIN is coded, the input records are delimited from any additional utility control statements by a record containing $$$E in columns 1 through 4. If "(cccc)" coded, the input records are delimited by a record containing EBCDIC characters beginning in column 1; the cccc can be any combination of from one to four EBCDIC characters.

PICTURE =
specifies the length, starting byte, and contents of a user-supplied picture (CREATE statement picture). 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 picture will occupy.

startloc
specifies a starting byte (within any applicable output record) in which the picture is to begin.

'character-string'
specifies an EBCDIC character string that is to be placed in the applicable record(s). The character string is left aligned at the defined starting byte. A character string may be broken in column 71 and continued in column 4.

P'*decimal-number***'**
specifies a decimal number that is to be converted to packed decimal and placed right aligned (within the boundaries of the defined length and starting byte) in the output records.

B'*decimal-number***'**
specifies a decimal number that is to be converted to binary and placed right aligned (within the boundaries of the defined length and starting byte) in the output records.

NAME =
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:

(name1,...)
specifies the name or names of a field or fields to be included in the applicable output record(s). Each field is included in an output record in the order in which its name is encountered in the CREATE statement.

COPY = *number*
indicates that all fields named in the inner parentheses (maximum of twenty) are to be treated as a group and included the specified number of times in each output record produced by this CREATE statement. Any number of sets of inner parentheses can be included with **NAME**; however, sets of parentheses cannot be embedded. Within each set of inner parentheses, **COPY** must appear before the name of any field.

EXIT = *routinename*
specifies the name of a user routine that is to receive control from IEBDG before writing each output record.

After processing each potential output record, the user routine provides a return code to instruct IEBDG how to handle the output record. The user codes are:

- 00, which specifies that the record is to be written.
- 04, which specifies that 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 ten output records are to be generated and a user exit indicates that two records are to be skipped, 12 input records are processed.
- 12, which specifies that 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, which specifies that all processing is to halt.

Note: When an exit routine is loaded and when the user returns control to IEBDG, register one contains the address of the first byte of the output record. Each keyword should appear no more than once on any CREATE statement.

Figure 22 shows the addition of field X to two different records. In this example, field x does not have a special starting location. 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.

The user can also indicate that a numerical 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 user-specified number to a field.

The CREATE statement constructs an output record by referring to previously defined fields by name and/or by providing a picture to be placed in the record. The user can generate multiple records with a single CREATE statement.

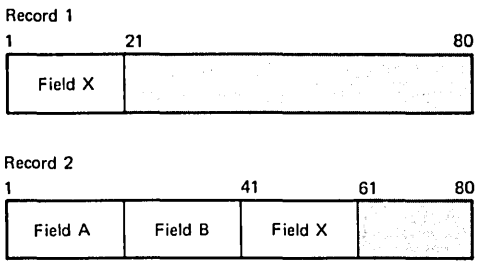


Figure 22. Default Placement of Fields Within an Output Record Using IEBDG

When defining a picture in a CREATE statement, the user must specify its length and starting location in the output record. The specified length must be equal to the number of specified EBCDIC or numeric characters. (When a specified decimal number is converted to packed decimal or binary, it is automatically right aligned.)

Figure 23 shows three ways in which output records can be created from utility control statements.

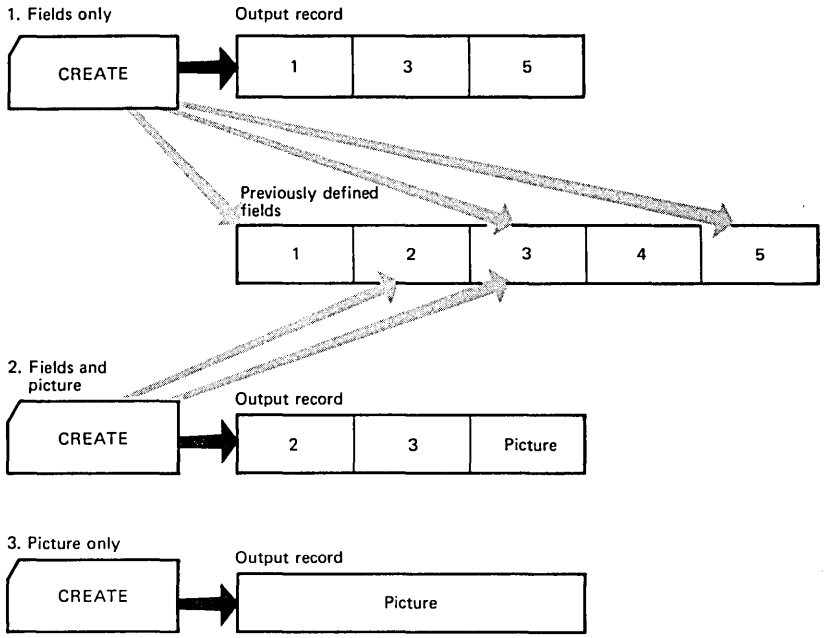


Figure 23. Creating Output Records with Utility Control

As an alternative to creating output records from utility control statements alone, the user can provide input records, which can be modified and written as output records. Input records can be provided directly in the input stream, or in a data set.

As previously mentioned, the CREATE statement is responsible for the construction of an output record. 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. An input record, if any is provided, is left aligned in the output record.
3. FD fields, if any, are placed in the output record in the order of the appearance of their names in the CREATE statement.
4. A CREATE statement picture, if any, is placed in the output record.

IEBDG provides a user exit so that the user can provide his own routine to analyze or further modify a newly constructed record before it is placed in the output data set.

A set of utility control statements contains one DSD statement, any number of FD, CREATE, and REPEAT statements, and one END statement when the **INPUT** parameter is omitted from the FD card.

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 this case, defined fields for field selection are not usable across sets of utility control statements. The FD card may be duplicated and used in more than one set of utility control statements within the job step.

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.

Figure 24 shows a group of five CREATE statements repeated *n* times.

The format of the REPEAT statement is:

```
[label] REPEAT QUANTITY = number[,CREATE = number]
```

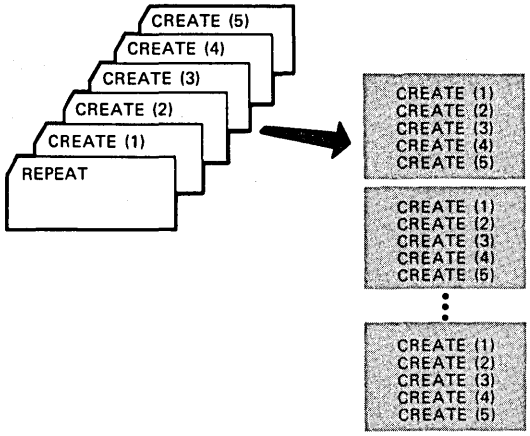


Figure 24. Repetition Due to the REPEAT Statement Using IEBDG

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 following CREATE statements to be included in the group. If the **CREATE** parameter is omitted, only one CREATE statement is repeated.

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 and a single output data set.

The format of the END statement is:

```
[label] END
```

IEBDG Examples

The following examples illustrate some of the uses of IEBDG. Table 16 can be used as a quick reference guide to IEBDG examples. The numbers in the "Example" column point to examples that follow.

Table 16. IEBDG Example Directory

Operation	*Data Set Organization*	*Device*	*Comments*	*Example*
Place binary zeros in selected fields.	Sequential	9-track tape	Blocked input and output.	1
Ripple alphabetic pattern	Sequential	9-track tape, 2314 Disk	Blocked input and output.	2
Create output records from utility control statements	Sequential	2314 Disk	Blocked output.	3
Modify records from partitioned members and input stream	Partitioned, Sequential	2314 Disk	Reblocking is performed. Each block of output records contains ten modified partitioned input records and two input stream records.	4
Create partitioned members for utility control statements	Partitioned	2314 Disk	Blocked output. One set of utility control statements per member.	5
Roll and wave user-supplied patterns	Sequential	2311 Disk	Output records are created from utility control statements.	6
Create indexed sequential data set using field selection and data generation	Sequential, Indexed sequential	2314 Disk	Output records are created by augmenting selected input fields with generated data.	7

IEBDG Example 1

In this example, binary zeros are to be placed in two fields of 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.

The example follows:

```
//CLEAROUT JOB  ,,MSGLEVEL=1
//         EXEC PGM=IEBDG
//SYSPRINT DD   SYSOUT=A
//SEQIN    DD   DSNAME=INSET,UNIT=2400,DISP=(OLD,KEEP),
// DCB=(RECFM=FB,LRECL=80,BLKSIZE=800),
//               VOLUME=SER=240000,LABEL=(,NL)
//SEQOUT   DD   DSNAME=OUTSET,UNIT=2400,VOLUME=SER=240001,
//               DCB=(RECFM=FB,LRECL=80,BLKSIZE=800),
//    .          DISP=(,KEEP),LABEL=(,NL)
//SYSIN    DD   *
       DSD      OUTPUT=(SEQOUT),INPUT=(SEQIN)
       FD       NAME=FIELD1,INPUT=SEQIN,LENGTH=80
       FD       NAME=FIELD2,LENGTH=10,STARTLOC=20
       FD       NAME=FIELD3,LENGTH=10,STARTLOC=50
       CREATE   QUANTITY=100,INPUT=SEQIN,NAME=(FIELD1,        ⟩C
               FIELD2,FIELD3)
       END
/*
```

The control statements are discussed below:

- SEQIN DD defines a sequential input data set (INSET). The data set was originally written on a 9-track, 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 FD statement defines an 80 byte field of input data.
- The second and third FD statements create two ten-byte fields (FIELD and FILED3) that contain 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, and FIELD3) 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.

IEBDG Example 2

In this example, a ten-byte alphabetic pattern is to be rippled. At the end of the job step the first output record contains "ABCEDFGHIJ", 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, etc.

The example follows:

```
//RIPPLE   JOB  ,,MSGLEVEL=1
//         EXEC PGM=IEBDG
//SYSPRINT DD   SYSOUT=A
//SEQIN    DD   DSNAME=INSET,DISP=(OLD,KEEP),VOL=SER=240000,
// DCB=(RECFM=FB,LRECL=80,BLKSIZE=800),UNIT=2400
//SEQOUT   DD   DSNAME=OUTSET,UNIT=2314,VOLUME=SER=231400,
//           DCB=(RECFM=FB,LRECL=80,BLKSIZE=800),
//              DISP=(,KEEP),SPACE=(TRK,(20,10))
//SYSIN    DD   *
 DSD      OUTPUT=(SEQOUT),INPUT=(SEQIN)
 FD        NAME=FIELD1,INPUT=SEQIN,LENGTH=80
 FD        NAME=FIELD2,LENGTH=10,FORMAT=AL,ACTION=RP,          %C
               STARTLOC=1
 CREATE   QUANTITY=100,INPUT=SEQIN,NAME=(FIELD1,FIELD2)
 END
/*
```

The control statements are discussed below:

- SEQIN DD defines an input sequential data set (INSET). The data set was originally written on a 9-track, standard labeled tape volume.
- SEQOUT DD defines the test output data set (OUTSET). Twenty tracks of primary space and ten tracks of secondary space are allocated for the sequential data set on a 2314 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 first FD statement defines an 80 byte field of input data.
- The second FD statement creates a ten-byte field in which the pattern ABCEDFGHIJ is 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.

IEBDG Example 3

In this example, output records are to be created entirely from utility control statements. Three fields are to be created and used in the construction of the output records. In two of the fields, alphabetic data is to be truncated; the other field is a numeric field that is to be indexed by one after each output record is written. Figure 25 shows the contents of the output records at the end of the job step.

Field 1	Field 2	Field 3 (packed decimal)	
1	31	61 71	80
ABCDEFGHIJKLMNOPQRSTUVWXYZABCD	ABCDEFGHIJKLMNOPQRSTUVWXYZABCD	FF . . . FF	123 . . . 90
BCDEFGHIJKLMNOPQRSTUVWXYZABCD	ABCDEFGHIJKLMNOPQRSTUVWXYZABC	FF . . . FF	123 . . . 91
CDEFGHIJKLMNOPQRSTUVWXYZABCD	ABCDEFGHIJKLMNOPQRSTUVWXYZAB	FF . . . FF	123 . . . 92
DEFGHIJKLMNOPQRSTUVWXYZABCD	ABCDEFGHIJKLMNOPQRSTUVWXYZA	FF . . . FF	123 . . . 93
EFGHIJKLMNOPQRSTUVWXYZABCD	ABCDEFGHIJKLMNOPQRSTUVWXYZ	FF . . . FF	123 . . . 94

Figure 25. Output Records at Job Step Completion

The example follows:

```
//UTLYONLY JOB  ,,MSGLEVEL=1
//         EXEC PGM=IEBDG
//SYSPRINT DD   SYSOUT=A
//SEQOUT   DD   DSNAME=OUTSET,UNIT=2311,DISP=(,KEEP),
//              DCB=(RECFM=FB,LRECL=80,BLKSIZE=800),
//              SPACE=(TRK,(20,10)),VOLUME=SER=240000
//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=TL
 FD       NAME=FIELD3,LENGTH=10,STARTLOC=71,PICTURE=10,       ¢C
              P'1234567890',INDEX=1
 CREATE QUANTITY=100,NAME=(FIELD1,FIELD2,FIELD3),FILL=X'FF'
 END
/*
```

The control statements are discussed below:

- SEQOUT DD defines the test output data set. Twenty tracks of primary space and ten tracks of secondary space are allocated for the sequential data set on a 2311 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 ten-byte field containing a packed decimal number (1234567890) to be incremented 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.
- END signals the end of a set of utility control statements.

IEBDG Example 4

In this example, two partitioned members and input records from the input stream are to be used as the basis of a partitioned output member. Each block of 12 output records is to contain ten modified records from an input partitioned member and two records from the input stream. Figure 26 shows the content of the output partitioned member at the end of the job step.

The example follows:

```
//MIX      JOB  ,,MSGLEVEL=1
//         EXEC PGM=IEBDG
//SYSPRINT DD   SYSOUT=A
//PARIN1   DD   DSNAME=INSET1(MEMBA),UNIT=2314,DISP=OLD,
//              DCB=(RECFM=FB,LRECL=80,BLKSIZE=800,DSORG=PS),
//              VOLUME=SER=231400
//PARIN2   DD   DSNAME=INSET2(MEMBA),UNIT=2314,DISP=OLD,
//              DCB(RECFM=FB,LRECL=80,BLKSIZE=960,DSORG=PS),
//              VOLUME=SER=231401
//PAROUT   DD   DSNAME=PARSET(MEMBA),UNIT=2314,DISP=(,KEEP),
//              VOLUME=SER=231402,SPACE=(TRK,(20,10,5)),
//              DCB(RECFM=FB,LRECL=80,BLKSIZE=960,DSORG=PS)
//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
/*
```

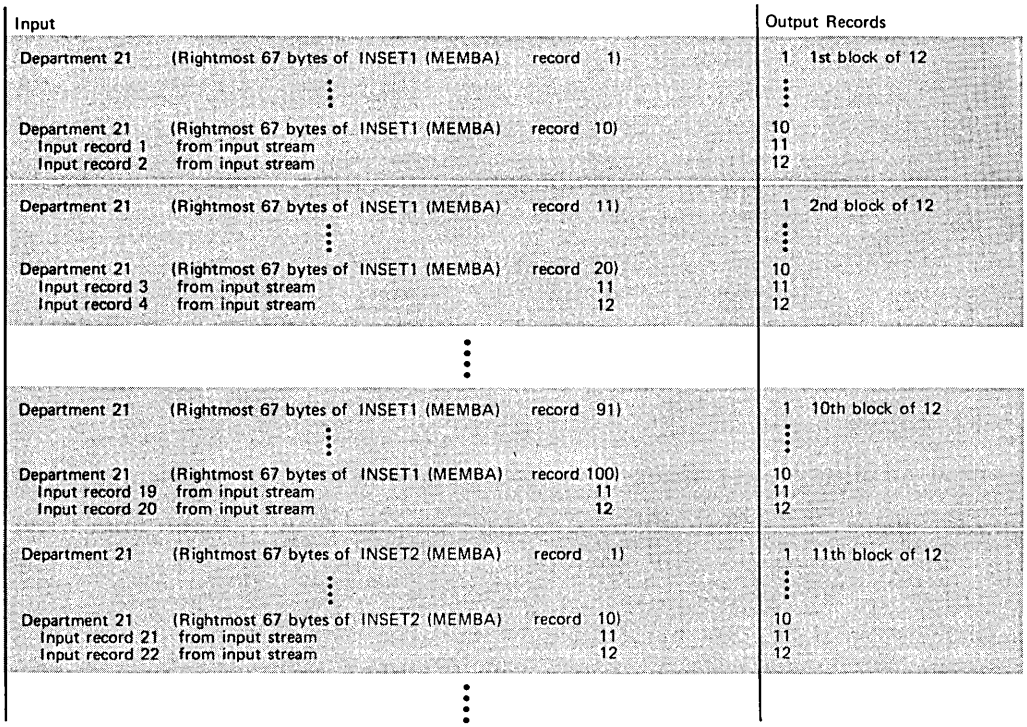


Figure 26. Output Partitioned Member at Job Step Completion

The control statements are discussed below:

- PARIN1 DD defines one of the input partitioned members.
- PARIN2 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 prior to 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 ten times.
- The first CREATE statement creates ten 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.

IEBDG Example 5

In this example, output records are to be created from three sets of utility control statements and written in three partitioned data set members. Four fields are to be created and used in the construction of the output records. In two of the fields (FIELD1 and FIELD3), alphabetic data is to be shifted. The other two fields are to be fixed alphameric and zoned decimal fields. Figure 27 shows the partitioned data set members at the end of the job step.

MEMBA

Field 1 1	Field 3 31	Field 2 51	Binary zeros 71 80
ABCDEFGHIJKLMNOPQRSTUVWXYZABCD	ABCDEFGHIJKLMNOPQRST	0000000010000000001	fill
BCDEFGHIJKLMNOPQRSTUVWXYZABCD	ABCDEFGHIJKLMNOPQRS	0000000010000000001	fill
CDEFGHIJKLMNOPQRSTUVWXYZABCD	ABCDEFGHIJKLMNOPQR	0000000010000000001	fill
DEFGHIJKLMNOPQRSTUVWXYZABCD	ABCDEFGHIJKLMNOPQ	0000000010000000001	fill

MEMBB

Field 3 1	Field 3 21	Field 3 41	Field 2 61 80
ABCDEFGHIJKLMNOPQRST	ABCDEFGHIJKLMNOPQRST	ABCDEFGHIJKLMNOPQRST	0000000010000000001
ABCDEFGHIJKLMNOPQRS	ABCDEFGHIJKLMNOPQRS	ABCDEFGHIJKLMNOPQRS	0000000010000000001
ABCDEFGHIJKLMNOPQR	ABCDEFGHIJKLMNOPQR	ABCDEFGHIJKLMNOPQR	0000000010000000001
ABCDEFGHIJKLMNOPQ	ABCDEFGHIJKLMNOPQ	ABCDEFGHIJKLMNOPQ	0000000010000000001

MEMBC

Field 4 1	Field 1 31	Binary zeros 61 80
ABCDEFGHIJKLMNOPQRSTUVWXYZ0123	ABCDEFGHIJKLMNOPQRSTUVWXYZABCD	fill
ABCDEFGHIJKLMNOPQRSTUVWXYZ0123	BCDEFGHIJKLMNOPQRSTUVWXYZABCD	fill
ABCDEFGHIJKLMNOPQRSTUVWXYZ0123	CDEFGHIJKLMNOPQRSTUVWXYZABCD	fill
ABCDEFGHIJKLMNOPQRSTUVWXYZ0123	DEFGHIJKLMNOPQRSTUVWXYZABCD	fill

Figure 27. Partitioned Data Set Members at Job Step Completion

The example follows:

```
//UTSTS     JOB  ,,MSGLEVEL=1
//·         EXEC PGM=IEBDG
//SYSPRINT  DD   SYSOUT=A
//PAROUT1   DD   DSNAME=PARSET(MEMBA),UNIT=2314,DISP=(,KEEP),
//               VOLUME=SER=231400,SPACE=(TRK,(10,10,5)),
//               DCB=(RECFM=FB,LRECL=80,BLKSIZE=800,DSORG=PS),
//PAROUT2   DD   DSNAME=PARSET(MEMBB),UNIT=AFF=PAROUT1,
//               DCB=(RECFM=FB,LRECL=80,BLKSIZE=800,DSORG=PS),
//               DISP=OLD,VOLUME=SER=231400
//PAROUT3   DD   DSNAME=PARSET(MEMBC),UNIT=AFF=PAROUT1,
//               DCB=(RECFM=FB,LRECL=80,BLKSIZE=800,DSORG=PS),
//               DISP=OLD,VOLUME=SER=231400
//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=((COPY=3,FIELD3),FIELD2)
 END
 DSD       OUTPUT=(PAROUT3)
 CREATE    QUANTITY=4,NAME=(FIELD4,FIELD1)
 END
/*
```

The control statements are discussed below:

- PAROUT1 DD defines the first member (MEMBA) of the partitioned output data set. This example assumes that the partitioned data set does not exist prior to 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, 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 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.

IEBDG Example 6

In this example, ten fields containing user-supplied EBCDIC pictures are to be 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 28 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	D D 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	D 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	D 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	D 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	D D	CCC

Figure 28. Contents of Output Records at Job Step Completion

The example follows:

```
//ROLLWAVE JOB  ,,MSGLEVEL=1
//         EXEC PGM=IEBDG
//SYSPRINT DD   SYSOUT=A
//OUTSET   DD   DSNAME=SEQSET,UNIT=2311,DISP=(,KEEP),
//              VOLUME=SER=2311,SPACE=(TRK,(20,10)),
//              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,     ¢C
               FIELD5,FIELD6,FIELD7,FIELD8,FIELD9,FIELD10)
 END
/*
```

The control statements are discussed below:

- OUTSET DD defines the output sequential data set on a 2311 volume. Twenty tracks of primary space and ten tracks of secondary space are allocated to the 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 statement defining the output data set.
- FD defines a field to be used in the subsequent construction of output records. Note that the direction and frequency of the initial roll or wave depends 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.

IEBDG Example 7

In this example, the first ten bytes of the output record contain zoned decimal format generated data. This field serves as the key field for the output record in the output indexed sequential data set. The key field is incremented (indexed) by one for each record. The input sequential data set provides an additional 80-byte field to complete the output record.

The example follows:

```
//CREATEIS JOB  MSGLEVEL=1
//BEGIN    EXEC PGM=IEBDG
//TAPEIN   DD   DCB=(BLKSIZE=80,LRECL=80,RECFM=F),
//              DISP=(OLD,KEEP),UNIT=2400,LABEL=(,SL),
//              DSNAME=TAPEIT,VOL=SER=MASTER
//DISKOUT  DD   DCB=(BLKSIZE=270,LRECL=90,RECFM=FB,
//              DSORG=IS,NTM=2,OPTCD=MY,RKP=0,KEYLEN=10,
//              CYLOFL=1),UNIT=2314,SPACE=(CYL,1),
//              VOL=SER=231400,DSNAME=CREATIS,DISP=(NEW,KEEP)
//SYSPRINT DD   SYSOUT=A
//SYSIN    DD   *
  DSD     OUTPUT=(DISKOUT),INPUT=(TAPEIN)
  FD      NAME=DATAFD,LENGTH=80,FROMLOC=1,STARTLOC=11,        ℛC
                INPUT=TAPEIN
  FD      NAME=KEYFD,LENGTH=10,STARTLOC=1,FORMAT=ZD,INDEX=1
  CREATE  INPUT=TAPEIN,NAME=(KEYFD,DATAFD)
  END
/*
```

The control statements are discussed below:

- TAPEIN DD defines the sequential input data set.
- DISKOUT DD defines the indexed sequential output 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 statement defining the output data set.
- FD defines a field that will be used in the subsequent construction of output records. The first FD statement in this example defines and locates an 80-byte field of input data. The data is field selected from one of the input logical records and placed at start location 11 of the output logical record. The second FD statement defines and locates the ten-byte key field.
- CREATE constructs a 90-byte output record by referring to the previously defined fields.
- END signals the end of a set of utility control statements.

IEBEDIT Program

IEBEDIT is a data set utility used to create an output data set containing a selection of jobs or job steps. (See "Introduction" for general data set utility information.) At a later time, the data set can be used as an input stream for job processing.

IEBEDIT creates an output job stream by editing and selectively copying a job stream provided as input. The program can copy:

- An entire job or jobs, including JOB statements and any associated JOBLIB statements.
- Selected job steps, including the JOB statement and any associated JOBLIB statement.

All selected JOB statements, JOBLIB statements, jobs, or job steps are placed in the output data set in the same order as they exist in the input data set. Note that a JOBLIB statement is copied only if it follows a selected JOB statement.

When IEBEDIT encounters a selected job step containing an input record having the characters "..*" in columns 1 through 3, the program automatically converts that record into a termination statement (/* statement) and places it in the output data set.

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.

IEBEDIT provides a return code to indicate the results of program execution. The return codes and their meanings are:

- 00, which indicates successful completion.
- 04, which indicates that an error occurred. The output data set may not be usable as a job stream. Processing continues.
- 08, which indicates that an unrecoverable error occurred while attempting to process the input, output, or control data set. The job step is terminated.

Control

IEBEDIT is controlled by job control statements and utility control statements. The job control statements are required to execute or invoke 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 17 shows the job control statements necessary for using IEBEDIT.

The minimum region size that can be specified for IEBEDIT is 10K.

Restrictions

- The block size for the SYSPRINT data set must be a multiple of 121. 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 block sizes.

Utility Control Statement

IEBEDIT is controlled through the EDIT utility control statement.

EDIT Statement

The EDIT statement indicates which step or steps of a specified job in the input data set are to be included in the output data set. Any number of EDIT statements can be included in an operation, thus including selected jobs in the output data set.

EDIT statements must be included in the same order as the input jobs that they represent. If no EDIT statement is present in the control data set, the entire input data set is copied.

Table 17. IEBEDIT Job Control Statements

Statement	Use
JOB	Initiates the job.
EXEC	Specifies the program name (PGM = IEBEDIT) or, if the job control statements reside in a procedure library, the procedure name.
SYSPRINT DD	Defines a sequential message data set. 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, a tape volume, or a 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.

The format of the EDIT statement is:

```
[label] EDIT  [START = jobname]
              [,TYPE = {POSITION }]
                       {INCLUDE  }
                       {EXCLUDE  }
              [,STEPNAME = ( {name-name } [,{name-name }],...)
                           {name       } [,{name       }]]
              [,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. 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 =
specifies the contents of the output data set. If **TYPE** is omitted, TYPE = POSITION is assumed. 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 it. All job steps preceding the specified step are omitted from the operation. This 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 jobs steps belonging to job except those steps specified in the **STEPNAME** parameter.

STEPNAME =
specifies the first job step to be placed in the output data set when coded with TYPE = POSITION. 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. If **NOPRINT** is omitted, the resultant output is listed in the message data set.

Note: Any JOBLIB DD statement that follows a selected JOB statement is automatically copied to the output data set.

IEBEDIT Examples

The following examples show some of the uses of IEBEDIT. Table 18 can be used as a quick-reference guide to IEBEDIT examples. The numbers in the "Example" column point to examples that follow.

Table 18. IEBEDIT Example Directory

Operation	Devices	Comments	Example
COPY	9-track tape	The input data set contains three jobs. One job is to be copied.	1
COPY	7-track tape	The output data set is the second data set on the volume. One job step is to be copied from each of three jobs.	2
COPY	2311 Disk, 9-track tape	Include a job step from one job and exclude a job step from another job.	3
COPY	2314 or 2319 Disk[1]	Latter portion of a job stream is to be copied.	4
COPY	9-track tape	All records in the input data set are to be copied. The ..* record is converted to a /* statement in the output data set.	5

[1] The 2319 disk is functionally equivalent to the 2314 disk; to use the 2319, specify 2314 in the control statement.

IEBEDIT Example 1

In this example one job (JOBA), including all of its job steps (A, B, C, and D), is to be copied into the output data set. The input data set contains three jobs: JOBA, which has four job steps; JOBB, which has three job steps; and JOBC, which has two job steps.

The example follows:

```
//EDIT1    JOB  09#440,SMITH
//         EXEC PGM=IEBEDIT
//SYSPRINT DD   SYSOUT=A
//SYSUT1   DD   UNIT=2400,DISP=(OLD,KEEP),VOLUME=SER=001234
//SYSUT2   DD   UNIT=2400,DISP=(NEW,KEEP),VOLUME=SER=001235,
// DCB=(RECFM=F,LRECL=80,BLKSIZE=80),
// DSNAME=OUTTAPE
//SYSIN    DD   *
         EDIT  START=JOBA
/*
```

The control statements are discussed below:

- SYSUT1 DD defines the input data set. The data set resides on a 9-track, standard labeled tape volume (001234).
- SYSUT2 DD defines the output data set. The data set is to reside as the first data set on a standard labeled, 9-track tape volume (001235).
- SYSIN DD defines the control data set, which follows in the input stream.
- EDIT indicates that JOBA is to be copied in its entirety.

IEBEDIT Example 2

This example copies: (1) the JOB statement and steps STEPC and STEPD for JOBA, (2) the JOB statement and STEPE for JOBB, and (3) the JOB statement and STEPJ for JOBC. The input data set contains three jobs: JOBA, which includes STEPA, STEPB, STEPC, and STEPD; JOB B, which includes STEPE, STEPF, and STEPG; and JOBC, which includes STEPH and STEPJ.

The example follows:

```
//EDIT2    JOB  09#440,SMITH
//         EXEC PGM=IEBEDIT
//SYSPRINT DD   SYSOUT=A
//SYSUT1   DD   DISP=(OLD,KEEP),VOLUME=SER=001234,
// UNIT=2400-2
//SYSUT2   DD   DSNAME=OUTSTRM,UNIT=2400-2,DISP=(NEW,KEEP),
// DCB=(DEN=1,RECFM=F,LRECL=80,BLKSIZE=80,
// TRTCH=C),LABEL=(2,SL)
//SYSIN    DD   *
         EDIT  START=JOBA,TYPE=INCLUDE,STEPNAME=STEPC,STEPD
         EDIT  START=JOBB,TYPE=INCLUDE,STEPNAME=STEPE
         EDIT  START=JOBC,TYPE=INCLUDE,STEPNAME=STEPJ
/*
```

The control statements are discussed below:

- SYSUT1 DD defines the input data set. The data set resides on a 7-track, standard labeled tape volume (001234).
- SYSUT2 DD defines the output data set. The data set is to reside as the second data set on a 7-track, standard labeled tape volume (001235).
- SYSIN DD defines the control data set, which follows in the input stream.
- The EDIT statements copy the indicated JOB statements and job steps.

IEBEDIT Example 3

This example copies: (1) the JOB statement and steps STEPF and STEPG for JOBB and (2) the JOB statement and STEPH, excluding STEPJ, for JOBC. 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.

The example follows:

```
//EDIT3    JOB  09#440,SMITH
//         EXEC PGM=IEBEDIT
//SYSPRINT DD   SYSOUT=A
//SYSUT1   DD   DSNAME=INSET,UNIT=2314,DISP=(OLD,KEEP),
//  VOLUME=SER=231400
//SYSUT2   DD   DSNAME=OUTTAPE,UNIT=2400,LABEL(,NL),
//  DCB=(DEN=2,RECFM=F,LRECL=80,BLKSIZE=80),
//  DISP=(,KEEP)
//SYSIN    DD   *
         EDIT  START=JOBB,TYPE=INCLUDE,STEPNAME=STEPF-STEPG
         EDIT  START=JOBC,TYPE=EXCLUDE,STEPNAME=STEPJ
/*
```

The control statements are discussed below:

- SYSUT1 DD defines the input data set. The data set resides on a 2314 volume (231400).
- SYSUT2 DD defines the output data set. The data set is to reside as the first or only data set on an unlabeled, 9-track (800 bits per inch) tape volume.
- SYSIN DD defines the control data set, which follows in the input stream.
- The EDIT statements copy selected JOB statements and job steps.

IEBEDIT Example 4

This example copies the JOBA JOB statement, the job step STEPF, and all the steps that follow it. The input data set contains one job (JOBA), which includes STEPA, STEPB, . . . STEPL. Job steps STEPA through STEPE are not included in the output data set.

The example follows:

```
//EDIT4    JOB  09#440,SMITH
//         EXEC PGM=IEBEDIT
//SYSPRINT DD   SYSOUT=A
//SYSUT1   DD   DSNAME=INSTREAM,UNIT=2314,DISP=(OLD,KEEP),
//              VOLUME=SER=231400
//SYSUT2   DD   DSNAME=OUTSTREM,UNIT=2314,DISP=(,KEEP),
// DCB=(RECFM=F,LRECL=80,BLKSIZE=80),
// VOLUME=SER=231401,SPACE=(TRK,2)
//SYSIN    DD   *
         EDIT  START=JOBA,TYPE=POSITION,STEPNAME=STEPF
/*
```

The control statements are discussed below:

- SYSUT1 DD defines the input data set. The data set resides on a 2314 or 2319 volume (231400).
- SYSUT2 DD defines the output data set. The data set is to reside on a 2314 volume (231401). 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 and job steps STEPF through STEPL.

IEBEDIT Example 5

This example copies the entire input (SYSUT1) data set. The record containing the characters "..*" in columns 1 through 3 is converted to a " /*" statement in the output data set.

The example follows:

```
//EDIT5    JOB  09#440,SMITH
//         EXEC PGM=IEBEDIT
//SYSPRINT DD   SYSOUT=A
//SYSUT2   DD   DSNAME=OUTTAPE,UNIT=2400,VOLUME=SER=001234,
// DCB=(RECFM=F,LRECL=80,BLKSIZE=80),DISP=(NEW,KEEP)
//SYSIN    DD   DUMMY
//SYSUT1   DD   DATA
//BLDGDGIX JOB
//         EXEC PGM=IEHPROGM
//SYSPRINT DD   SYSOUT=A
//DD1      DD   UNIT=2311,VOLUME=SER=111111,DISP=OLD
//SYSIN    DD   *
        BLDG   INDEX=A.B.C,ENTRIES=10,EMPTY
..*
/*
```

The control statements are discussed below:

- SYSUT2 DD defines the output data set. The data set is to reside as the first data set on a 9-track 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 terminated when the termination statement (/*) is encountered.

IEBGENER Program—Class C

IEBGENER is a data set utility used to copy a sequential data set or a partitioned member, or to create a partitioned data set from a sequential or partitioned member used as input. (See "Introduction" for general data set utility information.) IEBGENER can be used to expand an existing partitioned data set by creating partitioned members and merging them into the data set that is to be expanded.

IEBGENER provides optional editing facilities and exits for user routines that process labels, manipulate input data, create keys, and handle permanent input/output errors. Refer to "Appendix A: Exit Routine Linkage" for a discussion of linkage conventions that are applicable when user routines are provided.

IEBGENER can be used to:

- Create a backup copy of a sequential data set or a partitioned member.
- Produce a partitioned data set from sequential input.
- Expand a partitioned data set.
- Produce an edited sequential or partitioned data set.
- Reblock or change the logical record length of a data set.
- Create user labels on sequential output data sets.

At the completion or termination of IEBGENER, the highest return code encountered within the program is passed to the calling program.

Creating a Backup Copy

A backup copy of a sequential data set or partitioned member can be produced 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 direct access to tape, etc.

A data set that resides on a direct access volume can be copied to its own volume, provided that its data set name is changed. A partitioned data set cannot reside on a magnetic tape volume.

Producing a Partitioned Data Set from Sequential Input

IEBGENER can be used to produce a partitioned data set from sequential output. Through the use of utility control statements, the user can logically divide the sequential data set into *record groups* and assign member names to the record groups. IEBGENER places the newly created members in a partitioned output data set.

Note: A partitioned data set cannot be produced if an input or output data set contains spanned records.

Figure 29 shows how a partitioned data set is produced from a sequential data set used as input. The left-hand 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 right-hand side of the figure shows the partitioned data set produced from the sequential input.

Expanding a Partitioned Data Set

An expanded data set is a data set into which an additional member or members have been merged. IEBGENER creates the members from sequential input and places them in the data set being expanded. The merge operation—the ordering of the partitioned directory—is automatically performed by the program.

Figure 30 shows how sequential input is converted into members that are merged into an existing partitioned data set. The left-hand side of the figure shows the sequential input that is to be merged with the partitioned data set shown in the middle of the figure. 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 right-hand side of the figure shows the expanded partitioned data set. Note that members B, D, and E from the sequential data set were placed in *available space* and that they are sequentially ordered in the partitioned directory.

Producing an Edited Data Set

IEBGENER can be used to produce an edited sequential or partitioned data set. Through the use of utility control statements, the user can specify editing information that applies to a record, a group of records, selected groups of records, or an entire data set.

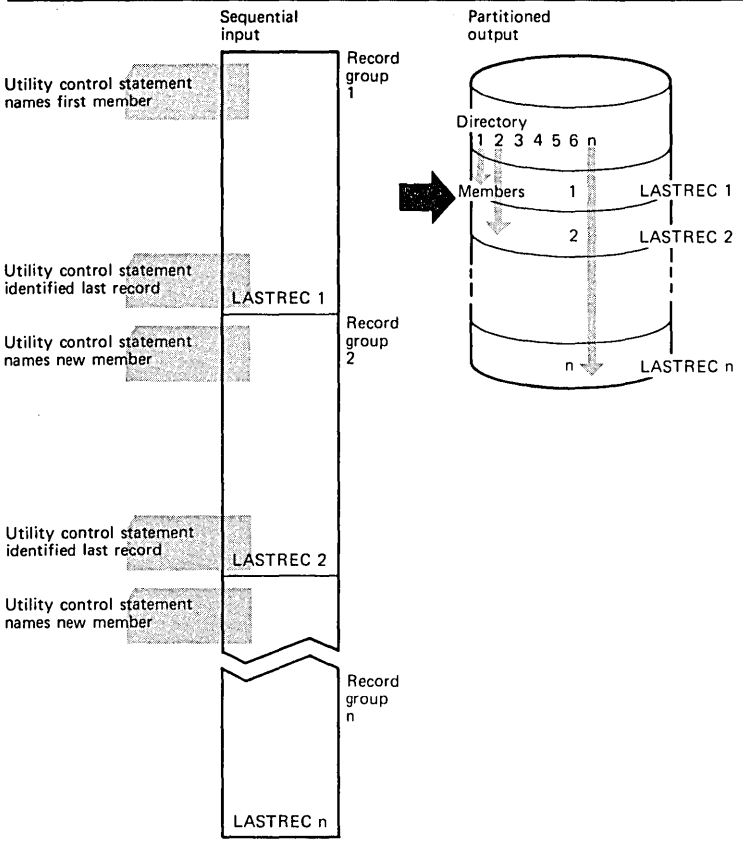


Figure 29. Creating a Partitioned Data Set from Sequential Input Using IEBGENER

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 H-set BCD to EBCDIC mode.

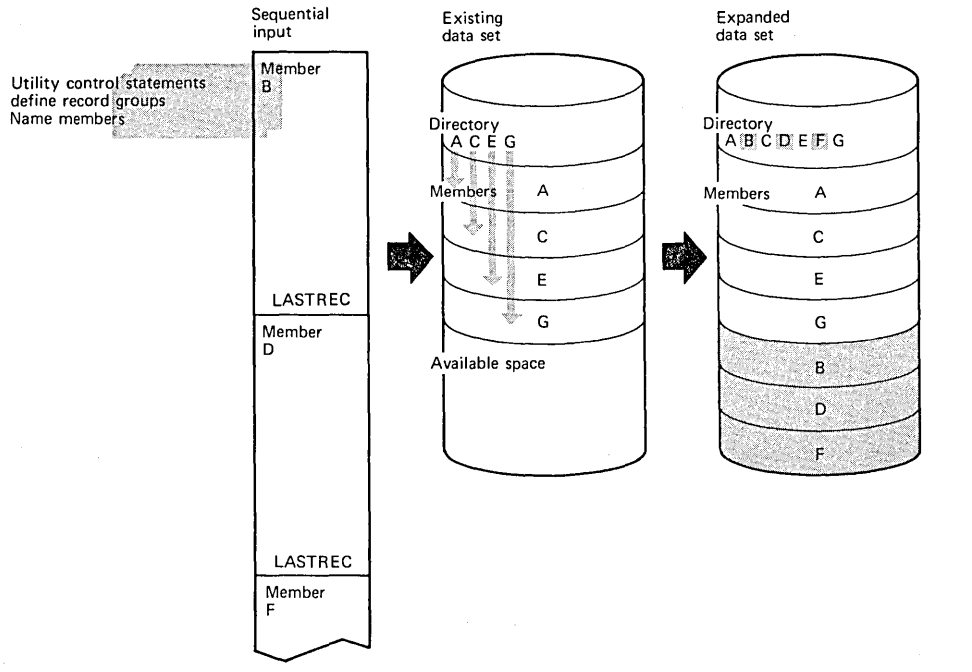


Figure 30. Expanding a Partitioned Data Set Using IEBGENER

Figure 31 shows part of an edited sequential data set. The left-hand side of the figure 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 right-hand side of the figure shows the data set after editing.

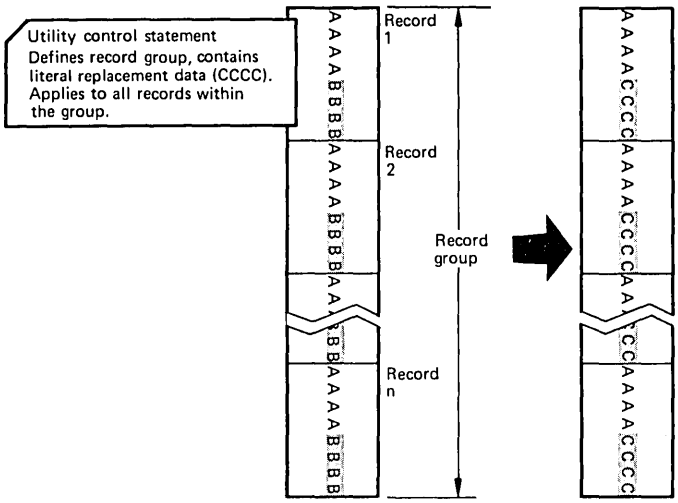


Figure 31. Editing a Sequential Data Set Using IEBGENER

Note: IEBGENER cannot be used to edit a data set if the input and output data sets consist of VS or VBS records and have equal block sizes and logical record lengths. In this case, any utility control statements that specify editing are ignored; IEBGENER performs a *straight copy*; that is, for each physical record read from the input data set, the utility writes an unedited physical record on the output data set.

Reblocking or Changing Logical Record Length

IEBGENER can be used to produce a reblocked output data set containing either fixed or variable records. In addition, the program can produce an output data set having a logical record length that differs from the input logical record length.

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 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.
- 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.

IEBGENER produces the following output:

- An output data set, which can be either sequential or partitioned. The output data set can be either a new data set (created during the current job step) or an existing partitioned data set that is to be expanded.
- A message data set, which contains informational messages (for example, the contents of utility control statements) and any error messages.

IEBGENER provides a return code to indicate the results of program execution. The return codes and their meanings are:

- 00, which indicates successful completion.
- 04, which indicates probable successful completion. A warning message is written.
- 08, which indicates that processing was terminated after the user requested processing of user header labels only.
- 12, which indicates an unrecoverable error. The job step is terminated.
- 16, which indicates that a user routine passed a return code of 16 to IEBGENER. The job step is terminated.

Control

IEBGENER is controlled by job control statements and utility control statements. The job control statements are required to execute or invoke IEBGENER 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 IEBGENER.

Job Control Statements

Table 19 shows the job control statements necessary for using IEBGENER.

Table 19. IEBGENER Job Control Statements

Statement	Use
JOB	Initiates the job.
EXEC	Specifies the program name (PGM = IEBGENER) or, if the job control statements reside in a procedure library, the procedure name.
SYSPRINT DD	Defines a sequential message data set. The data set can be written to a system output device, a tape volume, or a direct access volume.
SYSUT1 DD	Defines the input data set. It can define a sequential data set or a member of a partitioned data set.
SYSUT2 DD	Defines the output data set. It can define a sequential data set, a member of a partitioned data set, or a partitioned data set.
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 within a library of partitioned members.

The minimum region size that can be specified for the execution of IEBGENER is 14K + b, where b is the largest block size in the job step rounded to the next higher 2K.

IEBGENER always uses two buffers regardless of what was specified in the DCB.

If both the SYSUT1 and the SYSUT2 DD statements specify standard user labels (SUL), IEBGENER copies user labels from SYSUT1 to SYSUT2. See "Appendix E: Processing User Labels" for a discussion of the available options for user label processing.

Both the input data set and the output data set can contain fixed, variable, undefined, or variable spanned records. These records can be reblocked by the specification of a new maximum block length on the SYSUT2 DD statement. During reblocking, if the output data set resides on a direct access volume:

- For fixed or variable records, keys can be retained only by using the appropriate user exit.
- For variable spanned records, keys can never be retained.

When the input/output data set has fixed length, variable length, or variable spanned records, the block size, the logical record length, and the record format are required. When the input/output data set has undefined records, only the block size is required.

Refer to *OS Data Management Services Guide*, GC26-3746, for information on estimating space allocations.

Restrictions

- 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. The block size for the SYSIN data set must be a multiple of 80. Any blocking factor can be specified for these block sizes.
- Space must be allocated for an output data set (SYSUT2 DD statement) that is to reside on a direct access device unless the data set is an expanded data set, in which case space must not be allocated.
- DCB parameters in a SYSUT2 DD statement defining an expanded partitioned data set must be compatible with the specifications made when the data set was originally created.
- Concatenated data sets with unlike attributes are not allowed as input to IEBGENER. For information on concatenated data sets, see *OS Data Management Services Guide*, GC26-3746.
- The SYSIN DD statement is required for each use of IEBGENER.
- RECFM (except for undefined data sets), BLKSIZE, and LRECL (except for undefined data sets) must be specified on the SYSUT1/SYSUT2 DD statement when the data set is new and when the data set is a dummy data set, a card punch, or a printer.
- When neither RECFM, BLKSIZE, nor LRECL are present for the input data set, these values are copied from the input data set.
- Always specify the output BLKSIZE when the LRECL and RECFM (except for U) is specified. The default RECFM is U for the output data set. The output LRECL must

be present when editing is to be performed and the RECFM is either FB, VS, or VBS. In other cases a default LRECL value is generated by IEBGENER.

- The input data set must always have a BLKSIZE parameter. The default RECFM is U for the input data set. The input LRECL must be specified when RECFM is either VS, VBS, or FB. In other cases a default LRECL is generated by IEBGENER.

Utility Control Statements

IEBGENER is controlled by utility control statements. The statements and the order in which they must appear are:

- GENERATE statement, which is used to indicate the number of member names and alias names, record identifiers, literals, and editing information contained in the control data set.
- EXITS statement, which is used to indicate that user routines are provided.
- LABELS statement, which is used to specify user-label processing.
- MEMBER statement, which is used to specify the member name and alias of a member of a partitioned data set to be created.
- RECORD statement, which is used to define a record group to be processed and to supply editing information.

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 user exits are provided, an EXITS statement is used.

When the output is to be partitioned, one GENERATE statement, one MEMBER statement per output member, and RECORD statements, as required, are used. If user exits are provided, an EXITS statement is used.

GENERATE Statement

The GENERATE statement is used when: (1) output is to be partitioned, (2) editing is to be performed, or (3) user routines are provided and/or label processing is specified.

The GENERATE statement must appear before other statements. If it contains errors or is inconsistent with other statements, IEBGENER is terminated.

The format of the GENERATE statement is:

```
[label] GENERATE [MAXNAME = n]
                 [,MAXFLDS = n]
                 [,MAXGPS = n]
                 [,MAXLITS = n]
```

where:

MAXNAME = *n*
specifies a number that is no less than 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 that is no less than 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 that is no less than the total number of **IDENT** parameters appearing in subsequent RECORD statements. **MAXGPS** is required if there are any **IDENT** parameters in subsequent RECORD statements.

MAXLITS = *n*
specifies a number that is no less than the total number of characters contained in the **FIELD** literals of subsequent RECORD statements. **MAXLITS** is required if the **FIELD** parameters of subsequent RECORD statements contain literals. **MAXLITS** does not pertain to literals used in **IDENT** parameters.

EXITS Statement

The EXITS statement is used to identify exit routines supplied by the user. Linkages to and from exit routines are discussed in "Appendix A: Exit Routine Linkage."

The EXITS statement is used when user routines are provided.

The format of the EXITS statement is:

```
[label] EXITS  [INHDR = routinename]
               [,OUTHDR = routinename]
               [,INTLR = routinename]
               [,OUTTLR = routinename]
               [,KEY = routinename]
               [,DATA = routinename]
               [,IOERROR = routinename]
               [,TOTAL = (routinename,size)]
```

where:

INHDR = *routinename*
specifies the symbolic name of a routine that processes user input header labels.

OUTHDR = *routinename*
specifies the symbolic name of a routine that creates user output header labels. **OUTHDR** is ignored if the output data set is partitioned.

INTLR = *routinename*
specifies the symbolic name of a routine that processes user input trailer labels.

OUTTLR = *routinename*
specifies the symbolic name of a routine that processes user output trailer labels. **OUTTLR** is ignored if the output data set is partitioned.

KEY = *routinename*
specifies the symbolic name of a routine that creates the output record key. (This routine does not receive control when a data set consisting of VS or VBS type records is processed because no processing of keys is permitted for this type of data.)

DATA = *routinename*
specifies the symbolic name of a routine that modifies the physical record (logical record for VS or VBS type records) before it is processed by IEBGENER.

IOERROR = *routinename*
specifies the symbolic name of a routine that handles permanent input/output error conditions.

TOTAL =
specifies that exits to a user's routine are to be provided prior to writing each record. The keyword OPTCD = T must be specified for the SYSUT2 DD statement. **TOTAL** is valid only when the utility is used to process sequential data sets. These values must be coded:

routinename
specifies the name of a user-supplied totaling routine.

size
specifies the number of bytes needed to contain totals, counters, pointers, etc.

For a detailed discussion of the processing of user labels as data set descriptors, and for discussion of user label totaling), refer to "Appendix E: Processing User Labels."

LABELS Statement

The LABELS statement specifies whether or not user labels are to be treated as data by IEBGENER. For a detailed discussion of this option, refer to "Processing User Labels as Data," in "Appendix E: Processing User Labels."

The LABELS statement is used when the user wants to specify that: (1) no user labels are to be copied to the output data set, (2) user labels are to be copied to the output data set from records in the data portion of the SYSIN data set, or (3) 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 format of the LABELS statement is:

```
[label] LABELS [DATA = {YES   }
                       {NO    }
                       {ALL   }
                       {ONLY  }
                       {INPUT }]
```

where:

DATA =
specifies whether 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 user's label-processing routine are to be treated as data. Processing of labels as data ends in compliance with standard return codes. If no value is entered, **YES** is assumed.

NO
specifies that user labels are not to be treated as data.

ALL
specifies that user labels in the group currently being processed are to be treated as data regardless of any return code. A return code of 16 causes IEBGENER to complete processing the remainder of the group of user labels and to terminate 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 terminates 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.

Note: LABELS DATA = NO must be specified to make standard user label (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 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 aliases of a member that is to be created.

All RECORD statements following a MEMBER statement pertain to the member named in that MEMBER statement. If no MEMBER statements are included, the output data set is organized sequentially.

The format of the MEMBER statement is:

```
[label] MEMBER NAME = (name[,alias]...)
```

where:

NAME = (*name*[,*alias*]...)
specifies a member name followed by a list of its aliases. If only one name appears in the statement, it 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: (1) the output is to be partitioned, (2) editing is to be performed, or (3) 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 may appear in the control statement stream for IEBGENER.

Within a RECORD statement, one **IDENT** 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.

The format of the RECORD statement is:

```
[label] RECORD {[IDENT = (length,'name',input-location)]}
               {[,FIELD = ([length], [input-location ],[conversion],
                          [output-location])...],['literal']}
               {[,LABELS = n]}
```

where:

IDENT =
identifies the last record of the input group to which the **FIELD** parameters or MEMBER statement applies. If the RECORD statement is not followed by additional RECORD or MEMBER statements, **IDENT** also defines the last record to be processed. If **IDENT** is omitted, the remainder of the input data is considered to be in one record group; subsequent RECORD and MEMBER statements are ignored. These values can be coded:

length
specifies the length (in bytes) of the identifying name. The length cannot exceed eight characters.

'name'
specifies the exact literal that identifies the last input record of a record group. 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 location of the field that contains the identifying name in the input records.

FIELD =
specifies field-processing and editing information. Only the contents of specified fields in the input record is copied to the output record. 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 bytes is assumed. If a literal is to be processed, a length of 40 bytes or less must be specified.

input-location
specifies the starting byte of the field to be processed. If *input-location* is not specified, byte 1 is assumed.

'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.

conversion
specifies a two-byte 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:

PZ
specifies that data (packed decimal) is to be converted to unpacked decimal data.

ZP
specifies that data (unpacked decimal) is to be converted to packed decimal data.

HE
specifies that data (H-set BCD) is to be converted to EBCDIC.

output-location
specifies the starting location of this field in the output records. If *output-location* is not specified, byte 1 is assumed.

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 1 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.

If *conversion* is specified in **FIELD**, the following restrictions apply:

- PZ-type (packed-to-unpacked) conversion is impossible for packed decimal records longer than 16K bytes.
- For ZP-type (unpacked-to-packed) conversion, the normal 32K-byte maximum applies.

- 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 prior to 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 BCD to EBCDIC) specification, L.

If both output header labels and output trailer labels are to be contained in the SYSIN data set, the user must include one RECORD statement (including the **LABELS** parameter), indicating the number of input records to be treated as user labels, for header labels and one for 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 detailed discussion of the LABELS option, refer to "Processing User Labels As Data," in "Appendix E: Processing User Labels."

Note: IDENT and **FIELD** parameters are ignored in straight copy processing of data sets that contain VS or VBS records.

IEBGENER Examples

The examples that follow illustrate some of the uses of IEBGENER. Table 20 can be used as a quick reference guide to IEBGENER examples. The numbers in the "Example" column point to the examples that follow.

Table 20. IEBGENER Example Directory

Operation	Data Set Organization	Devices	Comments	Example
COPY	Sequential	Card Reader, Tape	Blocked output.	1
Copy—with editing	Sequential	Card Reader, Tape	Blocked output.	2
COPY—with editing	Sequential	Card Reader, Tape	Blocked output. Input includes // cards.	3
COPY—with editing	Sequential	Card Reader, 2311 Disk	Blocked output. Input includes // cards.	4
PRINT	Sequential	Card Reader, Printer	Input includes // cards. System output device is a printer	5
CONVERT	Sequential input, Partitioned output	Tape, 2314 Disk	Blocked output. Three members are to be created,	6
Copy—with editing	Sequential	2301 Drum	Blocked output. Two members are to be merged into existing data set.	7
COPY—with editing	Sequential	Tape	Blocked output. Data set edited as one record group.	8
COPY—with editing	Sequential	2314 Disk	Blocked output. New record length specified for output data set. Two record groups specified.	9
COPY—with editing	Sequential	Tape	Blocked output. Data set edited as one record group.	10

IEBGENER Example 1

In this example, a card-input, sequential data set is to be copied to a 9-track tape volume.

The example follows:

```
//CDTOTAPE JOB  09#660,SMITH
//         EXEC PGM=IEBGENER
//SYSPRINT DD   SYSOUT=A
//SYSIN    DD   DUMMY
//SYSUT2   DD   DSNAME=OUTSET,UNIT=2400,LABEL=(,SL),
// DISP=(,KEEP),VOLUME=SER=001234,
// DCB=(RECFM=FB,LRECL=80,BLKSIZE=2000)
//SYSUT1   DD   *
```

(input card data set)

```
/*
```

The control statements are discussed below:

- SYSIN DD defines a dummy data set. No editing is to be performed; therefore, no utility control statements are needed.
- SYSUT2 DD defines the output data set. The data set is written to a 9-track tape volume at a density of 800 bits per inch. The data set is to reside as the first (or only) data set on the volume.
- SYSUT1 DD defines the card-input data set. The data set can contain no // cards.

IEBGENER Example 2

In this example, a card-input, sequential data set is to be copied to a 7-track tape volume. The control data set is a member of a partitioned data set.

The example follows:

```
//CDTOTAPE JOB  09#660,SMITH
//         EXEC PGM=IEBGENER
//SYSPRINT DD   SYSOUT=A
//SYSIN    DD   DSNAME=CNTRLIBY(STMNTS),UNIT=2311,
// DISP=(OLD,KEEP),VOLUME=SER=111112,
// DCB=(RECFM=F,LRECL=80,BLKSIZE=80)
//SYSUT2   DD   DSNAME=OUTSET,UNIT=2400-2,LABEL=(,SL),
// DCB=(DEN=1,RECFM=FB,LRECL=80,BLKSIZE=2000,
// TRTCH=C),DISP=(,KEEP),VOLUME=SER=001234
//SYSUT1   DD   *
```

(input card data set)

```
/*
```

The control statements are discussed below:

- SYSIN DD defines the control data set, which contains the utility control statements. The control statements reside as a member, STMNTS, in a partitioned data set.
- SYSUT2 DD defines the output data set. The data set is written as the first or only data set on the volume. It is written at 556 bits per inch density on a 7-track tape volume.
- SYSUT1 DD defines the card-input data set. The data set can contain no // cards.

IEBGENER Example 3

In this example, a card-input, sequential data set is to be copied to a 9-track tape volume. The input contains cards that have slashes (//) in columns 1 and 2. The control data set is a member of a partitioned data set.

The example follows:

```
//CDTOTAPE JOB  09#660,SMITH
//         EXEC PGM=IEBGENER
//SYSPRINT DD   SYSOUT=A
//SYSIN    DD   DSNAME=CNTRLIBY(STMNTS),UNIT=2314,
// DISP=(OLD,KEEP),VOLUME=SER=111112,
// DCB=(RECFM=F,LRECL=80,BLKSIZE=80)
//SYSUT2   DD   DSNAME=OUTSET,UNIT=2400,LABEL=(2,SL),
// VOLUME=SER=001234,DCB=(RECFM=FB,
// LRECL=80,BLKSIZE=2000),DISP=(,KEEP)
//SYSUT1   DD   DATA
```

(input card data set, including // cards)

```
/*
```

The control statements are discussed below:

- SYSIN DD defines the data set containing the utility control statements. The statements reside as a member, STMNTS, in a partitioned data set.
- SYSUT2 DD defines the copied sequential data set (output). The data set is written as the second data set on the specified tape volume.
- SYSUT1 DD defines the card-input data set. The data set is to be edited as specified in the utility control statements (not shown). The input data set contains // cards.

IEBGENER Example 4

In this example, a card-input, sequential data set is to be copied to a 2311 volume. The input data set contains // cards.

The example follows:

```
//CDTOTAPE JOB  09#660,SMITH
//         EXEC PGM=IEBGENER
//SYSPRINT DD   SYSOUT=A
//SYSIN    DD   DSNAME=CNTRLIBY(STMNTS),UNIT=2311,
// DISP=(OLD,KEEP),VOLUME=SER=111112,
// DCB=(RECFM=F,LRECL=80,BLKSIZE=80)
//SYSUT2   DD   DSNAME=OUTSET,UNIT=2311,VOLUME=SER=111113,
// DISP=(,KEEP),SPACE=(TRK,(20,10)),
// DCB=(RECFM=FB,LRECL=80,BLKSIZE=2000)
//SYSUT1   DD   DATA
```

(input card data set, including // cards)

```
/*
```

The control statements are discussed below:

- SYSIN DD defines the control data set, which contains the utility control statements. The control statements reside as a member, STMNTS, in a partitioned data set.
- SYSUT2 DD defines the output data set. Twenty tracks of primary storage space and ten tracks of secondary space are allocated for the data set on a 2311 volume.
- SYSUT1 DD defines the card-input data set. The data set is to be edited as specified in the utility control statements (not shown). The input data set contains // cards.

IEBGENER Example 5

In this example, the content of a card data set is to be printed. The printed output is to be left aligned, with one 80-byte record appearing on each line of printed output.

The example follows:

```
//CDTOPTR  JOB  09#660,SMITH
//         EXEC PGM=IEBGENER
//SYSPRINT DD   SYSOUT=A
//SYSIN    DD   DUMMY
//SYSUT2   DD   SYSOUT=A,DCB=(RECFM=F,LRECL=80,BLKSIZE=80)
//SYSUT1   DD   DATA
```

(input card data set, including // cards)

```
/*
```

The control statements are discussed below:

- SYSIN DD defines a dummy data set. No editing is to be performed; therefore, no utility control statements are required.
- SYSUT2 DD indicates that the output is to be written on the system output device (printer). Carriage control can be specified by changing the RECFM = F subparameter to RECFM = FA.
- SYSUT1 DD defines the input card data set. The input data set contains // cards.

IEBGENER Example 6

In this example, a partitioned data set (consisting of three members) is to be created from sequential input.

The example follows:

```
//TAPEDISK JOB  09#660,SMITH
//         EXEC PGM=IEBGENER
//SYSPRINT DD   SYSOUT=A
//SYSUT1   DD   DSNAME=INSET,UNIT=2400,LABEL=(,SL),
// DISP=(OLD,KEEP),VOLUME=SER=001234,
// DCB=(RECFM=F,LRECL=80,BLKSIZE=80)
//SYSUT2   DD   DSNAME=NEWSET,UNIT=2314,DISP=(,KEEP),
// VOLUME=SER=111112,SPACE=(TRK,(20,10,5)),
// 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 discussed below:

- SYSUT1 DD defines the input data set (INSET). The data set was originally written on a 9-track tape volume at 800 bits per inch density.
- SYSUT2 DD defines the output partitioned data set (NEWSET). The data set is to be placed on a 2314 volume. Twenty tracks of primary space, ten 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: (1) three member names are included in subsequent MEMBER statements and (2) 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 bytes 1 through 8 of the input record.
- The remaining MEMBER and RECORD statements define the second and third members.

IEBGENER Example 7

In this example, sequential input is to be converted into two partitioned members. The newly created members are to be merged into an existing partitioned data set. User labels on the input data set are to be passed to the user exit routines.

The example follows:

```
//DRUMDRUM JOB  09#660,SMITH
//         EXEC PGM=IEBGENER
//SYSPRINT DD   SYSOUT=A
//SYSUT1   DD   DSNAME=INSET,UNIT=2301,DISP=(OLD,KEEP),
// VOLUME=SER=111112,DCB=(RECFM=FB,LRECL=80,
// BLKSIZE=20000),LABEL=(,SUL)
//SYSUT2   DD   DSNAME=EXISTSET,UNIT=2301,DISP=(MOD,KEEP),
// VOLUME=SER=111113,DCB=(RECFM=FB,LRECL=80,
// BLKSIZE=2000)
//SYSIN    DD   *
     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 discussed below:

- SYSUT1 DD defines the input data set (INSET). The input data set, which resides on a 2301 volume, has standard and 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 output records are blocked to reduce the space required by the new members.
- 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: (1) two member names and one alias are included in subsequent MEMBER statements and (2) an 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 first RECORD statement identifies the last record to be placed in the first member. The name of this record (FIRSTMEM) appears in bytes 1 through 8 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.

IEBGENER Example 8

In this example, a sequential input data set is to be edited and copied.

The example follows:

```
//TAPETAPE JOB  09#660,SMITH
//         EXEC PGM=IEBGENER
//SYSPRINT DD   SYSOUT=A
//SYSUT1   DD   DSNAME=OLDSET,UNIT=2400-2,DISP=(OLD,KEEP),
// VOLUME=SER=001234,LABEL=(3,SUL),
// DCB=(RECFM=F,LRECL=80,BLKSIZE=80,TRTCH=C),
//SYSUT2   DD   DSNAME=NEWSET,UNIT=2400-2,DISP=(NEW,PASS),
// DCB=(RECFM=FB,LRECL=80,BLKSIZE=2000,TRTCH=C),
// VOLUME=SER=001235,LABEL=(,SL)
//SYSIN    DD   *
     GENERATE   MAXFLDS=3,MAXLITS=11
       RECORD   FIELD=(10,'**********',,1),                    ℛC
               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 discussed below:

- SYSUT1 DD defines the sequential input data set (OLDSET). The data set was originally written as the third data set (800 bits per inch) on a 7-track tape volume.
- SYSUT2 DD defines the sequential output data set (NEWSET). The data set is written as the first or only data set on a 7-track tape volume. A density of 800 bits per inch and data conversion are specified for the write operation. 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 data set is passed to a subsequent job step.
- SYSIN DD defines the control data set, which follows in the input stream.
- GENERATE indicates that: (1) a maximum of three FIELD parameters is included in subsequent RECORD statements and (2) a maximum of 11 literal characters are included in subsequent FIELD parameters.
- 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 first RECORD statement controls the editing, as follows: (1) asterisks are placed in positions 1 through 10, (2) bytes 1 through 5 of the input record are converted from H-set BCD to EBCDIC mode and moved to positions 11 through 15, and (3) an equal sign is placed in byte 16.

- 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.

Note: This example shows the relationship between the RECORD LABELS statement and the EXITS statement. IEBGENER attempts to write a first and second label trailer as user labels at close time of SYSUT2, but, before returning control to the system; the user routine ROUT2 can review these records and change them, if necessary.

IEBGENER Example 9

In this example, a sequential input data set is to be edited and copied.

The example follows:

```
//DISKDISK JOB  09#660,SMITH
//         EXEC PGM=IEBGENER
//SYSPRINT DD   SYSOUT=A
//SYSUT1   DD   DSNAME=OLDSET,UNIT=2314,DISP=(OLD,KEEP),
// VOLUME=SER=111112,DCB=(RECFM=F,LRECL=100,BLKSIZE=100)
//SYSUT2   DD   DSNAME=NEWSET,UNIT=2314,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
GROUP1 RECORD   IDENT=(8,'FIRSTGRP',1),                          ℛC
               FIELD=(21,80,,60),FIELD=(59,1,,1)
GROUP2 RECORD   FIELD=(11,90,,70),FIELD=(69,1,,1)
/*
```

The control statements are discussed below:

- SYSUT1 DD defines the input data set (OLDSET). The logical record length of the input records is 100 bytes.
- SYSUT2 DD defines the output data set (OUTSET). Twenty tracks of primary storage space and ten tracks of secondary storage space are allocated for the data set on a 2314 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: (1) a maximum of four **FIELD** parameters is included in subsequent RECORD statements and (2) a maximum of one **IDENT** parameter appears in a subsequent RECORD statement.
- EXITS identifies the user routine that handles input/output errors.
- The first RECORD statement controls the editing of the first record group, as follows: (1) FIRSTGRP, which appears in bytes 1 through 8 of the input record, is defined as being the last record in the first group of records and (2) bytes 80 through 100 of each input record are moved into positions 60 through 80 of each corresponding output record. (This example implies that bytes 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 remaining bytes within each input record are transferred directly to the output records, specified in the second **FIELD** parameter.
- The second RECORD statement indicates that the remainder of the input records are to be processed as the second record group. Bytes 90 through 100 of each input record are moved into positions 70 through 80 of the output records. (This example implies that bytes 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 remaining bytes within each input record are transferred directly to the output records, specified in the second **FIELD** parameter.

If the logical record length of the output data set differs from that of the input data set, as in this example, all positions in the output records must undergo editing to justify the new logical record length.

IEBGENER Example 10

In this example, a sequential input data set is to be edited and copied.

The example follows:

```
//TAPETAPE JOB  09#660,SMITH
//         EXEC PGM=IEBGENER
//SYSPRINT DD   SYSOUT=A
//SYSUT1   DD   DSNAME=OLDSET,UNIT=2400,DISP=(OLD,KEEP),
// VOLUME=SER=001234,LABEL=(3,SUL),DCB=(RECFM=F,
//   LRECL=80,BLKSIZE=80)
//SYSUT2   DD  DSNAME=NEWSET,UNIT=2400,DISP=(NEW,PASS),
// VOLUME=SER=001235,LABEL=(,SUL),
// DCB=(RECFM=FB,LRECL=80,BLKSIZE=2000)
//SYSIN    DD   *
     GENERATE   MAXFLDS=3,MAXLITS=11
       RECORD   FIELD=(10,'**********',,1),                    %C
               FIELD=(5,1,HE,11),FIELD=(1,'=',,16)
       LABELS   DATA=INPUT
       RECORD   LABELS=3
```

(first header label record)
(second header label record)
(third header label record)

```
       RECORD   LABELS=2
```

(first trailer label record)
(second trailer label record)

```
/*
```

The control statements are discussed below:

- SYSUT1 DD defines the input data set (OLDSET). The data set was originally written as the third data set (800 bits per inch) on a 9-track tape volume.
- SYSUT2 DD defines the output data set (NEWSET). The data set is written as the first or only data set on a 9-track tape volume. A density of 800 bits per inch is specified for the write operation. 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 data set is passed to a subsequent job step.
- SYSIN DD defines the control data set, which follows in the input stream.
- GENERATE indicates that: (1) a maximum of three **FIELD** parameters is included in subsequent RECORD statements and (2) a maximum of 11 literal characters are included in subsequent **FIELD** parameters.
- LABELS indicates that label records are included in the input stream.
- The first RECORD statement controls the editing, as follows: (1) asterisks are placed in positions 1 through 10, (2) bytes 1 through 5 of the input record are converted from H-set BCD to EBCDIC mode and moved to positions 11 through 15, and (3) an equal sign is placed in byte 16.
- The second and third RECORD statements indicate that three 80-byte records (cards), to be written as user labels on the output data set, immediately follow. The first RECORD statement indicates that the following cards are to be treated as header labels. The second RECORD statement indicates that the following cards are to be treated as trailer labels.

IEBISAM Program—Class C

IEBISAM is a data set utility used to copy an indexed sequential data set directly from one direct access volume to another. (See "Introduction" for general data set utility information.)

Alternatively, IEBISAM can be used to reorganize an indexed sequential data set into a sequential (*unloaded*) data set and place that data set on a direct access or magnetic tape volume. The *unloaded* data set is in a form that can be subsequently *loaded*, that is, it can be converted back into an indexed sequential data set.

Optionally, IEBISAM can be used to print the records of an indexed sequential data set.

IEBISAM can be used to:

- Copy an indexed sequential data set.
- Create a sequential backup (transportable) copy of source data from an indexed sequential data set.
- Create an indexed sequential data set from an unloaded data set.
- Print an indexed sequential data set.

At the completion or termination of IEBISAM, the highest return code encountered within the program is passed to the calling program.

Copying an Indexed Sequential Data Set

IEBISAM can be used to copy an indexed sequential data set directly from one direct access volume to another. When the data set is copied, the records marked for deletion are only deleted if the DELETE parameter was specified in the OPTCD (optional control program service) field. Those records that are contained in the overflow area of the original data set are moved into the primary area of the copied data set. The control information characteristics such as BLKSIZE and OPTCD can be overridden by new specifications. Caution should be used, however, when overriding these characteristics (see "Unloaded Data Sets" in this chapter).

Creating a Sequential Backup Copy

An *unloaded* sequential data set can be created to serve as a backup or transportable copy of source data from an indexed sequential data set. When the unloaded data set is created, the records marked for deletion are only deleted if the DELETE parameter was specified in the OPTCD field. When the data set is subsequently *loaded*—reconstructed into an indexed sequential data set—records that were contained in the overflow area assigned to the original data set are moved sequentially into the primary area.

An unloaded data set consists of 80-byte logical records. The data set contains:

- Fixed records from an indexed sequential data set.
- Control information used in the subsequent loading of the data set.

Control information consists of characteristics that were assigned to the indexed sequential data set. These characteristics are:

- Optional control program service (OPTCD)
- Record format (RECFM)
- Logical record length (LRECL)
- Block size (BLKSIZE)
- Relative key position (RKP)
- Number of tracks in cylinder index (NTM)
- Key length (KEYLEN)
- Number of overflow tracks on each cylinder (CYLOFL)

When a load operation is specified, these characteristics can be overridden by new specifications in the DCB parameter of the SYSUT2 DD statement (refer to "Job Control Statements" for a discussion of the SYSUT2 DD statement). Caution should be used, however, because checks are made to ensure that:

1. Record format is the same as that of the original indexed sequential data set (either fixed or variable length).
2. Logical record length is greater than or equal to that of the original indexed sequential data set when the RECFM is V or VB.

3. For fixed records, the block size is equal to or a multiple of the logical record length of the records in the original indexed sequential data set. For variable records, the block size is equal to or greater than the logical record length plus four.
4. Relative key position is equal to or less than the logical record length minus the key length. Following are relative key position considerations:
   - If the RECFM is variable (V) or variable blocked (VB), the relative key position should be at least 4.
   - If the DELETE parameter was specified in the OPTCD field and the RECFM is fixed or fixed blocked, the relative key position should be at least 1. If the DELETE parameter was specified in the OPTCD field and the RECFM is V or VB, the relative key position should be at least 5.
5. The key length is less than or equal to 255 bytes.
6. For a fixed unblocked data set with relative key position equal to zero, the LRECL is the length of the data. In all other cases, the LRECL is the length of the key plus the data. When changing the record format from fixed unblocked and RKP = 0 to fixed blocked, the output LRECL value must be equal to the input LRECL plus the input key length.

If either RKP or KEYLEN is overridden, it might not be possible to reconstruct the data set.

The number of 80-byte logical records in an unloaded data set can be determined by the formula:

$$x = \frac{n(y+2) + 158}{78}$$

where x is the number of 80-byte logical records created, n is the number of records in the indexed sequential data set, and y is the length of a fixed record or the average length of variable records.

Figure 32 shows the format of an unloaded data set for the first three 100-byte records of an indexed sequential data set. Each is preceded by two bytes (bb) that indicate the number of bytes in that record. (The last record is followed by two bytes containing binary zeros to identify the last logical record in the unloaded data set.) The characteristics of the indexed sequential data set are contained in the first two logical records of the unloaded data set. Data from the indexed sequential data set begins in the third logical record. Each logical record in the unloaded data set contains a binary sequence number (aa) in the first two bytes of the record.

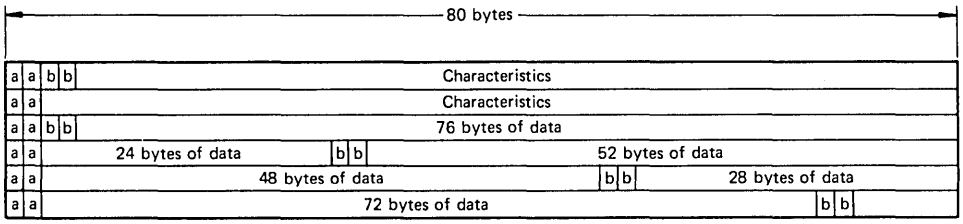


Figure 32. An Unloaded Data Set Created Using IEBISAM

Creating an Indexed Sequential Data Set from an Unloaded Data Set

An indexed sequential data set can be created from an unloaded version of an indexed sequential data set. When the unloaded data set is loaded, those records that were contained in the overflow area assigned to the original indexed sequential data set are moved sequentially into the primary area of the loaded indexed sequential data set.

Printing the Logical Records of an Indexed Sequential Data Set

The records of an indexed sequential data set can be printed or stored as a sequential data set for subsequent printing. Each input record is placed in a buffer from which it is printed or placed in a sequential data set. When the DELETE parameter is specified in the OPTCD field, each input record not marked for deletion is also placed in a buffer from which it is printed or placed in a sequential data set. Each printed record is converted to hexadecimal unless specified otherwise by the user.

IEBISAM provides user exits so that the user can include his own routines to:

- Modify records before printing.
- Select records for printing or terminate the printing operation after a certain number of records have been printed.
- Convert the format of a record to be printed.

- Provide a record heading for each record if the record length is at least 18 bytes. If no user routines are provided, each record is identified in sequential order on the printout.

When a user routine is supplied for a print operation, IEBISAM issues a LOAD macro instruction. A BALR 14,15 instruction is used to give control to the user's routine. When the user's routine receives control, register 0 contains a pointer to a record heading buffer; register 1 contains a pointer to an input record buffer.

The input record buffer has a length equal to the length of the input logical record.

Figure 33 shows the record heading buffer.

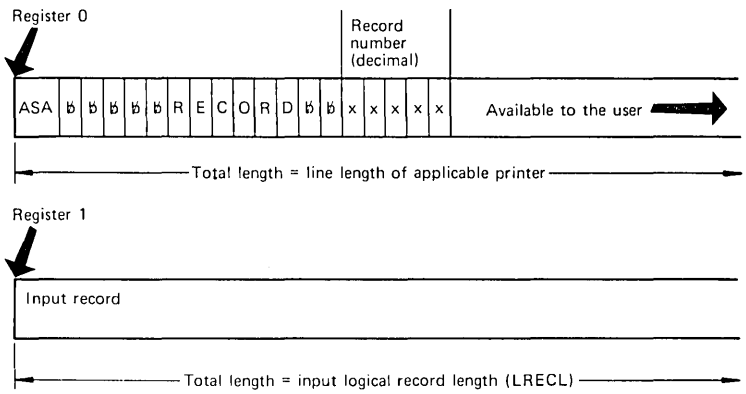


Figure 33. Record Heading Buffer Used by IEBISAM

The user returns control to IEBISAM by issuing a RETURN macro instruction (via register 14) or by using a BR 14 instruction after restoring registers 2 through 14. (Note that the user must save registers 2 through 14 when control is given to the user routine.)

A user routine must place a return code in register 15 before returning control to IEBISAM. The possible return codes and their meanings are:

- 00, which indicates that buffers are to be printed.
- 04, which indicates that the buffers are to be printed and the operation is to be terminated.
- 08, which indicates that this input record is not to be printed; processing continues.
- 12, which indicates that this input record is not to be printed; terminate the operation.

Input and Output

IEBISAM uses an input data set; the organization of the input data set depends on the operation to be performed, as follows:

- If a data set is to be copied, unloaded, or printed in logical sequence, the input is an indexed sequential data set.
- If a data set is to be loaded, the input is an unloaded sequential version of an indexed sequential data set.

IEBISAM produces as output an output data set, which is the result of the IEBISAM operation, and a message data set, which contains informational messages and any error messages.

IEBISAM provides a return code to indicate the results of program execution. The return codes and their meanings are:

- 00, which indicates successful completion.
- 04, which indicates that a return code of 04 or 12 was passed to IEBISAM by the user routine.
- 08, which indicates that an error condition occurred that caused termination of the operation.
- 12, which indicates that a return code other than 00, 04, 08, or 12 was passed to IEBISAM from a user routine. The job step is terminated.
- 16, which indicates that an error condition caused termination of the operation.

Control

IEBISAM is controlled by job control statements. Utility control statements are not used.

Job Control Statements

Table 21 shows the job control statements necessary for using IEBISAM.

Table 21. IEBISAM Job Control Statements

Statement	Use
JOB	Initiates the job.
EXEC	Specifies the program name (PGM = IEBISAM). Additional information is required on the EXEC statement to control the execution of IEBISAM; see "PARM Information on the EXEC Statement" below.
SYSUT1 DD	Defines the input data set.
SYSUT2 DD	Defines the output data set.
SYSPRINT DD	Defines a sequential message data set, which can be written to a system output device, a tape volume, or a direct access device.

The minimum region size that can be specified for the execution of IEBISAM is 8K.

If the block size of the SYSPRINT data set is not a multiple of 121, a default value of 121 is taken (no error message is issued, and no condition code is set).

PARM Information on the EXEC Statement

The PARM parameter on the EXEC statement is used to control the execution of IEBISAM. The PARM parameter is entered:

```
PARM = {COPY                                }
       {UNLOAD                              }
       {LOAD                                }
       {PRINTL                              }
       {'PRINTL[,N][,EXIT = routinename]' }
```

The PARM values have the following meaning:

- COPY specifies a copy operation.
- UNLOAD specifies an unload operation. This is the default.
- LOAD specifies a load operation.
- PRINTL specifies a print operation in which each record is converted to hexadecimal before printing. The N is an optional value that specifies that records are not to be converted to hexadecimal before printing.
- EXIT is an optional value that specifies the name of an exit routine that is to receive control before each record is printed.

Note: Exit routines must be included in either the job library or the link library.

For a COPY operation, the SYSUT2 DD statement must include a primary space allocation that is sufficient to accommodate records that were contained in overflow areas in the original indexed sequential data set. New overflow areas can be specified when the data set is copied.

For an UNLOAD operation, specifications that are implied by default or included in the DCB parameter of the SYSUT2 DD statement (for example, tape density) must be considered when the data set is subsequently loaded. If a block size is specified in the DCB parameter of the SYSUT2 DD statement, it must be a multiple of 80 bytes.

For a LOAD operation, if the input data set resides on an unlabeled tape, the SYSUT1 DD statement must specify a BLKSIZE that is a multiple of 80 bytes. Specifications that are implied by default or included in the DCB parameter of the SYSUT1 DD statement must be consistent with specifications that were implied or included in the DCB parameter of the SYSUT2 DD statement used for the UNLOAD operation. The SYSUT2 DD statement must include a primary space allocation that is sufficient to accommodate records that were contained in overflow areas in the original indexed sequential data set. If new overflow areas are desired, they must be specified when the data set is loaded.

For a PRINTL operation, if the device defined by the SYSUT2 DD statement is a printer, the specified BLKSIZE must be equal to or less than the physical printer size; that is 121, 133, or 145 bytes. If BLKSIZE is not specified, 121 bytes is assumed. LRECL (or BLKSIZE when no LRECL was specified) must be between 55 and 255 bytes.

If a user routine is supplied for a PRINTL operation, IEBISAM issues a LOAD macro instruction to make the user routine available. A BALR 14,15 instruction is subsequently used to give control to the routine. When the user routine receives control, register 0 contains a pointer to a *record heading buffer*; register 1 contains a pointer to an *input record buffer*.

IEBISAM Examples

The following examples illustrate some of the uses of IEBISAM. Table 22 can be used as a quick reference guide to IEBISAM examples. The numbers in the "Example" column point to the examples that follow.

Table 22. IEBISAM Example Directory

Operation	Data Set Organization	Devices	Comments	Example
COPY	Indexed sequential	2314 Disks	Unblocked input; blocked output Prime area and index separation.	1
UNLOAD	Indexed sequential, Sequential	2314 Disk, 9-track tape	Blocked output.	2
UNLOAD	Indexed sequential, Sequential	2314 Disk, 7-track tape	Blocked output. Data set written as second data set on input volume.	3
LOAD	Sequential, Indexed sequential	9-track tape, 2314 Disk	Input data set is second data set on tape volume.	4
PRINTL	Indexed sequential, Sequential	2311 Disk, System Printer	Blocked input. Output not converted.	5

IEBISAM Example 1

In this example, an indexed sequential data set is to be copied from two 2314 volumes to two other 2314 volumes. The output data is blocked.

The example follows:

```
//CPY       JOB  09#770,SMITH
//          EXEC PGM=IEBISAM,PARM=COPY
//SYSPRINT  DD   SYSOUT=A
//SYSUT1    DD   DSNAME=ISAM01,VOLUME=SER=(222222,333333),
// DISP=(OLD,DELETE),UNIT=(2314,2),
// DCB=(DSORG=IS,RECFM=F,LRECL=500,RKP=4,BLKSIZE=500)
//SYSUT2    DD   DSNAME=ISAM02(INDEX),UNIT=2314,
// DISP=(NEW,KEEP),VOLUME=SER=444444,
// DCB=(DSORG=IS,RECFM=FB,BLKSIZE=1000),SPACE=(CYL,(2))
//     DD   DSNAME=ISAM02(PRIME),UNIT=(2314,2),
// DSB=(DSORG=IS,BLKSIZE=1000),SPACE=(CYL,(10)),
// VOLUME=SER=(444444,555555),DISP=(NEW,KEEP)
/*
```

The control statements are discussed below:

- EXEC specifies the program name and the COPY operation.
- SYSUT1 DD defines an indexed sequential input data set, which resides on two 2314 volumes.
- SYSUT2 DD defines the output data set index area; the index and prime areas are separated.
- The second SYSUT2 DD defines the output data set prime area. Ten cylinders are allocated for the prime area on each of the two 2314 volumes.

IEBISAM Example 2

In this example, indexed sequential input is to be converted into a sequential data set; the output is to be placed on a 9-track tape volume.

The example follows:

```
//STEP1     JOB  09#770,SMITH
//          EXEC PGM=IEBISAM,PARM=UNLOAD
//SYSPRINT  DD   SYSOUT=A
//SYSUT1    DD   DSNAME=INDSEQ,UNIT=2314,DISP=(OLD,KEEP),
// VOLUME=SER=111112,DCB=(DSORG=IS)
//SYSUT2    DD   DSNAME=UNLDSET,UNIT=2400,LABEL=(,SL),
// DISP=(,KEEP),VOLUME=SER=001234,
// DCB=(RECFM=FB,LRECL=80,BLKSIZE=640)
/*
```

The control statements are discussed below:

- EXEC specifies the program name and the UNLOAD operation.
- SYSUT1 DD defines the indexed sequential input data set, which resides on a 2314 volume.

- SYSUT2 DD defines the unloaded output data set. The data set consists of fixed blocked records, and is to reside as the first or only data set on a 9-track tape volume. The data set is to be written at a density of 800 bits per inch.

IEBISAM Example 3

In this example, indexed sequential input is to be converted into a sequential data set and placed on a 7-track, tape volume.

The example follows:

```
//STEPA    JOB  09#770,SMITH
//         EXEC PGM=IEBISAM,PARM=UNLOAD
//SYSPRINT DD   SYSOUT=A
//SYSUT1   DD   DSNAME=INDSEQ,UNIT=2314,DISP=(OLD,KEEP),
// VOLUME=SER=111112,DCB=(DSORG=IS)
//SYSUT2   DD   DSNAME=UNLDSET,UNIT=2400-2,LABEL=(2,SL),
// VOLUME=SER=001234,DCB=(DEN=2,RECFM=FB,
// LRECL=80,BLKSIZE=1040,TRTCH=C),DISP=(,KEEP)
/*
```

The control statements are discussed below:

- EXEC specifies the program name and the UNLOAD operation.
- SYSUT1 DD defines the input data set, which is an indexed sequential data set. The data set resides on a 2314 volume.
- SYSUT2 DD defines the unloaded output data set. The data set consists of fixed blocked records, and is to reside as the second data set on a 7-track tape volume. The data set is to be written at 800 bits per inch density.

IEBISAM Example 4

In this example, an unloaded data set is to be converted to the form of the original indexed sequential data set.

The example follows:

```
//STEPA    JOB  09#770,SMITH
//         EXEC PGM=IEBISAM,PARM=LOAD
//SYSPRINT DD   SYSOUT=A
//SYSUT1   DD   DSNAME=UNLDSET,UNIT=2400,LABEL=(2,SL),
// DISP=(OLD,KEEP),VOLUME=SER=001234
//SYSUT2   DD   DSNAME=INDSEQ,DISP=(,KEEP),DCB=(DSORG=IS),
// SPACE=(CYL,(1)),VOLUME=SER=111112,UNIT=2314
/*
```

The control statements are discussed below:

- EXEC specifies the program name and the LOAD operation.
- SYSUT1 DD defines the input data set, which is a sequential (unloaded) data set. The data set is the second data set on a 9-track tape volume.
- SYSUT2 DD defines the output data set, which is an indexed sequential data set. One cylinder of space is allocated for the data set on a 2314 volume.

IEBISAM Example 5

In this example, the logical records of an indexed sequential data set are to be printed on a system output device.

The example follows:

```
//PRINT    JOB  09#770,SMITH
//         EXEC PGM=IEBISAM,PARM='PRINTL,N'
//SYSPRINT DD   SYSOUT=A
//SYSUT1   DD   DSNAME=ISAM03,UNIT=2311,DISP=OLD,
// VOLUME=SER=222222,DCB=(DSORG=IS)
//SYSUT2   DD   SYSOUT=A
/*
```

The control statements are discussed below:

- EXEC specifies the program name and the PRINTL operation. The output records are not to be converted to hexadecimal prior to printing.
- SYSUT1 DD defines the input data set, which resides on a 2311 volume.
- SYSUT2 DD defines the output data set. A logical record length (LRECL) of 121 bytes is assumed.

IEBPTPCH Program—Class C

IEBPTPCH is a data set utility used to print or punch all, or selected portions, of a sequential or partitioned data set. Records can be printed or punched to meet either standard specifications or user specifications. (See "Introduction" for general data set utility information.)

The standard specifications are:

- Each logical record begins on a new printed line or punched card.
- Each printed line consists of groups of 8 characters separated by 2 blanks. 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 record is delimited by "*" and each block is delimited by "**".

User formats can be specified, provided that no output record exceeds the capability of the output device.

IEBPTPCH provides optional editing facilities and exits for user routines that can be used to process labels or manipulate input or output records.

IEBPTPCH can be used to:

- Print or punch a sequential or partitioned data set in its entirety.
- Print or punch selected members from a partitioned data set.
- Print or punch selected records from a sequential or partitioned data set.
- Print or punch the directory of a partitioned data set.
- Print or punch an edited version of a sequential or partitioned data set.

At the completion or termination of the program, the highest return code encountered within the program is passed to the calling program.

Printing or Punching a Data Set

IEBPTPCH can be used to print or punch a sequential data set or a partitioned data set in its entirety. Data to be printed or punched can be either hexadecimal or a character representation of valid alphameric bit configurations. For a print operation, packed decimal data should be converted to unpacked decimal or hexadecimal mode to ensure that all characters are printable.

For a standard print operation, each logical record is printed in groups of eight characters. Each set of eight characters is separated from the next by two blanks. Up to 112 characters can be included on a printed line. (An edited output can be produced to omit the blank delimiters and print up to 144 characters per line.)

Data from an input logical record is punched in contiguous columns in the punched card(s) representing that record. Sequence numbers can be created and placed in columns 73 through 80 of the punched cards.

Printing or Punching Selected Members

IEBPTPCH can be used to print or punch selected members of a partitioned data set. Utility control statements are used to specify members to be printed or punched.

Printing or Punching Selected Records

IEBPTPCH can be used to print selected records from a sequential or partitioned data set. 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 *nth* record.
- The starting of a print or punch operation after a specified number of records.

Printing or Punching a Partitioned Directory

IEBPTPCH can be used to print or punch the contents of a partitioned directory. Each directory block is printed in groups of eight characters. If the directory is printed in hexadecimal representation, the first four printed characters of each directory block indicate the total number of used bytes in that block. For details of the format of the directory, see *OS System Control Blocks*, GC28-6628.

Data from a directory block is punched in contiguous columns in the punched cards representing that block.

Printing or Punching an Edited Data Set

IEBPTPCH can be used to print or punch an edited version of a sequential or a partitioned data set. 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.

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 alphameric to hexadecimal representation.

Input and Output

IEBPTPCH uses the following input:

- An input data set, which contains the data that is to be 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 IEBPTPCH.

IEBPTPCH produces the following output:

- 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.

IEBPTPCH provides a return code to indicate the results of program execution. The return codes and their meanings are:

- 00, which indicates successful completion.
- 04, which indicates either that a physical sequential data set is empty, or that a partitioned data set contains *no* members.
- 08, which indicates that a member specified for printing does not exist in the input data set. Processing continues with the next member.
- 12, which indicates that an unrecoverable error occurred or that a user routine passed a return code of 12 to IEBPTPCH. The job step is terminated.
- 16, which indicates that a user routine passed a return code of 16 to IEBPTPCH. The job step is terminated.

Control

IEBPTPCH is controlled by job control statements and utility control statements. The job control statements are required to execute or invoke the IEBPTPCH 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 IEBPTPCH.

Job Control Statements

Table 23 shows the job control statements necessary for using IEBPTPCH.

Table 23. IEBPTPCH Job Control Statements

Statement	Use
JOB	Initiates the job step.
EXEC	Specifies the program name (PGM = IEBPTPCH) or, if the job control statements reside in a procedure library, the procedure name.
SYSPRINT DD	Defines a sequential message data set. The data set can be written to a system output device, a tape volume, or a direct access device.
SYSUT1 DD	Defines a sequential or partitioned input data set.
SYSUT2 DD	Defines the output (printed or punched) 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.

The minimum region size that can be specified for the execution of IEBPTPCH is 16K + 2b, where b is the largest block size in the job step rounded to the next higher 2K.

The input data set can contain fixed, variable, undefined, or variable spanned records. Variable spanned records are allowed only when the input is sequential.

Both the output data set and the message data set can be written to the system output device if it is a printer. Variable spanned records are allowed only when the input is sequential.

If the logical record length of the input records is such that the output would exceed the output record length, the utility divides the record into multiple lines or cards in the case of standard printed output, standard punched output, or when the **PREFORM** parameter was specified. Otherwise, only part of the input record is printed (a maximum of 144 characters) or punched (a maximum of 80 characters).

Restrictions

- The SYSPRINT DD statement is required for each use of IEBPTPCH.
- The RECFM must be fixed block with ASCII carriage control characters (FBA), and the LRECL must be 121. Output can be blocked by specifying a BLKSIZE which is a multiple of 121 on the SYSPRINT DD statement. The default BLKSIZE is 121.
- The SYSUT1 DD statement is required for each use of IEBPTPCH. The RECFM (except for undefined records), and the BLKSIZE and the LRECL (except for undefined and fixed unblocked records) must be present on the DD statement, in the DSCB, or on the tape label.
- The SYSUT2 DD statement is required for each use of IEBPTPCH. The RECFM must be FBA or fixed block with machine-code control characters (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 includes a 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 for the SYSPRINT data set must be a multiple of 121. The block size for the SYSIN data set must be a multiple of 80. Any blocking factor can be specified for these block sizes.
- The SYSIN DD statement is required for each use of IEBPTPCH.
- The RECFM must be FB and the LRECL must be 80. Any blocking factor can be specified for the BLKSIZE (multiple of 80). The default BLKSIZE is 80.
- A partitioned directory to be printed/punched must be defined as a sequential data set (TYPORG = PS). Code the RECFM, BLKSIZE, and LRECL in the SYSUT1 DD card.

Utility Control Statements

IEBPTPCH is controlled by utility control statements. The control statements are shown in the order in which they must appear, as follows:

- PRINT or PUNCH statement, which specifies that the data is to be either printed or punched.
- TITLE statement, which specifies that a title is to precede the printed or punched data.
- EXITS statement, which specifies that user routines are provided.
- MEMBER statement, which specifies that the input is a partitioned data set and that a selected member is to be printed or punched.
- RECORD statement, which specifies whether editing is to be performed, that is, records are to be printed or punched to nonstandard specifications.
- LABELS statement, which specifies whether user labels are to be treated as data.

The 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.

The PRINT statement is used to initiate the IEBPTPCH operation. If this is a print operation, PRINT must be the first statement in the control data set.

The format of the PRINT statement is:

```
[label] PRINT [PREFORM = {A }
                         {M }]
              [,TYPORG = {PS }
                         {PO }]
              [,TOTCONV = {XE }
                          {PZ }]
              [,CNTRL = n]
              [,STRTAFT = n]
              [,STOPAFT = n]
              [,SKIP = n]
              [,MAXNAME = n]
              [,MAXFLDS = n]
              [,MAXGPS = n]
              [,MAXLITS = n]
              [,INITPG = n]
              [,MAXLINE = n]
```

where:

PREFORM =
specifies that a control character is provided as the first character of each record to be printed. The control characters are used to control the spacing, number of lines per page, and page ejection. If an error occurs, the print operation is terminated. If **PREFORM** is coded, except for syntax checking, any additional PRINT operands and all other control statements except LABELS statements are ignored. **PREFORM** must not be used for printing data sets with VS or VBS records longer than 32K bytes. These values can be coded:

A
specifies that an ASA control character is provided as the first character of each record to be printed. 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.

M
specifies that a machine-code control character is provided as the first character of each record to be printed. 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.

TYPORG =
specifies the organization of the input data set. If **TYPORG** is omitted, sequential organization is assumed. These values can be coded:

PS
specifies that the input data set is organized sequentially.

PO
specifies that the input data set is partitioned.

TOTCONV =
specifies the representation of data to be printed. **TOTCONV** can be overridden by any user specifications (RECORD statements) that pertain to the same data. These values can be coded:

XE
specifies that data is to be printed in 2-character per byte hexadecimal representation (for example, C3 40 F4 F6). If **XE** is not specified, data is printed in 1-character per byte alphameric representation. The above example would appear as C 46.

PZ
specifies that data (packed decimal mode) is to be converted to unpacked decimal mode. If **TOTCONV** is omitted, data is not converted. IEBPTPCH does not check for packed decimal mode. The output is unpredictable when the input is not packed decimal.

CNTRL = *n*
specifies a control character for the output device that indicates line spacing, as follows: 1 indicates single spacing; 2 indicates double spacing; and 3 indicates triple spacing. If **CNTRL** is omitted, 1 is assumed.

STRTAFT = *n*
specifies, for sequential data sets, the number of logical records (physical blocks in the case of VS or VBS type records longer than 32K bytes) to be skipped before printing begins. For partitioned data sets, specifies the number of logical records to be skipped in each member before printing begins. The *n* value must not exceed 32,767. 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.

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. For partitioned data sets, this specifies the number of logical records to be printed in each member to be processed. The *n* value must not exceed 32,767. If **STOPAFT** is specified and RECORD statements are present, the operation is terminated when the **STOPAFT** count is satisfied, at the end of a record group, or at the end of the data set; whichever occurs first.

SKIP = *n*
specifies that every *nth* record (or physical block in the case of VS or VBS records longer than 32K bytes) is to be printed. If **SKIP** is omitted, successive logical records are printed.

MAXNAME = *n*
specifies a number no less than the total number of subsequent MEMBER statements. If **MAXNAME** is omitted when there is a MEMBER statement present, the print request is terminated.

MAXFLDS = *n*
specifies a number no less than the total number of **FIELD** parameters appearing in subsequent RECORD statements. If **MAXFLDS** is omitted when there is a **FIELD** parameter present, the print request is terminated.

MAXGPS = *n*
specifies a number no less than the total number of **IDENT** parameters appearing in subsequent RECORD statements. If **MAXGPS** is omitted when there is an **IDENT** parameter present, the print request is terminated.

MAXLITS = *n*
specifies a number no less than the total number of characters contained in the **IDENT** literals of subsequent RECORD statements. If **MAXLITS** is omitted when there is a literal present, the print request is terminated.

INITPG = *n*
specifies the initial page number; the pages are numbered sequentially thereafter. If **INITPG** is omitted, 1 is assumed. The value of *n* must not be greater than 9999.

MAXLINE = *n*
specifies the maximum number of lines to a printed page. Spaces, titles, and subtitles are included in this number. If *n* is smaller than the total number of spaces, titles, and subtitles, an error message is issued. If **MAXLINE** is omitted, 60 is assumed.

The PUNCH statement is used to initiate the IEBPTPCH operation. If this is a punch operation, PUNCH must be the first statement in the control data set.

The format of the PUNCH statement is:

```
[label] PUNCH [ PREFORM = {A }
                          {M }]
              [,TYPORG = {PS }
                         {PO}]
              [,TOTCONV = {XE }
                          {PZ }]
              [,CNTRL = n]
              [,STRTAFT = n]
              [,STOPAFT = n]
              [,SKIP = n]
              [,MAXNAME = n]
              [,MAXFLDS = n]
              [,MAXGPS = n]
              [,MAXLITS = n]
              [,CDSEQ = n]
              [,CDINCR = n]
```

where:

PREFORM =
specifies that a control character is provided as the first character of each record to be punched. The control characters are used to select a stacker. If an error is discovered, the punch operation is terminated. If **PREFORM** is coded, except for syntax checking, any additional PUNCH operands and all other control statements except LABELS statements are ignored. **PREFORM** must not be used for punching data sets with VS or VBS records longer than 32K bytes. These values can be coded:

A
specifies that an ASA control character is provided as the first character of each record to be punched. If the input record length exceeds the output record length, the utility duplicates the ASA character on each output card of the record.

M
specifies that a machine-code control character is provided as the first character of each record to be punched. If the input record length exceeds the output record length, the utility duplicates the machine control character on each output card of the record.

TYPORG =
specifies the organization of the input data set. If **TYPORG** is omitted, sequential organization is assumed. These values can be coded:

PS
specifies that the input data set is organized sequentially. This is the default.

PO
specifies that the input data set is partitioned.

TOTCONV =
specifies the representation of data to be punched. **TOTCONV** can be overridden by any user specifications (RECORD statements) that pertain to the same data. These values can be coded:

XE
specifies that data is to be 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 alphameric representation. The above example would appear as C 46.

PZ
specifies that data (packed decimal mode) is to be converted to unpacked decimal mode. If **TOTCONV** is omitted, data is not converted. IEBPTPCH does not check for packed decimal mode. The output is unpredictable when the input is not packed decimal.

CNTRL = *n*
specifies a control character for the output device that is used to select the stacker, as follows: 1 indicates the first stacker and 2 indicates the second stacker. If **CNTRL** is omitted, 1 is assumed.

STRTAFT = *n*
specifies, for sequential data sets, the number of logical records (physical blocks in the case of VS or VBS type records longer than 32K bytes) to be skipped before punching begins. For partitioned data sets, specifies the number of logical records (physical blocks in the case of VS or VBS type records longer than 32K bytes) to be skipped in each member before punching begins. The *n* value must not exceed 32,767. If **STRTAFT** is specified and RECORD statements are present, the first RECORD statements of a member describes the format of the first logical record to be 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 punched. For partitioned data sets, this specifies the number of logical records (or physical blocks in the case of VS or VBS records longer than 32K bytes) to be punched in each member to be processed. The *n* value must not exceed 32,767. If **STOPAFT** is specified and RECORD statements are present, the operation is terminated when the **STOPAFT** count is satisfied or at the end of the first record group, whichever occurs first.

SKIP = *n*
specifies that every *nth* record (or physical block in the case of VS or VBS records longer than 32K bytes) is to be punched. If **SKIP** is omitted, successive logical records are punched.

MAXNAME = *n*
specifies a number no less than the total number of subsequent MEMBER statements. If **MAXNAME** is omitted when there is a MEMBER statement present, the punch request is terminated.

MAXFLDS = *n*
specifies a number no less than the total number of **FIELD** parameters appearing in subsequent RECORD statements. If **MAXFLDS** is omitted when there is a **FIELD** parameter present, the punch request is terminated.

MAXGPS = *n*
specifies a number no less than the total number of **IDENT** parameters appearing in subsequent RECORD statements. If **MAXGPS** is omitted when there is an **IDENT** parameter present, the punch request is terminated.

MAXLITS = *n*
specifies a number no less than the total number of characters contained in the **IDENT** literals of subsequent RECORD statements. If **MAXLITS** is omitted when there is a literal present, the punch request is terminated.

CDSEQ = *n*
specifies the initial 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. If **CDSEQ** is omitted, the cards are not numbered. If the value of *n* is zero, 00000000 is assumed as a starting sequence.

CDINCR = *n*
specifies the increment to be used in generating sequence numbers. If **CDINCR** is omitted and **CDSEQ** is coded, 10 is assumed as an increment value for sequence numbering.

TITLE Statement

The TITLE statement is used to request title and subtitle records. Two TITLE statements can be included for each use of IEBPTPCH. A first TITLE statement defines the title, and a second defines the subtitle. The TITLE statement, if included, must immediately follow the PRINT or PUNCH statement in the control data set.

The format of the TITLE statement is:

```
[name] TITLE ITEM = ('title'[,output-location])[,ITEM...]
```

where:

ITEM =
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.

output-location
specifies the starting position at which the literal for this item is to be placed in the output record. If *output-location* is not specified, 1 is assumed. The specified title may not exceed the output logical record length minus 1.

EXITS Statement

The EXITS statement is used to identify exit routines supplied by the user. Exits to label processing routines are ignored if the input data set is partitioned. Linkage to and from user routines are discussed in "Appendix A: Exit Routine Linkage."

The EXITS statement, if included, must immediately follow any TITLE statement or follow the PRINT or PUNCH statement.

The format of the EXITS statement is:

```
[label] EXITS  [INHDR = routinename]
               [INTLR = routinename]
               [INREC = routinename]
               [OUTREC = routinename]
```

where:

INHDR = *routinename*
specifies the symbolic name of a routine that processes user input header labels.

INTLR = *routinename*
specifies the symbolic name of a routine that processes user input trailer labels.

INREC = *routinename*
specifies the symbolic name of a 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 symbolic name of a 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. When standard specifications are used, this exit is not available.

MEMBER Statement

The MEMBER statement is used to identify members to be 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 to standard specifications.

If no MEMBER statement appears, and a partitioned data set 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.

The format of the MEMBER statement is:

```
[label] MEMBER NAME = {membername }
                      {aliasname  }
```

where:

NAME =
specifies a member to be printed or punched. These values can be coded:

membername
specifies a member by its member name.

aliasname
specifies a member by its alias.

If the **NAME** parameter is specified in the MEMBER statement, **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 to be printed or punched to the user's 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 to standard specifications. 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.

The format of the RECORD statement is:

```
[label] RECORD  [IDENT = (length,'name',input-location)]
                [,FIELD = (length[,input-location][,conversion]
                [,output-location])][,FIELD = ...]
```

where:

IDENT =
identifies the last record of the record group to which the **FIELD** parameters apply. If **IDENT** is omitted and **STOPAFT** is not included with the PRINT or PUNCH statement, record processing halts 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. 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 eight bytes.

'name'
specifies the exact literal that identifies the last record of a record group. If the literal contains apostrophes, each must be written as two consecutive apostrophes.

input-location
specifies the starting location of the field that contains the identifying name in the input records.

Note: The sum of the length and input location must be equal to or less than the initial LRECL plus one.

FIELD =
specifies field processing and editing information. These values can be coded:

length
specifies the length (in bytes) of the input field to be processed.

Note: The length must be equal to or less than the initial input LRECL.

input-location
specifies the starting byte of the input field to be processed. If *input-location* is not specified, 1 is assumed.

Note: The sum of the length and input location must be equal to or less than the initial input LRECL plus one.

conversion
specifies a two-byte code that indicates the type of conversion to be performed on this field before it is printed or punched. If *conversion* is not specified, the field is moved to the output area without change. The values that can be coded are:

PZ
specifies that data (packed decimal) is to be converted to unpacked decimal data. The converted part of the input record (length L) occupies 2L - 1 output characters.

XE
specifies that data (alphameric) is to be converted to hexadecimal data. The converted part of the input record (length L) occupies 2L output characters.

output-location
specifies the starting location of this field in the output records. If *output-location* is not specified, 1 is assumed. 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. The specified fields may not exceed the output logical record length minus 1. When either one or multiple "FIELDS" are specified, the sum of all lengths and extra characters needed for conversions must be equal to or less than the output LRECL minus one.

A RECORD statement referring to a partitioned data set 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/PUNCH statement.

If an **IDENT** parameter is included in the RECORD statement, **MAXGPS** must be specified in the PRINT/PUNCH statement. If a literal is specified in the IDENT parameter, **MAXLITS** must be specified in the PRINT/PUNCH statement.

LABELS Statement

The LABELS statement specifies whether user labels are to be treated as data. For a detailed discussion of this option, refer to "Processing User Labels as Data," in "Appendix E: Processing User Labels."

The format of the LABELS statement is:

```
[name] LABELS [DATA = {YES  }
                      {NO   }
                      {ALL  }
                      {ONLY }]
```

where:

DATA =
specifies whether 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. If no value is entered, **YES** is assumed.

NO
specifies that user labels are not to be treated as data.

ALL
specifies that user labels are to be treated as data regardless of any return code. A return code of 16 causes the utility to complete the processing of the remainder of the group of user labels and to terminate 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 terminates upon return from the OPEN routine.

Note: DATA = NO must be specified to make standard user labels (SUL) exits inactive when input data sets with nonstandard labels (NSL) are to be processed.

If more than one valid LABELS statement is included, all but the last LABELS statement are ignored.

IEBPTPCH Examples

The following examples illustrate some of the uses of IEBPTPCH. Table 24 can be used as a quick reference guide to IEBPTPCH examples. The numbers in the "Example" column point to the examples that follow.

Table 24. IEBPTPCH Example Directory

Operation	Data Set Organization	Devices	Comments	Example
PRINT	Sequential	9-track tape, System printer	Standard format. Conversion to hexadecimal.	1
PUNCH	Sequential	7-track tape, Card Reader	Standard format. Conversion to hexadecimal.	2
PRINT	Partitioned	3330 Disk Storage System printer	Standard format. Conversion to hexadecimal. Ten records from each member are to be printed.	3
PRINT	Partitioned	2314 Disk, System printer	Standard format. Conversion to hexadecimal. Two members are to be printed.	4
PRINT	Sequential	9 track tape, System printer	User specified format. Input data set is the second data set on the volume.	5
PUNCH	Sequential	2314 Disk, Card Reader Punch	User specified format. Sequence numbers are to be assigned and punched.	6
PRINT	Sequential, Partitioned	2314 Disk, System printer	Standard format. Conversion to hexadecimal.	7
PUNCH	Sequential	Card Reader, Card Read Punch	Standard format. Control data set is a member in a cataloged partitioned data set.	8
PRINT	Sequential	2311 Disk, System printer	User specified format. User routines are provided. Processing ends after first record group is printed.	9

IEBPTPCH Example 1

In this example, a sequential data set is to be printed according to standard specifications. The input data set resides on a 9-track tape volume, originally written at 800 bits per inch density. The printed output is to be converted to hexadecimal.

The example follows:

```
//PRINT    JOB  09#660,SMITH
//         EXEC PGM=IEBPTPCH
//SYSPRINT DD   SYSOUT=A
//SYSUT1   DD   UNIT=2400,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 discussed below.

- SYSUT1 DD defines the input data set. 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). Each printed line contains groups (8 characters each) of hexadecimal information. Each record begins a new line of printed output.
- SYSIN DD defines the control data set, which follows in the input stream. The control data set contains the PRINT and TITLE statements.
- PRINT initiates the print operation and specifies conversion from alphameric 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.

IEBPTPCH Example 2

In this example, a sequential data set is to be punched according to standard specifications. The input data set resides on a 7-track tape volume, originally written at a density of 556 bits per inch. The punched output is converted to hexadecimal.

The example follows:

```
//PUNCHSET JOB  09#660,SMITH
//         EXEC PGM=IEBPTPCH
//SYSPRINT DD   SYSOUT=A
//SYSUT1   DD   DSNAME=INSET,UNIT=2400-2,VOLUME=SER=001234,
// LABEL=(,NL),DISP=(OLD,KEEP),DCB=(DEN=1,RECFM=FB,
//   LRECL=80,BLCKSIZE=2000,TRTCH=C)
//SYSUT2   DD   UNIT=2540-2
//SYSIN    DD   *
      PUNCH   TOTCONV=XE
      TITLE   ITEM=('PUNCH SEQ DATA SET WITH CONV TO HEX',10)
/*
```

The control statements are discussed below:

- SYSUT1 DD defines the input data set. The data set contains 80-byte, fixed blocked records.
- SYSUT2 DD defines the output data set. The data set is to be punched by an IBM 2540-2 Card Read Punch (punch feed). Each record from the input data set is represented by two punched cards.
- SYSIN DD defines the control data set, which follows in the input stream. The control data set contains the PUNCH and TITLE statements.
- PUNCH initiates the punch operation and specifies conversion from alphameric to hexadecimal representation.
- TITLE specifies a title to be placed beginning in column 10. The title is not converted to hexadecimal.

IEBPTPCH Example 3

In this example, a partitioned data set (ten records from each member) is to be printed according to standard specifications. The input data set resides on a 3330 volume. The printed output is converted to hexadecimal.

The example follows:

```
//PRINTPDS JOB  09#660,SMITH
//         EXEC PGM=IEBPTPCH
//SYSPRINT DD   SYSOUT=A
//SYSUT1   DD   DSNAME=PDS,UNIT=3330,DISP=(OLD,KEEP),
// VOLUME=SER=111112,DCB=(RECFM=U,BLKSIZE=3265)
//SYSUT2   DD   SYSOUT=A
//SYSIN    DD   *
        PRINT   TOTCONV=XE,TYPORG=PO,STOPAFT=10
        TITLE   ITEM=('PRINT PDS - 10 RECS EACH MEM',20)
/*
```

The control statements are discussed below:

- SYSUT1 DD defines the input data set. The data set contains undefined records; no record is larger than 3,625 bytes.
- SYSUT2 DD defines the output data set on the system output device (printer assumed). Each printed line contains groups (8 characters each) of hexadecimal information. Each record begins a new line of printed output. The size of the record determines how many lines of printed output are required per record.
- SYSIN DD defines the control data set, which follows in the input stream. The control data set contains the PRINT and TITLE statements.
- PRINT initiates the print operation, specifies conversion from alphameric to hexadecimal representation, indicates that the input data set is partitioned, and specifies that ten 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.

IEBPTPCH Example 4

In this example, two partitioned members are to be printed according to standard specifications. The input data set resides on a 2314 volume. The printed output is to be converted to hexadecimal.

The example follows:

```
//PRNTMEMS JOB  09#660,SMITH
//         EXEC PGM=IEBPTPCH
//SYSPRINT DD   SYSOUT=A
//SYSUT1   DD   DSNAME=PDS,DISP=(OLD,KEEP),VOLUME=SER=111112,
// DCB=(RECFM=F,LRECL=80,BLKSIZE=80),UNIT=2314
//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 discussed below:

- SYSUT1 DD defines the input data set. The data set contains 80-byte, fixed records.
- SYSUT2 DD defines the output data set on the system output device (printer assumed). Each printed line contains groups (8 characters each) of hexadecimal information. Each record begins a new line of printed output.
- SYSIN DD defines the control data set, which follows in the input stream. The control data set contains PRINT, TITLE, and MEMBER statements.
- PRINT initiates the print operation, indicates that the input data set is partitioned, specifies conversion from alphameric to hexadecimal representation, and indicates that two MEMBER statements appear in the control data set.
- 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.

IEBPTPCH Example 5

In this example, a sequential data set is to be printed according to user specifications. The input data set is the second data set on a 9-track tape volume. The data set was originally written at a density of 800 bits per inch.

The example follows:

```
//PTNONSTD JOB  09#660,SMITH
//         EXEC PGM=IEBPTPCH
//SYSPRINT DD   SYSOUT=A
//SYSUT1   DD   DSNAME=SEQSET,UNIT=2400,LABEL=(2,SUL),
// DISP=(OLD,KEEP),VOLUME=SER=001234,
// DCB=(RECFM=FB,LRECL=80,BLKSIZE=2000)
//SYSUT2   DD   SYSOUT=A
//SYSIN    DD   *
        PRINT   MAXFLDS=1
        EXITS   INHDR=HDRIN,INTLR=TRLIN
       RECORD   FIELD=(80)
       LABELS   DATA=YES
/*
```

The control statements are discussed below:

- SYSUT1 DD defines the input data set. The data set contains 80 byte, fixed blocked records.
- 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. The control data set contains the PRINT, RECORD, EXITS and LABELS statements.
- PRINT initiates the print operation and indicates that one **FIELD** parameter is included in a subsequent RECORD statement.
- 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 to be 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 to be printed according to the return code issued by the user exits.

IEBPTPCH Example 6

In this example, a sequential data set is to be punched according to user specifications. The input data set resides on a 2314 volume.

The example follows:

```
//PHSEQNO  JOB  09#660,SMITH
//         EXEC PGM=IEBPTPCH
//SYSPRINT DD   SYSOUT=A
//SYSUT1   DD   DSNAME=SEQSET,UNIT=2314,LABEL=(,SUL),
// VOLUME=SER=111112,DISP=(OLD,KEEP),
// DCB=(RECFM=FB,LRECL=80,BLKSIZE=2000)
//SYSUT2   DD   DSNAME=PUNCHSET,UNIT=2540-2
//SYSIN    DD   *
        PUNCH   MAXFLDS=1,CDSEQ=00000000,CDINCR=20
       RECORD   FIELD=(72)
       LABELS   DATA=YES
/*
```

The control statements are discussed below:

- SYSUT1 DD defines the input data set. The data set contains 80-byte, fixed blocked records.
- SYSUT2 DD defines the output data set. The data set is to be punched by an IBM 2540-2 Card Read Punch (punch feed). Each record from the input data set is represented by one punched card.
- SYSIN DD defines the control data set, which follows in the input stream. The control data set contains the PUNCH, RECORD, and LABELS statements.
- PUNCH initiates the punch operation, indicates that one FIELD parameter is included in a subsequent RECORD statement, 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 bytes 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 are to be punched. Labels cannot be edited. They are moved to the first 80 bytes of the output buffer. In this example, there are no sequence numbers present in the cards with user header and user trailer labels.

IEBPTPCH Example 7

In this example, the directory of a partitioned data set is to be printed. The input data set resides on a 2314 volume. The printed output is to be converted to hexadecimal.

The example follows:

```
//PRINTDIR JOB  09#660,SMITH
//         EXEC PGM=IEBPTPCH
//SYSPRINT DD   SYSOUT=A
//SYSUT1   DD   DSNAME=PDS,UNIT=2314,VOLUME=SER=111112,
// 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 discussed below:

- SYSUT1 DD defines the input data set (the partitioned directory).
- SYSUT2 DD defines the output data set on the system output device (printer assumed). Each printed line contains groups (8 characters each) of hexadecimal information. Six lines of print are required for each record. Each record begins a new line of printed output.
- SYSIN DD defines the control data set, which follows in the input stream. The control data set contains the PRINT, TITLE, and LABELS statements.
- PRINT initiates the print operation, indicates that the partitioned directory is organized sequentially, and specifies conversion from alphameric to hexadecimal representation.
- The first TITLE statement specifies a title, which is not converted to hexadecimal.

- The second TITLE statement specifies a subtitle, which is not converted to hexadecimal.
- LABELS specifies that no user labels are to be printed.

Note: Not all of the bytes in a directory block need contain data pertaining to the partitioned data set; unused bytes are sometimes used by the operating system as temporary work areas. 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.

IEBPTPCH Example 8

In this example, a card deck containing valid punch card code or BCD is to be duplicated. The input card deck resides in the input stream.

The example follows:

```
//PUNCH    JOB  09#660,SMITH
//         EXEC PGM=IEBPTPCH
//SYSPRINT DD   SYSOUT=A
//SYSIN    DD   DSNAME=PDSLIB(PNCHSTMT),DISP=(OLD,KEEP)
//SYSUT2   DD   UNIT=2540-2
//SYSUT1   DD   DATA
(input card data set including // cards)
/*
```

The control statements are discussed below:

- SYSIN DD defines the control data set. The control data set contains a PUNCH statement and is defined as a member of the partitioned data set PDSLIB. (The data set is cataloged. The RECFM must be FB and the LRECL must be 80.)
- SYSUT2 DD defines the output data set. The data set is to be punched on an IBM 2540-2 Card Read Punch (punch feed).
- SYSUT1 DD defines the input card data set, which follows in the input stream.

IEBPTPCH Example 9

In this example a record group is to be printed. A user routine is provided to manipulate output records before they are printed.

The example follows:

```
//PRINT    JOB  09#660,SMITH
//         EXEC PGM=IEBPTPCH
//SYSPRINT DD   SYSOUT=A
//SYSUT1   DD   DSNAME=SEQDS,UNIT=2311,DISP=(OLD,KEEP),
// LABEL=(,SUL),VOLUME=SER=111112,
// DCB=(RECFM=FB,LRECL=80,BLKSIZE=2000)
//SYSUT2   DD   SYSOUT=A
//SYSIN    DD   *
       PRINT   MAXFLDS=2,MAXGPS=1,MAXLITS=6,STOPAFT=32767
       TITLE   ITEM=('TIMECONV-DEPT D06'),ITEM=('JAN 10-17')
       EXITS   OUTREC=NEWTIME,INHDR=HDRS,INTLR=TLRS
      RECORD   IDENT=(6,'498414',1),                          ¢C
               FIELD=(8,1,,10),FIELD=(30,9,XE,20)
      LABELS   DATA=ALL,CONV=XE
/*
```

The control statements are discussed below:

- SYSUT1 DD defines the input data set. The data set resides on a 2311 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 control data set contains the PRINT, TITLE, EXITS, and RECORD statements.
- The PRINT statement: (1) initializes the print operation, (2) indicates that two **FIELD** parameters are included in subsequent RECORD statements, (3) indicates that one **IDENT** parameter is included in a subsequent RECORD statement, (4) indicates that six literal characters are included in the subsequent **IDENT** parameter, and (5) indicates that processing is to be terminated after 32,767 records are processed or after the first record group is processed, whichever comes first. Because **MAXLINE** is omitted, 60 lines are printed on each page.
- TITLE specifies a title.
- EXITS specifies the name of a user routine (NEWTIME), which is used to manipulate output records before they are printed.

- RECORD defines the record group to be processed and indicates where information from the input records is placed in the output records. Bytes 1 through 8 of the input records appear in columns 10 through 17 of the punched output, and bytes 9 through 38 are printed in hexadecimal representation and placed in columns 20 through 79.
- 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 to be converted from alphameric to hexadecimal representation.

IEBTCRIN Program

IEBTCRIN is a data set utility used to read input from the IBM 2495 Tape Cartridge Reader (TCR), edit the data as specified by the user, and produce a sequentially organized output data set. (See "Introduction" for general data set utility information.)

The input to IEBTCRIN is in the form of cartridges written by either the IBM Magnetic Tape SELECTRIC Typewriter (MTST) or the IBM 50 Magnetic Data Inscriber (MTDI). An input data set (one or more cartridges) must consist of either all MTST cartridges or all MTDI cartridges.

IEBTCRIN can be used to construct records from the stream of data bytes read sequentially from the Tape Cartridge Reader. The user has the option of gaining temporary control (via a user-supplied exit routine) to process each logical record.

When MTDI input is edited, IEBTCRIN maintains information about each record as it is being edited. This information is summarized in the Error Description Word (EDW) which is described below. When the EDW contains a nonzero value in either the level status (byte 0) or the type status (byte 1), the record is considered an error record by the program and the EDW is appended to the start of the record to aid the user in analyzing the error.

Error Records

If a record is found to be in error, the record is passed to the user error exit routine if one is specified. If an error exit is not specified, the action to be taken is determined by the option specified in a utility control statement.

When either MTST input or MTDI input without editing is specified, the only error that can be recognized is a record containing one or more permanent data checks. The data check bytes are replaced as described in a utility control statement. The record is considered an error record, but because a data check is the only error that can occur, no EDW is appended to the error record.

Error Description Word (EDW)

The Error Description Word (EDW) consists of four bytes that are appended to the start of an error record.

The error description word is in EBCDIC format; for example, a 2 is represented as X'F2' and a C is represented as X'C3'. The information provided in each of the four bytes of the EDW is discussed below.

Level Status (Byte 0)

The level status indicator identifies error records that result from inter-record dependency that cannot be identified in the type status byte.

The level status is presented with each error record and has a value of:

- 0, for any error record that will not cause questionable data in following records. A nonzero type status accompanies this byte.
- 1, for any error record that may cause questionable data in following records, and for which the level status of the previous record was 0.
- 2, for any error record that contains questionable data because the error level of the preceding record was 1 or 2, or for any error record that may cause questionable data in the following records and for which the level status of the previous record was 1 or 2.

A level status of other than 0 is presented with error records resulting from the following:

- The start-of-record (SOR) location has a character defined as an error.
- The record contains two or more data check bytes side by side. These may have been an SOR and EOR.
- The record is longer than the user-specified maximum length record.
- The length of the record is not equal to the length of the first valid record of the same program level encountered on this cartridge. For this purpose, a valid record is one that contains no errors as identified in the type status, with the possible exception of being shorter than the user-specified minimum length.
- The record has a data-duplication dependency on a previous record with one of the above errors.

The level status is set to 0 when IEBTCRIN encounters: (1) a record without one of the previous errors, (2) a canceled record, or (3) the first record of a cartridge.

Type Status (Byte 1)

The type status indicator identifies records in error because of SOR, EOR, length, field, or data check error conditions.

The type status is presented with each error record and has a value of:

- 0, for any record that contains none of the following identifiable errors, but contains questionable data due to a nonzero level status. (See "Level Status" earlier in this chapter.)
- 1, for any record that has: (1) an SOR character of other than P1 through P8 or a GS code, (2) an EOR character of other than a VOK code for records when the user specified a record verification check, or (3) an EOR character of other than a VOK or RM code for records when the user specified no record verification check.
- 2, for any record that has an incorrect length because it is: (1) longer than the user-specified maximum, (2) shorter than the user-specified minimum, or (3) not equal to the length of the first valid record of the same program level encountered on this cartridge.
- 4, for any record that has a field error. A field error occurs when duplication or left-zero justification functions did not occur in a field due to an error condition. See "MTDI Editing Criteria" below.
- 8, for any record that has a permanent data check error.

The type status indicator can also have values of 3, 5, 6, 7, 9, A, B, C, D, E, and F. These values indicate a combination of SOR, EOR, length, field, and data check errors. For example, a value of A indicates a record with a data check error (8) as well as an incorrect length (2).

Start-of-Record (Byte 2)

This byte contains an indication of the start-of-record (SOR) character associated with this record. The SOR character can be 1 through 8, where 1 indicates P1, 2 indicates P2, etc., or E, which indicates that the SOR character is in error.

End-of-Record (Byte 3)

This byte contains an indication of the end-of-record (EOR) character associated with this record. The EOR character can be: U, which indicates an unverified record; V, which indicates a verified record; or E, which indicates that the EOR character is in error.

Sample Error Records

Figure 34 shows a stream of data bytes read sequentially from the tape cartridge reader.

Figure 35 shows the records constructed by IEBTCRIN from the input records shown in Figure 34. These records show some of the errors that can occur during processing and their effect on the Error Description Word. The following parameters were specified on the TCRGEN statement for these records:

```
TCRGEN TYPE = MTDI,EDIT = EDITR,VERCHK = VOKCHK,                    ℛC
       MAXLN = 50,REPLACE = X'5B'
```

IEBTCRIN classifies records 2 through 9 in Figure 35 as error records. The records are classified, as follows:

- Record 1 is a valid record. It contains a program level 1 code, and thus establishes the valid length for all program level 1 records in this cartridge to be 25 bytes.
- Record 2 has a data check in the SOR location. Level status is set to 1 because the SOR location might have contained a cancel code that would cause any data duplicated on the following record to be questionable. The type status (9) indicates the record has an incorrect SOR/EOR character (1) and a data check error (8).
- Record 3 contains no identifiable error, but contains questionable data because it requires duplication from the previous record, which had a level status of 1.
- Record 4 has a data check. Because it contained no DUP codes, the level status is set to 0.
- Record 5 is shorter than the first program level 1 record on this cartridge (length error). This record also contains an RM code rather than a VOK code in the EOR location (VOKCHK was specified). Because IEBTCRIN cannot determine why the record is short, all data duplicated from this record is questionable; the level status is set to 1. The type status is set to 3 indicating an SOR/EOR error (1) and length error (2).
- Record 6 contains a DUP code that is beyond the last position of the preceding record.

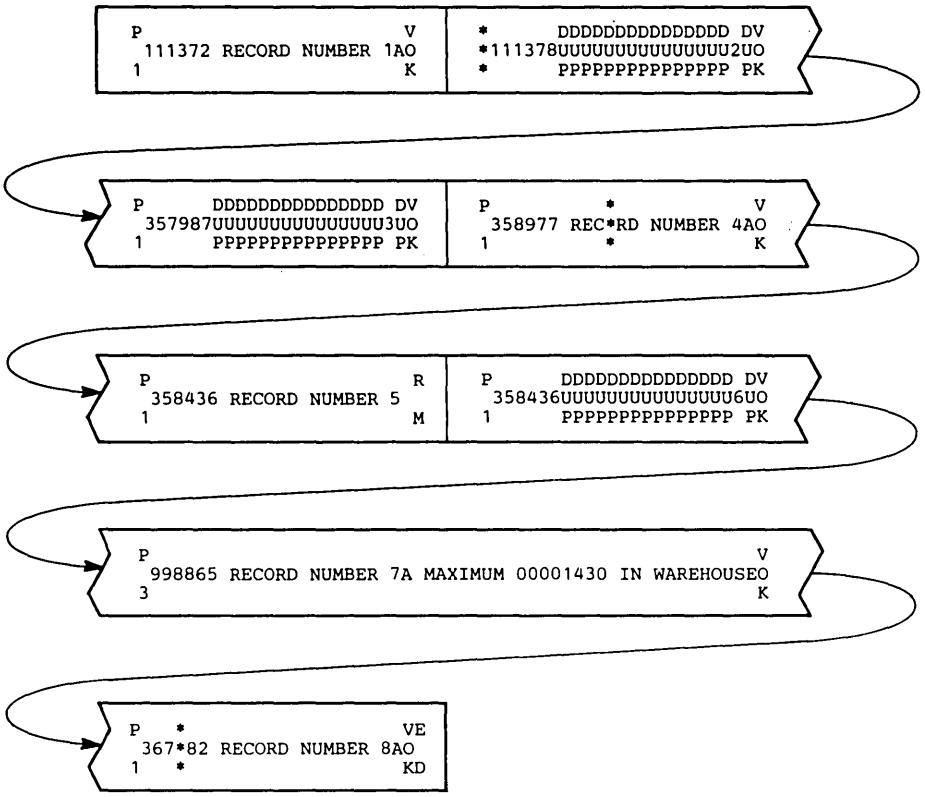


Figure 34. Tape Cartridge Reader Data Stream

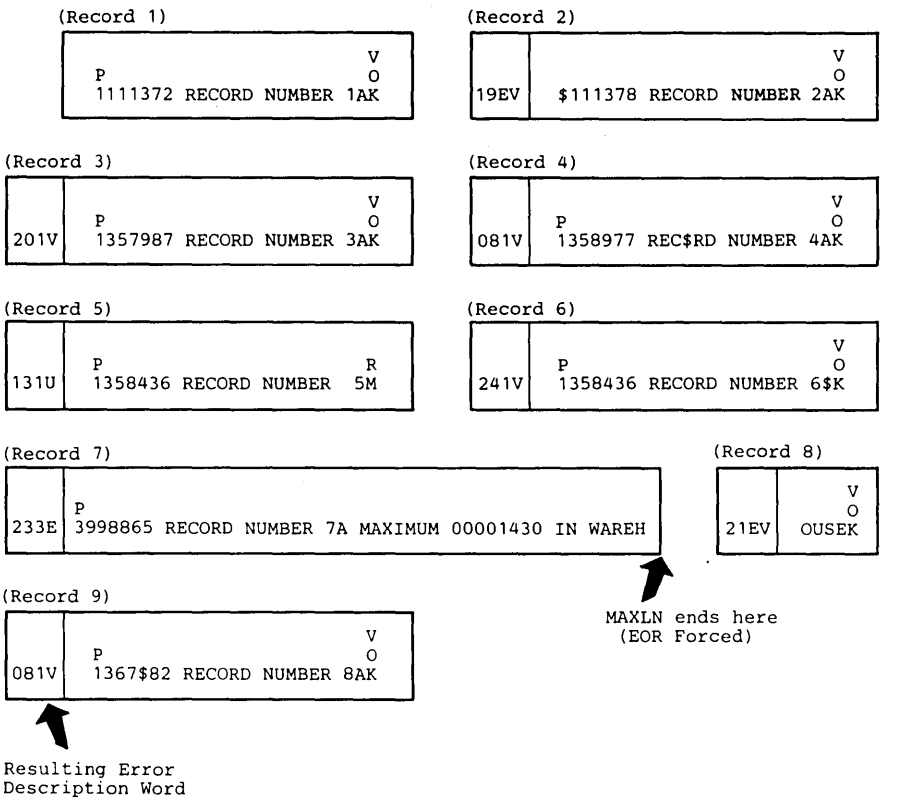


Figure 35. Record Construction

- The seventh input record is longer than the maximum user-specified record length. Note that it is passed as two records. The first record (record 7) indicates an EOR error and a length error; the second (record 8) indicates an SOR error. Because record 7 is an error record, its length (50 bytes) is not established as the valid length for all program level 3 records on this cartridge.
- Record 9 has a data check. Because it contained no DUP codes, the level status is set to 0.

MTDI Editing Criteria

The cartridges created on the IBM 50 Magnetic Data Inscriber contain a continuous stream of data bytes (that is, there are no interblock gaps). Therefore, when editing is specified, IEBTCRIN extracts records one at a time from the data stream. To accomplish this, IEBTCRIN scans for control codes written by MTDI. IEBTCRIN uses start-of-record (SOR) and end-of-record (EOR) locations to extract MTDI records from the input stream.

The (SOR) location is defined as:

- The location of the first character on a cartridge.
- The location of the first character after the previous record's (EOR) location.
- The location of an SOR code.
- The location of a GS code.

The character in the SOR location is checked to determine if it is a valid start-of-record character. A P1 through P8, a cancel code, or a GS code are valid start-of-record characters; all others are invalid.

The EOR location by priority sequence is:

1. The same location as the SOR location, if the SOR character was a valid GS code.
2. The location of the first encountered RM or VOK code if that location is within the length of the maximum user-specified record size.
3. The location of any code preceding either a valid SOR code or the end-of-media code, if that location is within the length of the maximum user-specified record size.
4. The location determined in 2 or 3, regardless of the maximum user-specified record size if the SOR location contains a cancel code.
5. If one of the previous EOR locations cannot be defined, an EOR condition will be forced at the location where the record length equals the maximum user-specified record size.

The character in the EOR location is checked to determine if it is a valid end-of-record character. Valid EOR characters are the GS character (if the SOR character was a GS code) and VOK or RM codes; all others are invalid. Each GS code is considered a valid SOR code or EOR code and will be bypassed.

MTDI Editing Restrictions

Following are the restrictions that apply when editing MDTI records:

- All canceled records are bypassed; they are not passed to any exit routines or written on any data sets. The level status is set to 0.
- All input records less than three bytes in length (SOR location, one data byte, and EOR location) are treated as canceled records. The remaining portion of a record that was longer than the user-specified maximum record size can result in an input record of this size.
- Data duplication is accomplished by replacing the DUP code with the character from the corresponding location of the previous record.
- The record used for data duplication is the record returned from any user exits.
- GS codes will not affect the level status or duplication of following records.
- Data duplication does not occur for any of the following conditions:
  1. The DUP code is encountered in the first record of a cartridge.
  2. The DUP code is encountered in a record immediately following a canceled record. A canceled record is one that contains a cancel code in the SOR location or an input record of less than three bytes as described above.
  3. The DUP code is encountered in a position that would cause duplication of a position beyond the last data byte of the previous record.
  4. The DUP code is encountered in a position that would cause duplication of an error-replace character.

In each case, the DUP code is replaced with the user specified error-replace character, and a field error is indicated.

- Left-zero justification does not occur; the left-zero fill code (LZ) is replaced with the user-specified error-replace character and a field error is indicated for either of the following conditions:
  1. The left-zero fill code (LZ) is encountered without first having encountered its corresponding left-zero start code (LZS).
  2. The user-specified maximum record size is exceeded before encountering the valid end of a left-zero field.

End-of-Cartridge

Unique codes, written by the MTST or the MTDI device, signal the program when all data on a cartridge has been read. For MTST cartridges, this end-of-cartridge code is a lowercase stop code (st) or an uppercase stop code (ST). For MTDI cartridges, the end-of-cartridge code is the end-data code (ED).

IEBTCRIN terminates input from a cartridge upon encountering the end-of-cartridge code and rewinds the cartridge. IEBTCRIN continues to process cartridges until end-of-file is encountered.

End-of-file is signaled following a rewind operation when there are no more cartridges in the feed hopper, the END OF FILE button is pressed, and end-of-cartridge for the last cartridge is recognized. An end-of-file indication will be passed to the OUTREC and/or ERROR exits if specified by setting register 1 equal to 0.

Input and Output

IEBTCRIN uses the following input:

- An input data set, which contains data on tape cartridges to be read from the Tape Cartridge Reader (TCR). The input data set was created on either MTST or MTDI.
- A control data set, which contains utility control statements that are used to control the functions of IEBTCRIN.

IEBTCRIN produces the following output:

- An output data set, which contains the sequential output produced by the utility as a result of processing the cartridge input according to the utility control statements.
- An error output data set, which contains records that do not conform to the specifications for a valid record.
- A message data set, which contains diagnostic messages.

Control

IEBTCRIN is controlled by job control statements and utility control statements. The job control statements are required to execute or invoke IEBTCRIN and to define the data sets that are used and produced by the program. The utility control statements are used to indicate the source of the input data cartridges (MTST or MTDI) and to specify the type of processing to be done.

Job Control Statements

Table 25 shows the job control statements necessary for using IEBTCRIN.

Table 25. IEBTCRIN Job Control Statements

Statement	Use
JOB	Initiates the job.
EXEC	Specifies the program name (PGM = IEBTCRIN) or, if the job control statements reside in a procedure library, the procedure name.
SYSPRINT DD	Defines a sequential message data set, which can be written to any QSAM-supported output device.
SYSUT1 DD	Defines the input data set.
SYSUT2 DD	Defines a sequential output data set for valid records.
SYSUT3 DD	Defines a sequential output data set for error records.
SYSIN DD	Defines the control data set. The control 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. If this statement is not included, all utility control statement defaults are assumed and a message is issued to SYSPRINT. If DUMMY is specified, all utility control statement defaults are assumed.

The minimum region size that can be specified for the execution of IEBTCRIN is 12K + 2b + c + e, where (1) b is value specified for BUFL on the SYSUT1 DD statement, (2) c is the maximum logical record length, and (3) e is the sum of user exit routines, each rounded to the next higher 2K.

If the SYSPRINT DD statement is missing, a message is written on the operator console and processing continues.

If some parameters are specified but others are omitted, IEBCTRIN attempts to set defaults for the missing parameters that are consistent with those supplied. For example, if RECFM = VBA is specified, IEBTCRIN assumes BLKSIZE = 129 and LRECL = 125. If LRECL, BLKSIZE, and RECFM are not specified, the defaults are LRECL = 121, BLKSIZE = 121, and RECFM = FBA.

For the SYSUT1 DD statement, only the UNIT keyword is required. The value specified in UNIT = xxxx can be '2495', the device address, or any other name that was generated in the system as a unit device name. The VOLUME = SER = keyword may be specified to identify the tape cartridges to be mounted. The volume serial number must be an externally recognizable name associated with the cartridges to be processed. A message is issued to the operator instructing that the cartridges identified by that name be mounted. If VOLUME is not specified, the name TCRINP is assumed and used in the mount message. The BUFL DCB parameter can be specified to indicate the size of input buffers; if BUFL is not specified, a value of 2000 is assumed.

Fixed and variable records on the SYSUT2 or SYSUT3 data set can be blocked through the specification of the BLKSIZE and RECFM DCB parameters.

SYSUT2 DD and SYSUT3 DD statements may be omitted or specified as DUMMY. A message is issued on SYSPRINT and processing continues.

The DCB parameters defining the SYSIN, SYSPRINT, SYSUT2, and SYSUT3 data sets can be supplied from any valid source (for example, DD statements or a data set label). Because the output (SYSUT2 and/or SYSUT3) data sets are not opened until the first record is ready for output (after any OUTREC and/or ERROR exits), DCB parameters to be supplied from an existing data set label are not available for records constructed before the data set is opened. Therefore, the DCB parameters should always be provided in the DD statement even though they may already exist in the label. Otherwise, defaults are used to construct records until the data set is opened.

If a permanent error occurs on SYSIN, SYSUT1 (not including a data check), SYSUT2, or SYSUT3, a message is issued on SYSPRINT and the program is terminated. If a permanent input/output error occurs on SYSPRINT, both the failing message and a SYNADAF message indicating the error, are written on the programmer's console and processing is terminated.

Restrictions

- Because IEBTCRIN always constructs the SYSPRINT records with USASI (type A) control characters, type A control characters should be indicated when RECFM is specified.
- If a parameter is specified on SYSPRINT DD that is not consistent with the other parameters, a message is issued and processing is ended.
- The SYSUT1 DD statement is required for each use of IEBTCRIN.
- The SYSUT2 DD and SYSUT3 DD statements must identify sequential data sets; the data sets can have fixed, variable, variable spanned, or undefined records. These data sets can be written on any QSAM-supported device.
- If editing of MTDI input is specified on the utility control statements, the SYSUT3 LRECL parameter should be four bytes greater than the SYSUT2 LRECL parameter to include a four-byte Error Descriptor Word appended to the front of the record by IEBTCRIN. (See "Error Records" earlier in this chapter.) For variable records on either SYSUT2 or SYSUT3, the LRECL and BLKSIZE DCB parameters must be large enough to include the four-byte record descriptor word.
- If inconsistent parameters are specified on SYSUT2 DD or SYSUT3 DD, a message is issued and processing is ended.

Utility Control Statements

IEBTCRIN is controlled by the following utility control statements:

- TCRGEN statement, which specifies whether MTDI or MTST input is to be processed and the type of processing to be performed.
- EXITS statement, which specifies any exit routines provided by the user.

If these statements contain errors or inconsistencies, the program is terminated and the appropriate diagnostics are sent to the message data set. If TCRGEN is not specified, standard defaults are used.

Note: If TCRGEN or EXITS is specified, the operand must be made up of one or more parameters.

TCRGEN Statement

The TCRGEN statement is used to indicate the device (MTDI or MTST) on which the input data was created and the type of processing to be performed on the input data.

The format of the TCRGEN statement is:

```
[label] TCRGEN [TYPE = {MTDI  }
                       {MTST  }]
               [,TRANS = {STDUC  }
                         {STDLC  }
                         {name   }
                         {NOTRAN }]
               [,EDIT = {EDITD  }
                        {EDITR  }
                        {NOEDIT }]
               [,VERCHK = {NOCHK  }
                          {VOKCHK }]
               [,MINLN = n ]
               [,MAXLN = n]
               ,REPLACE = X'xx'
               [,ERROPT = {NORMAL }
                          {NOERR  }]
```

where:

TYPE =
specifies the device on which the magnetic tape cartridge(s) was written. These values can be coded:

MTDI
specifies that the input was created on a Magnetic Data Inscriber. This is the default.

MTST
specifies that the input was created on a Magnetic Tape SELECTRIC typewriter.

TRANS =
specifies the type of processing to be performed on MTST input. These values can be coded:

STDUC
specifies that the MTST code is to be translated to standard EBCDIC; alphabetic characters are translated to uppercase. This is the default.

STDLC
specifies that the MTST code is to be translated to standard EBCDIC; alphabetic characters are not translated to uppercase.

name
specifies a user-translate table to be used by IEBTCRIN. The translate table must exist as a load module named in a user job library or the link library. This load module must consist of a translate table which begins at the entry point and conforms to the specifications for the translate instruction (TR) found in *IBM System/360 Principles of Operation*, GA22-6821.

NOTRAN
specifies that no translation and no special processing is to be performed. Data is passed exactly as read from the cartridge.

EDIT =
specifies the type of processing to be performed on MTDI input. These values can be coded:

EDITD
specifies that the input is to be edited and that SOR and EOR codes are to be deleted and not included as part of the output record. This is the default.

EDITR
specifies that the input is to be edited and SOR and EOR codes are to be kept as part of the output record.

NOEDIT
specifies that no editing is to be performed. Data, including any group separator (GS) codes, is passed exactly as read from the cartridge.

VERCHK =
specifies whether a record-verification check is to be made on MTDI input that is to be edited. This parameter is valid only when TYPE = MTDI and either EDIT = EDITD or EDIT = EDITR are specified. These values can be coded:

NOCHK
specifies that no record-verification check is to be made. Either a record mark (RM) or a verify OK (VOK) code is considered a valid end-of-record code. This is the default.

VOKCHK
specifies that a record-verification check is to be made. A record that does not contain a verify OK code is to be considered an error record.

MINLN = *n*
specifies in bytes the length, *n*, of the shortest valid edited record. This parameter is valid only when TYPE = MTDI and either EDIT = EDITD or EDIT = EDITR are specified. If IEBTCRIN encounters a record shorter than this specified length, the record is considered an error record. If **MINLN** is omitted, no minimum length checking is performed.

MAXLN = *n*
specifies the number of bytes, *n*, plus four for the record descriptor word when variable records are specified, to be contained in all but the last record passed to the output routine when editing is not performed. IEBTCRIN does not indicate the end of data from one cartridge and the beginning of data from the next. Usually this transition from one cartridge to another occurs within an output record. The last record passed to the output routine contains only the number of bytes remaining (plus four if the record format is variable) and is the only record that can be shorter than the length specified by **MAXLN**. The size of the records actually written depends on the record length (LRECL) specified for the output data set. If **MAXLN** is omitted, a value of 120 is assumed.

REPLACE = **X'***xx***'**
specifies the hexadecimal representation of the character to be used by IEBTCRIN to replace error bytes. **REPLACE** allows the user to identify and possibly correct error bytes in the error exit routine or in subsequent processing. The specified **REPLACE** character should be one that does not normally appear in the data. X'19', end-of-data, is assumed if **REPLACE** is not coded. To replace error bytes on MTDI data, select a value for *xx* from Figure 36. to replace error bytes on MTST data, select a value for *xx* from Figure 37. The replacement of error bytes is accomplished before any specified MTST translation.

ERROPT =
specifies the disposition of all error records. **ERROPT** is ignored if a user error routine is specified in the EXITS statement. These values can be coded:

NORMAL
specifies that all error records are to be placed in the error data set (SYSUT3).

NOERR
specifies that all records (including error records) are placed in the normal output data set (SYSUT2). No records are placed in the error data set (SYSUT3). This is the default.

If **STDUC**, **STDLC**, or *name* is specified, certain of the MTST codes are processed in a special way before translation. Feed codes (FD), switch codes (SW), and autosearch codes (AS), both uppercase and lowercase, are deleted from the data. Each 61-character reference code is reduced to a single search code (SRC).

A stop code, whether uppercase (ST) or lowercase (st), indicates that all data on a cartridge has been read. Therefore, when an MTST cartridge to be processed by IEBTCRIN is created, the user must not use a stop code for any purpose other than signaling end-of-data on the cartridge. Stop codes within meaningful data cause any subsequent data on the cartridge to be lost because the cartridge is rewound and unloaded when a stop code is encountered.

If **EDITD** or **EDITR** is specified, the edit consists of the following functions:

- Records are extracted one at a time from the input buffers by scanning for the record-delimiting codes (SOR and EOR).
- DUP codes are replaced with the character from the corresponding location in the preceding record.
- Left-zero fields are right aligned and leading zeros are inserted where necessary.

- Left-zero start codes are deleted from the records.
- Group separator codes and records that start with cancel record codes are bypassed.

For MTDI input with editing specified, **MAXLN** is used to specify in bytes the length of the longest valid record after editing. If the program encounters a record in which a valid end-of-record cannot be determined within this length, an end-of-record condition is forced and the record is considered an error record.

The values that can be specified for **MINLN** and **MAXLN** are:

- For MTST processing or MTDI processing without editing, **MINLN** is not specified. **MAXLN** should equal the number of bytes to be passed as a record.
- For MTDI processing when EDIT = EDITD, **MINLN** should equal the number of bytes in the shortest valid record after editing, excluding SOR and EOR codes. **MAXLN** should equal the number of bytes in the longest valid record after editing, excluding SOR and EOR codes.
- For MTDI processing when EDIT = EDITR, **MINLN** should equal the number of bytes in the shortest valid record after editing, including SOR and EOR codes. **MAXLN** should equal the number of bytes in the longest valid record after editing, including SOR and EOR codes.

Note: The values for **MINLN** and **MAXLN** should not include the four-byte record descriptor word added to a variable length record.

Table 26 shows the hexadecimal characters representing special purpose codes that must not be used as replacement bytes.

Table 26. Special Purpose Codes

MTDI Codes

X'00'	(LZ)	X'1E'	(VOK)	X'74'	(P4)
X'11'	(DUP)	X'3C'	(RM)	X'75'	(P5)
X'12'	(LZS)	X'71'	(P1)	X'76'	(P6)
X'18'	(CAN)	X'72'	(P2)	X'77'	(P7)
X'1D'	(GS)	X'73'	(P3)	X'78'	(P8)

MTST Codes

X'10'	(cr)	X'14'	(CR)	X'51'	(as)
X'11'	(sw)	X'15'	(SW)	X'55'	(AS)
X'13'	(fd)	X'17'	(FD)	X'80'	(src)
				X'81 through X'FF'	

The special purpose codes listed in Table 26 are used by IEBTCRIN when constructing records. Use of these codes causes a message to be issued and the utility to be terminated.

Figure 36 shows the values that can be chosen to replace error bytes for MTDI input.

Figure 37 shows the values that can be chosen to replace error bytes for MTST input.

Figure 38 shows MTST codes after they have been translated by IEBTCRIN when TRANS = STDLC is specified.

EXITS Statement

The EXITS statement is used to identify user-supplied exit routines, which must exist in either the user job library or the link library.

Upon entry, a parameter list is supplied to the exit routine Upon returning from the exit routine, the user must provide an acceptable return code. See "Appendix A: Exit Routine Linkage."

The format of the EXITS statement is:

```
[label] EXITS  [ERROR = routinename]
               [,OUTREC = routinename]
               [,OUTHDR2 = routinename]
               [,OUTHDR3 = routinename]
               [,OUTTLR2 = routinename]
               [,OUTTLR3 = routinename]
```

where:

ERROR = *routinename*
specifies the symbolic name of a routine that receives control before an error record is passed to the error output data set (SYSUT3). This exit routine can be used to analyze and, if possible, correct the error record. This parameter nullifies any **ERROPT** value.

Bit Positions 4, 5, 6, 7	Second Hexadecimal Digit	00				01				10				11			
		00	01	10	11	00	01	10	11	00	01	10	11	00	01	10	11
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000	0	LZ				SP	&	-							0	082	0
0001	1		DUP					/	P1					A	J		1
0010	2		LZS						P2					B	K	S	2
0011	3								P3					C	L	T	3
0100	4								P4					D	M	U	4
0101	5								P5					E	N	V	5
0110	6								P6					F	O	W	6
0111	7								P7					G	P	X	7
1000	8		CAN						P8					H	Q	Y	8
1001	9		ED											I	R	Z	9
1010	A					¢	!		:								
1011	B					.	$	,	#								
1100	C				RM	<	*	%	@								
1101	D		GS			(	)	-	/								
1110	E		VOK			+	;	>	=								
1111	F					|	¬	?	"								

Bit Positions 0, 1

Bit Positions 2, 3

First Hexadecimal Digit

Special Control:

LZ = Left zero fill
DUP = Duplicate
LZS = Left zero start
ED = End data
GS = Group Separator

Start of Record (SOR):

P1 = Program level 1
P2 = Program level 2
P3 = Program level 3
P4 = Program level 4
P5 = Program level 5
P6 = Program level 6
P7 = Program level 7
P8 = Program level 8
CAN = Cancel

End of Record (EOR):

RM = Record mark
VOK = Verify OK

This figure represents the character set and control codes as read from an MTDI created cartridge.

Figure 36. MTDI Codes from TCR

OUTREC = *routinename*
specifies the symbolic name of a routine that receives control before the record is passed to the normal output data set (SYSUT2). In this exit routine, the user can process the record and perform his own output if output other than the SYSUT2 data set is desired. Any modification of an edited MTDI record may affect the editing of following records because the record returned from this exit is used to accomplish data duplication in the record that follows. If the SYSUT2 data set has specified variable length records, a four-byte RDW is appended to the front of the record.

OUTHDR2 = *routinename*
specifies the symbolic name of a routine that receives control during the opening of the SYSUT2 data set; this exit routine can be used to create user output header labels for the normal output data set (SYSUT2).

OUTHDR3 = *routinename*
specifies the symbolic name of a routine that receives control during the opening of the SYSUT3 data set; this exit routine can be used to create user output header labels for the error data set (SYSUT3).

OUTTLR2 = *routinename*
specifies the symbolic name of a routine that receives control during the closing of the SYSUT2 data set; this exit routine can be used to create user output trailer labels for the normal output data set (SYSUT2).

OUTTLR3 = *routinename*
specifies the symbolic name of a routine that receives control during the closing of the SYSUT3 data set; this exit routine can be used to create user output trailer labels for the error data set (SYSUT3).

If MTDI is edited, a four-byte (EDW) is appended to the front of each error record describing the error condition. For further definition of the EDW, see "Error Records" earlier in this chapter. If the SYSUT3 DD statement specified variable length records, a four-byte Record Descriptor Word (RDW) is also appended to the front of the record. For further description of the RDW, see *OS Supervisor Services Guide*, GC28-6646.

Bit Positions 4, 5, 6, 7	Second Hexadecimal Digit	00				01				10				11			
		00	01	10	11	00	01	10	11	00	01	10	11	00	01	10	11
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000	0	z	cr	5	0	l	tab	'	s	src							
0001	1	2	sw	6	9	.	as	i	w								
0010	2	t		e	h	j	sp	p	y								
0011	3	n	fd	k	b	=		q	-								
0100	4	Z	CR	%	)	°	TAB	"	S	SRC							
0101	5	@	SW	¢	(	•	AS	I	W								
0110	6	T		E	H	J	SP	P	Y								
0111	7	N	FD	K	B	+		Q	-								
1000	8	1		7	4	m	bsp	r	o								⁄
1001	9	3	st	8		v		a									
1010	A	x		d	l	g		:	/								
1011	B	u		c		f	stx	,									
1100	C	±		&	$	M	BSP	R	O								
1101	D	#	ST	*		V		A									
1110	E	X		D	L	G		:	?								
1111	F	U		C		F	STX	,									

Bit Positions 0, 1
Bit Positions 2, 3
First Hexadecimal Digit

cr and CR = Carrier return code
sw and SW = Switch code
fd and FD = Feed code
st and ST = Stop code
tab and TAB = Tab code
as and AS = Automatic search
sp and SP = Space
bsp and BSP = Backspace
stx and STX = Stop transfer
src and SRC = Search

This figure represents the character set and control codes as read from an MTST created cartridge.

Figure 37. MTST Codes from TCR

The user-supplied routines specified in **ERROR** and **OUTREC** can be used to examine and modify any byte in the record or EDW. The record length can be changed, subject to the following restrictions:

- A work area used to construct the records is allocated by the program equal in size to the largest of (1) **MAXLN**, (2) LRECL on SYSUT2, or (3) LRECL on SYSUT3.
- The record length must not be increased beyond this size. Overlaying of other work areas may then occur, causing unpredictable results.

The new record length must be placed in the location pointed to by the second parameter word as received at entry to the routine. This length must include the EDW and RDW (if applicable). It is not necessary to modify the RDW because it is

re-created if the record is to be written by IEBTCRIN. However, if the user does his own output from this routine, he must ensure that the RDW is correct for the record.

Bit Positions 4, 5, 6, 7	Second Hexadecimal Digit	00				01				10				11				Bit Positions 0, 1
		00	01	10	11	00	01	10	11	00	01	10	11	00	01	10	11	Bit Positions 2, 3
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	First Hexadecimal Digit
0000	0					SP	&	-									0	
0001	1							/		a	j	°		A	J		1	
0010	2			STX						b	k	s		B	K	S	2	
0011	3									c	l	t		C	L	T	3	
0100	4									d	m	u		D	M	U	4	
0101	5	TAB								e	n	v		E	N	V	5	
0110	6		BSP							f	o	w		F	O	W	6	
0111	7									g	p	x		G	P	X	7	
1000	8									h	q	y		H	Q	Y	8	
1001	9									i	r	z		I	R	Z	9	
1010	A					¢	!		:									
1011	B					.	$	,	#									
1100	C						*	%	@									
1101	D	CR				(	)	—	'									
1110	E		SRC			+	;		=		±							
1111	F							?	"									

TAB = Tab code
CR = Carrier return
BSP = Backspace
SRC = Search
STX = Stop transfer
SP = Space

Note: The STDUC option permits translating both lowercase and uppercase alphabetic characters to uppercase.

Figure 38. MTST Codes after Translation by IEBTCRIN with TRANS = STDCL

If IEBTCRIN is to write the record, the length of the output record depends on the RECFM specification, as follows:

- Fixed and variable records may have a maximum length equal to LRECL. Records larger than this are truncated.
- Undefined records may have a maximum length equal to BLKSIZE. Records larger than this are truncated.

These record lengths include the EDW and RDW, where applicable.

The record length returned from the error exit is used to establish the location of the last data byte in the record. The location is used to control data duplication in the following record. However, it is not used for checking the record length of subsequent records.

Modifications to the EDW, record, or record length may affect the editing of subsequent records. If the input is not edited, the user can examine and modify any byte in the record. The record length can also be changed, subject to the MTDI editing restrictions.

Return Codes from IEBTCRIN

At job termination, IEBTCRIN produces a return code to indicate the results of program execution. Table 27 shows the return codes used by IEBTCRIN.

Table 27. IEBTCRIN Return Codes

Return Code	*Interpretation*
00	Normal termination.
04	Warning message issued; execution permitted. Conditions leading to issuance of this code are: (1) SYSPRINT, SYSIN, SYSUT2, or SYSUT3 DD statements missing and (2) DCB parameters missing in SYSUT2 or SYSUT3 DD statements.
12	Diagnostic error message issued; execution terminated. Conditions leading to issuance of this code are: (1) SYSUT1 DD statement missing, (2) conflicting DCB parameters in DD statements, and (3) invalid or conflicting utility control statements.
16	Terminal error message issued; execution terminated. Conditions leading to issuance of this code are: (1) permanent input/output errors (not including data checks on the TCR), (2) unsuccessful opening of data sets, (3) requests for termination by user exit routine, (4) insufficient storage available for execution, and (5) user exit routine not found.

IEBTCRIN Examples

The following examples illustrate some of the uses of IEBTCRIN. Table 28 can be used as a quick reference guide to IEBTCRIN examples. The numbers in the "Example" column point to examples that follow.

Table 28. IEBTCRIN Example Directory

Operation	Data Set Organization	Device	Comments	Example
Edit MDTI input	Sequential	2314 Disk, 9-track tape	Fixed blocked output. Error exit routine specified	1
Invoke IEBTCRIN with LINK macro instruction				2

IEBTCRIN Example 1

In this example, input from a tape cartridge is to be edited with normal records written to a 2314 volume and error records written to a 9-track tape volume.

The example follows:

```
//JOBNAME  JOB     0,SMITH,MSGLEVEL=1
//STPNAME  EXEC    PGM=IEBTCRIN
//SYSPRINT DD      SYSOUT=A
//SYSUT1   DD      UNIT=TCR,VOLUME=SER=MYTAPE,DCB=(BUFL=3000)
//SYSUT2   DD      DSNAME=GOODSET,DISP=(NEW,CATLG),UNIT=2314,
// VOLUME=SER=111222,SPACE=(TRK,(10,10)),
// DCB=(LRECL=100,BLKSIZE=1000,REDFM=FB)
//SYSUT3   DD      DSNAME=ERRSET,UNIT=2400,
// VOLUME=SER=000001,DISP=(NEW,KEEP),
// DCB=(BLKSIZE=104,RECFM=U)
//SYSIN    DD      *
       TCRGEN TYPE=MTDI,EDIT=EDITD,MAXLN=100,REPLACE=X'5B'
       EXITS  ERROR=MYERR
/*
```

The control statements are discussed below:

- SYSUT1 DD defines the input tape cartridge data set. A console message instructs the operator to mount a set of cartridges named MYTAPE. The two input buffers are each 3000 bytes long (BUFL). The UNIT parameter assumes that TCR has been system generated as a unit name for the Tape Cartridge Reader.
- SYSUT2 DD defines a sequential data set for the normal output records. The data will be written to a 2314 volume.
- SYSUT3 DD defines a sequential data set for the error records. The records are undefined with a maximum block size of 104 bytes, including a 4-byte error description word.
- SYSIN DD defines the control data set, which follows in the input stream.
- TCRGEN indicates MTDI input. The input is to be edited with SOR and EOR codes deleted, the maximum valid record length is to be 100 bytes, and the replace character is a hexadecimal "5B". **VERCHK** is defaulted to **NOCHK**. Minimum record length checking is not requested.

- EXITS indicates that a user has provided an exit routine to handle error records. Because no job library has been specified, the exit routine (MYERR) must reside in the link library.

IEBTCRIN Example 2

In this example, IEBTCRIN is invoked via the LINK macro instruction in an Assembler language program. An alternate name has been assigned to each of the DD statements used by IEBTCRIN. The job control for this step must include DD statements with the alternate DD names.

The example follows:

```
         LINK  EP=IEBTCRIN,PARAM=(OPTLIST,DDNAME),VL=1
         CNOP  2,4  (OPTLIST must be on halfword boundary)
OPTLIST  DC    H'0'  (Length must be zero for IEBTCRIN)
         CNOP  2,4  (DDNAME list must be on halfword boundary)
DDNAME   DC    H'82'  (Length of DDNAME list)
         DC    8F'0'
         DC    C'NEWIN    '  (Alternate DDNAME for SYSIN)
         DC    C'NEWPRINT'  (Alternate DDNAME for SYSPRINT)
         DC    2F'0'
         DC    C'NEWUT1   '  (Alternate DDNAME for SYSUT1)
         DC    C'NEWUT2   '  (Alternate DDNAME for SYSUT2)
         DC    C'NEWUT3   '  (Alternate DDNAME for SYSUT3)
```

IEBUPDAT Program—Class C

IEBUPDAT is a data set utility used to incorporate IBM- and user-generated source language modifications into a symbolic library—a partitioned data set containing 80-byte records, such as SYS1.PROCLIB and SYS1.MACLIB. (See "Introduction" for general data set utility information.)

IEBUPDAT can be used to:

- Add, copy, and replace members.
- Add, delete, replace, and renumber the records within an existing member.
- Assign sequence numbers to the records of a new member.

Input and Output

IEBUPDAT uses the following input:

- A partitioned input data set, which contains an old master data set.
- A sequential input data set, which contains the transactions that are to be applied to the old master data set.
- A control data set, which contains utility control statements.

IEBUPDAT produces as output a new master partitioned data set and a sequential data set (SYSPRINT) that reflects either the latest changes applied to the old master data set or to the entire new master data set.

Control

IEBUPDAT is controlled by job control statements and utility control statements.

Job Control Statements

Table 29 shows the job control statements necessary for using IEBUPDAT.

Table 29. IEBUPDAT Job Control Statements

Statement	Use
JOB	Initiates an IEBUPDAT job.
EXEC	Specifies the program name (PGM = IEBUPDAT). Additional information can be specified on the EXEC statement; see "PARM Information on the EXEC Statement" below.
SYSUT1 DD	Defines an input data set.
SYSUT2 DD	Defines an output data set.
SYSPRINT DD	Defines the sequential message data set.
SYSIN DD	Defines the control data set.

The minimum region size that can be specified for IEBUPDAT is 10K + 2b, where b is the largest block size in the job step rounded to the next higher 2K.

The input data set defined by SYSUT1 and the output data set defined by SYSUT2 can contain either blocked or unblocked records with a logical record length of 80 bytes. The output data set can have a blocking factor different from the input data set.

If the DD statements SYSUT1 and SYSUT2 define the same data set, the user can make modifications to the old master without creating a new master.

If enough space cannot be allocated for reblocked output records, the update request is terminated.

PARM Information on the EXEC Statement

IEBUPDAT obtains control information through the EXEC statement and the SYSIN data set. The EXEC statement for this program may contain the parameter:

```
PARM = (input,[inhdr],[intlr])
```

The input value is either NEW or MOD, as follows:

- NEW, which indicates that the input consists of the SYSIN data set.
- MOD, which indicates that the input consists of both the SYSIN and SYSUT1 data sets.

The SYSUT1 data set need not be defined if NEW is specified. If the input value is neither NEW nor MOD, an error is indicated and the operation is terminated. If an input value is not specified, MOD is assumed.

The "inhdr" value specifies the symbolic name of a routine that processes the user header label on the SYSIN data set.

The "intlr" value specifies the symbolic name of a routine that processes the user trailer label on the SYSIN data set.

Utility Control Statements

The utility control statements used to control IEBUPDAT are shown in the order in which they must appear, as follows:

- Header statement, which is used to identify members to be processed.
- NUMBR statement, which is used to identify the sequence number of records to be processed.
- DELET statement, which is used to identify records to be deleted.
- Logical Record, which contains data to be added to or to replace an existing record.
- ALIAS statement, which is used to create or retain aliases in a new master directory.
- ENDUP statement, which indicates the end of the SYSIN input to IEBUPDAT.

The SYSIN data set can contain any number of Header statements and ALIAS statements, each followed by a group of NUMBR, DELET, Logical Record, and ALIAS statements.

Header Statement

A Header statement is used to identify a member to be processed. The statements must be in binary collating sequence by member name.

The format of the Header statement is:

```
./      {ADD   }membername,level,source,list[,ssi]
        {REPL  }
        {CHNGE }
        {REPRO }
```

where:

./
is required and must appear in columns 1 and 2.

ADD
specifies that the named member is to be added in its entirety to the new master. If **ADD** is included, it must begin in column 10.

REPL
specifies that the named member is being entered in its entirety as a replacement for a member in the old master. If **REPL** is included, it must begin in column 10.

CHNGE
specifies that modifications are to be made within the named member. If **CHNGE** is included, it must begin in column 10.

REPRO
specifies that the entire named member is to be copied to the new master. Members are deleted from a library by being omitted from a series of **REPRO** Header statements. If **REPRO** is included, it must begin in column 10.

membername
specifies the name of the member to which the update transactions are to be applied. The *membername* value must begin in column 16.

level
specifies the current run number, a two-digit number from 00 through 99.

source
specifies whether user or IBM modifications are to be made. The values that can be coded are:

0
specifies user modifications.

1
specifies IBM modifications.

list
specifies what the SYSPRINT data set is to contain. The values that can be coded are:

0
specifies that the SYSPRINT data set is to contain only modifications and control statements.

1
specifies that the SYSPRINT data set is to contain the entire updated member and control statements.

ssi
specifies eight hexadecimal characters of new system status index information that is to be placed in the directory of the new master as the first four hexadecimal bytes of user data. If *ssi* is not specified, the user data is copied as it exists in the directory of the old master. System status index information is discussed in detail in *IBM System/360 Operating System: Maintenance*, GC27-6918.

NUMBR Statement

The NUMBR statement contains information to be applied to the member that is named in the Header statement.

The NUMBR statement is used with **CHNGE** Header statements to change the sequence number of one or more logical records within a member, and with **ADD** and **REPL** Header statements to assign sequence numbers to the records within new and replacing members. This statement affects only those sequence numbers that fall in the specified range.

The format of the NUMBR statement is:

```
./        NUMBR seqnumb1,seqnumb2,newseq,increment
```

where:

./
is required and must appear in columns 1 and 2.

NUMBR
specifies that this is a NUMBR statement. **NUMBR** must begin in column 10.

seqnumb1
specifies the sequence number of the first record to be renumbered when used with a **CHNGE** Header statement. This value is ignored when used with **ADD** and **REPL** Header statements. This value must be contained in columns 16 through 23.

,
is a delimiter, which must appear in columns 24, 33, and 42.

seqnumb2
specifies the sequence number of the last record to be renumbered when used with a **CHNGE** Header statement. This value is ignored when used with **ADD** and **REPL** Header statements.

newseq
specifies the first new sequence number. This value must be contained in columns 34 through 41.

increment
specifies the increment value of successive new sequence numbers. This value must be contained in columns 43 through 50.

All of the sequence numbers must be eight-digit alphameric fields.

DELET Statement

The DELET statement contains information to be applied to the member that is named in the Header statement.

The DELET statement is used to delete one or more logical records within a member. It is used only in conjunction with a **CHNGE** Header statement.

The format of the DELET statement is:

```
./        DELET seqnumb1,seqnumb2
```

where:

./
is required and must appear in columns 1 and 2.

DELET
specifies that this is a DELET statement. **DELET** must begin in column 10.

seqnumb1
specifies the sequence number of the first logical record to be deleted. This value must begin in column 16 and not extend beyond column 23.

,
is a delimiter, which must appear in column 24.

seqnumb2
specifies the sequence number of the last logical record to be deleted. This value must begin in column 25 and cannot extend beyond column 32.

Logical Record Statement

The Logical Record statement contains information to be applied to the member that is named in the Header statement. The Logical Record statements contain the data to be added to, or used to replace, existing logical records. They are used in conjunction with **ADD**, **REPL**, or **CHNGE** Header statements.

The format of the Logical Record statement is:

```
data                                                    record-sequence-number
```

where:

data
is the data to be added to or to replace a record. This value must be contained in columns 1 through 72.

record-sequence-number
specifies the sequence number of the record to be processed. This value must be contained in columns 73 through 80.

ALIAS Statement

The ALIAS statement is used to create or retain aliases in a new master directory. The ALIAS statement can be used in conjunction with any of the Header statements. One ALIAS statement must be supplied for each alias.

The format of the ALIAS statement is:

```
./          ALIAS aliasname
```

where:

./
is required and must appear in columns 1 and 2.

ALIAS
specifies that this is an ALIAS statement. **ALIAS** must begin in column 10.

aliasname
specifies an alias. The *aliasname* value must begin in column 16.

If system status index information is provided for the member name, it is also inserted into the alias entry in the directory of the new master.

ENDUP Statement

The ENDUP statement can be used to indicate the end of the SYSIN input to this program; it serves as an end-of-data indication if there is no other indication. The ENDUP statement follows the last group of SYSIN control statements.

The format of the ENDUP statement is:

```
./          ENDUP
```

where:

./
is required and must appear in columns 1 and 2.

ENDUP
specifies that this is an ENDUP statement. **ENDUP** must begin in column 10.

IEBUPDAT Examples

The examples that follow illustrate some of the uses of IEBUPDAT. Table 30 can be used as a quick reference guide to IEBUPDAT examples. The numbers in the "Example" column point to examples that follow.

Table 30. IEBUPDAT Example Directory

Operation	Comments	Example
CATALOG	Job control statements are cataloged in the procedure library.	1
REPLACE	A member of a symbolic library is replaced.	2
DELETE	Records are deleted from a symbolic library.	3
COPY	A member of a symbolic library is copied.	4
CREATE	A three-member library is created.	5

IEBUPDAT Example 1

In this example, a set of job control statements are cataloged in the cataloged procedure library.

The example follows:

```
//UPD1      JOB  09#770,D.P.BROWN
//          EXEC PGM=IEBUPDAT,PARM=NEW
//SYSPRINT  DD   SYSOUT=A
//SYSUT2    DD   (Definition of cataloged procedure library)
//SYSIN     DD   DATA
./        ADD    PROC6,05,0,1
./        NUMBR 00000000,00000000,00000010,00000010
//STEP1     EXEC ---
//DD1       DD   ---
//STEP      EXEC ---
/*
```

PROC6 will be added to the cataloged procedure library defined in SYSUT2. The first record of this procedure will contain the sequence number 00000010 with successive records numbered 00000020, 00000030, etc. The resulting cataloged procedure can be executed with an EXEC statement specifying PROC = PROC6. For additional information on cataloged procedures, see *OS JCL Reference*, GC28-6704.

IEBUPDAT Example 2

In this example, a member in the old master is replaced by a new member.

The example follows:

```
//UPD2      JOB  09#770,D.P.BROWN
//          EXEC PGM=IEBUPDAT,PARM=MOD
//SYSPRINT  DD   SYSOUT=A
//SYSUT1    DD   (Definition of old master data set)
//SYSUT2    DD   (Definition of new master data set)
//SYSIN     DD   *
./        REPL   MBR7,06,0,1
./        NUMBR 00000000,00000000,00000010,00000010
(Logical Record statements)
./        ALIAS ALS7
/*
```

MBR7 (alias ALS7) replaces MBR7 of the library named in SYSUT1. Successive records in the new member will contain the sequence numbers 00000010, 00000020, etc.

IEBUPDAT Example 3

In this example, logical records are deleted from a member of a symbolic library.

The example follows:

```
//UPD3      JOB  09#770,D.P.BROWN
//          EXEC PGM=IEBUPDAT
//SYSPRINT  DD   SYSOUT=A
//SYSUT1    DD   (Definition of old master data set)
//SYSUT2    DD   (Definition of new master data set)
//SYSIN     DD   *
./        CHNGE MBRS,13,0,1
./        DELET 00000050,00000090
./        ALIAS ALS5
/*
```

Logical records 00000050 through 00000090 are deleted from a symbolic library.

IEBUPDAT Example 4

In this example, a member from an old master tape is copied to a new master.

The example follows:

```
//UPD4      JOB  09#770,D.P.BROWN
//          EXEC PGM=IEBUPDAT,PARM=MOD
//SYSPRINT  DD   SYSOUT=A
//SYSUT1    DD   (Definition of old master data set)
//SYSUT2 DD (Definition of new master data set)
//SYSIN     DD   *
./        REPRO MBRZ,12,0,1
/*
```

Member MBRZ, run number 12, is copied from the old master. The SYSPRINT data set is to contain the entire updated member and control statements.

IEBUPDAT Example 5

In this example, a three-member libary is created; the members are named LIBMEMB1, LIBMEMB2, and LIBMEMB3.

The example follows:

```
//UPD5      JOB  09#770,D.P.BROWN
//          EXEC PGM=IEBUPDAT,PARM=NEW
//SYSPRINT  DD   SYSOUT=A
//SYSUT2    DD   (Definition of new master data set)
//SYSIN     DD   *
./        ADD    LIBMEMB1,01,0,1,1234ABCD
(Logical Record statements)
./        ADD    LIBMEMB2,01,0,1,1234EFAB
(Logical Record statements)
./        ADD    LIBMEMB3,01,0,1,1234ABAB
(Logical Record statements)
/*
```

The control statements are discussed below:

- SYSUT2 DD defines a master data set to which the new members are to be written.
- The first **ADD** Header statement specifies that LIBMEMB1 is to be added to the new master data set.
- The first set of Logical Record statements contain the records that are to become LIBMEMB1.
- The second **ADD** Header statement specifies that LIBMEMB2 is to be added to the new master data set.
- The second set of Logical Record statements contain the records that are to become LIBMEMB2.
- The third **ADD** Header statement specifies that LIBMEMB2 is to be added to the new master data set.
- The third set of Logical Record statements contain the records that are to become LIBMEMB3.

IEBUPDTE Program

IEBUPDTE is a data set utility used to incorporate IBM and user-generated source language modifications into sequential or partitioned data sets. (See "Introduction" for general data set utility information.) Exits are provided for user routines that process user header and trailer labels.

IEBUPDTE can be used to:

- Create and update symbolic libraries.
- Incorporate changes to partitioned members or sequential data sets.
- Change the organization of a data set from sequential to partitioned or vice versa.

At the completion or termination of IEBUPDTE, the highest return code encountered within the program is passed to the calling program.

Creating and Updating Symbolic Libraries

IEBUPDTE can be used to create a library of partitioned members consisting of (at the most) 80-byte logical records. 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.

Incorporating Changes

IEBUPDTE can be used to modify an existing partitioned or sequential data set. 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 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 to change a member of a partitioned data set to a sequential data set (the original data set, however, 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 following input:

- An input data set (also called the old master data set), which is to be 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.
- 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 following output:

- An output data set, which is the result of the IEBUPDTE operation. The data set can be either sequential or partitioned. 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.

IEBUPDTE provides a return code to indicate the results of program execution. The return codes and their meanings are:

- 00, which indicates successful completion.
- 04, which indicates that a control statement is coded incorrectly or used erroneously. If either the input or output is sequential, the job step is terminated. If both are partitioned, the program continues processing with the next function to be performed.
- 12, which indicates an unrecoverable error. The job step is terminated.
- 16, which indicates that a label processing code of 16 was received from a user's label processing routine. The job step is terminated.

Control

IEBUPDTE is controlled by job control statements and utility control statements. The job control statements are required to execute or invoke 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 31 shows the job control statements necessary for using IEBUPDTE.

Table 31. IEBUPDTE Job Control Statements

Statement	Use
JOB	Initiates 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; see "PARM Information on the EXEC Statement" below.
SYSPRINT DD	Defines a sequential message data set. 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. Or, it can define a partitioned data set on a direct access volume. This DD statement is not required if the input consists solely of the control data set.
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 on a direct access device.
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.

The minimum region size that can be specified for IEBUPDTE is 14K + 2b, where b is the largest block size in the job step rounded to the next higher 2K.

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.

If an **ADD** operation is specified with PARM = NEW on the EXEC card, the SYSUT1 DD statement need not be coded. See "PARM Information on the EXEC Statement" below. Refer to "Function Statement" below for a discussion of the **ADD** operation.

If an **UPDATE** operation is specified, the SYSUT2 DD statement should not be coded. Refer to "Function Statement" for a discussion of the **UPDATE** operation.

If the SYSUT1 DD statement defines a sequential data set, 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 both the SYSUT1 and SYSUT2 DD statements specify standard user labels (SUL), IEBUPDTE copies user labels from SYSUT1 to SYSUT2. See "Appendix E: Processing User Labels" for a discussion of the available options for user label processing.

If the SYSUT1 and SYSUT2 DD statements define the same partitioned data set, 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 member(s) will point to the newly written member(s). The member names themselves should not appear on the DD statements; they should be referenced only through IEBUPDTE control statements.

Restrictions

- The SYSPRINT DD statement is required for each use of IEBUPDTE.
- The output data set can have a blocking factor that is different from the input data set; however, if insufficient space is allocated for reblocked records, the update request is terminated.
- When adding a member to an existing partitioned data set 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.
- Space must be allocated for an output data set (SYSUT2 DD statement) that is to reside on a direct access device, unless the data set is an existing data set, in which case space should not be allocated.
- The SYSUT2 DD statement must not specify a DUMMY data set.

- If the SYSUT1 and SYSUT2 DD statements define the same sequential data set (direct access only), only those operations that add data to the end of the existing data set can be made. In these cases:
  1. The PARM parameter of the EXEC statement must imply or specify MOD. (See "PARM Information on the EXEC Statement" below.)
  2. The DISP parameter of the SYSUT1 DD statement must specify OLD.
  3. The DISP parameter of the SYSUT2 DD statement must specify MOD.
- The SYSIN DD statement is required for each use of IEBUPDTE.
- The message data set has a logical record length of 121 bytes, and consists of fixed length, blocked or unblocked records with an ASA control character in the first byte of each record. The input and output data sets have a logical record length of 80 bytes or less, and consist of fixed blocked (RECFM = FB) or unblocked records. The control data set contains 80-byte, blocked or unblocked records.

PARM Information on the EXEC Statement

Additional information can be coded in the PARM parameter of the EXEC statement, as follows:

```
PARM = ( {NEW }.[inhdr],[intlr])
       {MOD }
```

Following are the PARM values:

- NEW, which specifies that the input consists solely of the control data set. The input data set is not defined if NEW is specified.
- MOD, which specifies that the input consists of both the control data set and the input data set. If neither NEW nor MOD is coded, MOD is assumed.
- "inhdr," which specifies the symbolic name of a routine that processes the user header label on the volume containing the control data set.
- "intlr," which specifies the symbolic name of a routine that processes the user trailer label on the volume containing the control data set.

For a detailed discussion of the processing of user labels as data set descriptors, refer to "Appendix E: Processing User Labels."

Utility Control Statements

The utility control statements used to control IEBUPDTE are:

- Function statement, which is used to initiate an IEBUPDTE operation.
- Detail statement, which is used with the Function statement for special applications.
- Data statement, which is a logical record of data to be used as a new or replacement record in the output data set
- LABEL statement, which is used to indicate that the following data statements are to be treated as user labels.
- ALIAS statement, which is used to assign aliases.
- ENDUP statement, which is used to terminate IEBUPDTE.

Function Statement

The Function statement is used to initiate 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.

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.

The format of the Function statement is:

```
./ [name]  {ADD     }[LIST = ALL]
           {REPL    }
           {CHANGE  }[,SEQFLD = ddl]
           {REPRO   }
                     [,NEW = {PO }]
                             {PS }
                     [,MEMBER = cccccccc]
                     [,COLUMN = dd]
                     [,UPDATE = INPLACE]
                     [,INHDR = cccccccc]
                     [,INTLR = cccccccc]
                     [,OUTHDR = cccccccc]
                     [,OUTTLR = cccccccc]
                     [,TOTAL = (routinename,size)]
                     [,NAME = cccccccc]
                     [,LEVEL = hh]
                     [,SOURCE = x]
                     [,SSI = hhhhhhhh]
```

where:

./
is required and must appear in columns 1 and 2.

name
specifies an optional name which begins in column 3 and extends no further than column 10.

ADD
specifies that a member or a data set is to be added to an old master data set. If a member is to be added and the member name already exists in the old master data set, processing is terminated. 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. At least one blank must precede and follow **ADD**.

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. At least one blank must precede and follow **REPL**.

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. At least one blank must precede and follow **CHANGE**.

REPRO
specifies that a member or a data set is to be copied in its entirety to a new master data set. At least one blank must precede and follow **REPRO**.

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. 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.

SEQFLD = *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 LRECL +1. If **SEQFLD** is omitted, 738 is assumed, that is, an eight-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. In any case, sequence numbers on incoming Data statements and existing logical records must be padded to the left with enough zeros to fill the length of the sequence field.

NEW =
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. Refer to Table 31 for the use of **NEW** with **NAME** and **MEMBER**. 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.

PS
specifies that the old master data set is a partitioned data set, and that a member of that data set is to be converted into a sequential data set.

MEMBER = *cccccccc*
specifies a name to be assigned to the member placed in the partitioned data set defined by the SYSUT2 DD statement. **MEMBER** is used only when SYSUT1 defines a sequential data set, SYSUT2 defines a partitioned data set, and NEW = PO is specified. Refer to Table 32 for the use of **MEMBER** with **NEW**.

COLUMN = *dd*
specifies, in decimal, the starting column of a data field within a logical record image. The field extends to the end of the image. Within an existing logical record, the data in the defined field is replaced by data from a subsequent Data statement. **COLUMN** is valid only when **CHANGE** is coded.

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** is valid only when coded with **CHANGE**.

INHDR = *cccccccc*
specifies the symbolic name of the user routine that handles any user input (SYSUT1) header labels. When used with **UPDATE**, this routine assumes a special function. This parameter is valid only when a sequential data set is being processed. See "LABEL Statement" for a discussion of this function.

INTLR = *cccccccc*
specifies the symbolic 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** is coded.

OUTHDR = *cccccccc*
specifies the symbolic 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** is coded.

OUTTLR = *cccccccc*
specifies the symbolic 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** is coded.

TOTAL =
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 the user's totaling routine.

size
specifies the number of bytes required for the user's 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. Refer to "Appendix A: Exit Routine Linkage" for a discussion of linkage conventions for user routines.

NAME = *cccccccc*
indicates the name of the member placed into the partitioned data set. The member name need not be specified in the DD statement itself. **NAME** must be provided to identify each input member. Refer to Table 32 for the use of **NAME** with **NEW**. This parameter is valid only when a member of a partitioned data set 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. This parameter is valid only when a member of a partitioned data set is being processed.

SOURCE = *x*
specifies user 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 parameter is valid only when a member of a partitioned data set is being processed.

SSI = *hhhhhhhh*
specifies eight hexadecimal characters of system status information (SSI) to be placed in the directory of the new master data set as four packed hexadecimal bytes of user data. This parameter is valid only when a member of a partitioned data set is being processed. **SSI** overrides any **LEVEL** or **SOURCE** data given on the same Function statement.

Members can be 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 deleted by being omitted from a series of job steps which copy only the desired data sets to a new volume. If the NEW subparameter is coded in the EXEC statement, only the **ADD** Function statement is permitted.

Figure 39 shows how the system status information (SSI = 0A3C123B) is packed. Refer to *IBM System/360 Operating System: Maintenance*, GC27-6918, for a detailed discussion of the format of system status information.

Change level byte 1		Flag byte byte 2		Serial number byte 3		byte 4	
0	A	3	C	1	2	3	B

Figure 39. Format of System Status Information

When **UPDATE** is specified:

- The SYSUT2 DD statement need not be coded.
- The PARM parameter of the EXEC statement must imply or specify MOD.
- The NUMBER 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** parameter are permitted per job step.
- No functions other than replacement, renumbering, and header label modification (via the LABEL statement) can be specified.
- Only replaced records are listed unless the entire data set is renumbered.
- System status information cannot be changed.

Within an existing logical record, the data in the field defined by **COLUMN** 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 32 shows the use of **NEW**, **MEMBER**, and **NAME** parameters for different input and output data set organizations.

Table 32. NEW, MEMBER, and NAME Parameters

Input Data Set Organization	*Output Data Set Organization*	*Parameter Combinations*
Partitioned	Partitioned	With an ADD Function statement, use NAME to specify the name of the member to be placed in the partitioned data set defined by the SYSUT2 DD statement. If an additional name is required, an ALIAS statement can also be used. With a CHANGE, REPL, or REPRO Function statement, use NAME to specify the name of the member within the partitioned data set defined by the SYSUT1 DD statement. If a different or additional name is desired for the member in the partitioned data set defined by the SYSUT2 DD statement, use an ALIAS statement also.
None	Partitioned (New)	With each ADD Function statement, use NAME to assign a name for each member to be placed in the partitioned data set.
Partitioned	Sequential	With a Function statement, use NAME to specify the name of the member in the partitioned 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 assigned to the output master data set are specified in the SYSUT2 DD statement.)
Sequential	Partitioned	With a Function statement, use MEMBER to assign a name to the member to be placed in the partitioned data set defined by the SYSUT2 DD statement. Use NEW = PO to specify the change in organization from sequential to partitioned.

For a detailed discussion of the processing of user labels as data set descriptors, and for a discussion of user-label totaling, see "Appendix E: Processing User Labels."

Detail Statement

A Detail statement is used with a Function statement for certain applications, such as deleting or renumbering selected logical records.

Note: Logical records cannot be deleted in part; that is, a **COLUMN** 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 format of a Detail statement is:

```
./[name]  {NUMBER }[SEQ1 = {cccccccc }]
          {DELETE }        {ALL      }
                   [,SEQ2 = cccccccc]
                   [,NEW1 = cccccccc]
                   [,INCR = cccccccc]
                   [,INSERT = YES]
```

where:

./
is required and must appear in columns 1 and 2.

name
specifies an optional name which begins in column 3 and extends no further than column 10.

NUMBER
specifies, when coded with a **CHANGE** Function statement, that the sequence number of one or more logical records is to be 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. If **NUMBER** is coded, it must be preceded and followed by at least one blank.

DELETE
specifies, when coded with a **CHANGE** Function statement, that one or more logical records is to be deleted from a member or data set. If **DELETE** is coded, it must be preceded and followed by at least one blank.

SEQ1 =
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 insert 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: (1) **SEQ2** need not be coded and (2) 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. 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). This parameter is valid only in a **NUMBER** Detail statement.

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. This parameter is valid only in a **NUMBER** Detail statement.

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.

When **INSERT** is coded:

- The **SEQ1** parameter specifies the existing logical record after which the insertion is to be made. The **SEQ2** parameter need not be coded; SEQ1 = ALL cannot be coded.
- The **NEW1** parameter assigns a sequence number to the first logical record to be inserted.
- 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**. 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 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 terminated when a Function statement, a Detail statement, an end-of-file indication, or a Data statement containing a sequence number is encountered.

The **SEQ1**, **SEQ2**, **NEW1**, and **INCR** parameters (with the exception of SEQ = ALL) specify eight-character (maximum) decimal numbers. Only the significant numbers of a value need be coded; for example, SEQ1 = 00000010 can be shortened to SEQ1 = 10.

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 to be placed in the output master data set.

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 with 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 terminated. 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.

Sequence numbers in Data statements must be in the same relative position as sequence numbers in existing logical records. (Sequence numbers 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.)

The sequence number of the Data statement must always be equal to or greater than the sequence number of the next old master record.

LABEL Statement

The LABEL statement indicates that the following Data statements are to be treated as user labels. These new user labels are placed on the output data set. The next Function statement indicates to IEBUPDTE that the last label data statement of the group has been read.

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 four 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. This parameter list is described fully in "Appendix A: Exit Routine Linkage." 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 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. 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 **UPDATE** is specified in a Function 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. See "Appendix E: Processing User Labels" for a complete discussion of the linkage between utility programs and user label-processing routines. The return codes when **UPDATE** is used differ slightly from the standard codes discussed in "Appendix E: Processing User Labels," as follows:

- 0, which specifies that the system resumes normal processing; any additional user labels are ignored.
- 4, which specifies that 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's routine. If there are no more user labels, the system resumes normal processing.
- 8, which specifies that the system writes the user labels from the label buffer area and resumes normal processing.
- 12, which specifies that 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.

If the user wants to examine the replaced labels from the old master data set, he must:

1. Specify an update of the old master by coding the **UPDATE** 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 the above 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 the user specifies, 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.

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

ALIAS Statement

The ALIAS statement is used to create or retain an alias in an output (partitioned) master directory. The ALIAS statement can be used with any of the Function statements. Multiple aliases can be assigned to each member.

The format of the ALIAS statement is:

```
./[name]        ALIAS NAME = cccccccc
```

where:

./
is required and must appear in columns 1 and 2.

name
specifies an optional name which begins in column 3 and extends no further than column 10.

NAME = cccccccc
specifies a one- to eight-character alias.

ALIAS must be preceded and followed by at least one blank. If multiple ALIAS statements are used, they must follow the data records.

ENDUP Statement

An ENDUP statement can be used to indicate the end of SYSIN input to this job step. It serves as an end-of-data indication if there is no other indication. The ENDUP statement follows the last group of SYSIN control statements.

The format of the ENDUP statement is:

```
./[name]        ENDUP
```

where:

./
is required and must appear in columns 1 and 2.

name
specifies an optional name which begins in column 3 and extends no further than column 10.

ENDUP must be preceded and followed by at least one blank. The ENDUP statement must follow the last label data statement if IEBUPDTE is used to create output trailer labels.

IEBUPDTE Examples

The following examples illustrate some of the uses of IEBUPDTE. Table 33 can be used as a quick reference guide to IEBUPDTE examples. The numbers in the "Example" column point to examples that follow.

Table 33. IEBUPDTE Example Directory

Operation	Data set Organization	Device	Comments	Example
CATALOG a procedure	Partitioned	2314 Disks	SYSUT1 and SYSUT2 DD statements define the same data set. Procedure to be cataloged is in the control data set.	1
CREATE a partitioned library	Partitioned	2314 Disk	Input data is in the control data set. Output partitioned data set is to contain three members.	2
CREATE a partitioned data set	Partitioned	2314 Disk	Input from control data set and from existing partitioned data set. Output partitioned data set is to contain four members.	3
UDPATE INPLACE and renumber	Partitioned	2314 Disk	Input data set is considered to be the output data set as well; therefore, no SYSUT2 DD statement is required.	4
CREATE and DELETE	Partitioned, Sequential	3330 Disk 9-track tape	Sequential master is to be created from partitioned tape input. Selected records are to be deleted. Blocked output.	5
CREATE, DELETE, and UPDATE	Sequential, Partitioned	9-track tape, 2314 Disk	Partitioned data set is to be created from sequential input. Records are to be deleted and updated. Sequence numbers in columns other than 73 through 80. One member is to be placed in the output data set.	6
INSERT	Partitioned	2314 Disks	Block of logical records is to be inserted into an existing member. SYSUT1 and SYSUT2 DD statements define the same data set.	7
INSERT	Partitioned	2314 Disks	Two blocks of logical records are to be inserted into an existing member. SYSUT1 and SYSUT2 DD statements define the same data set.	8
CREATE	Sequential	Card Reader, 2314 Disk	Sequential data set with user labels is to be created from card input.	9
COPY	Sequential	2314 Disks	Sequential data set is to be copied from one direct access volume to another; user labels can be processed by exit routines.	10

IEBUPDTE Example 1

In this example, a procedure is to be placed in the cataloged procedure library, SYS1.PROCLIB. The example assumes that the new procedure (ERASE) can be accommodated within the space originally allocated to the procedure library; therefore, the update operation is performed directly to the old master.

The example follows:

```
//UPDATE   JOB  09#660,SMITH
//         EXEC PGM=IEBUPDTE,PARM=MOD
//SYSPRINT DD   SYSOUT=A
//SYSUT1   DD   DSNAME=SYS1.PROCLIB,DISP=(OLD,KEEP)
//SYSUT2   DD   DSNAME=SYS1.PROCLIB,DISP=(OLD,KEEP)
//SYSIN    DD   DATA
./        ADD   LIST=ALL,NAME=ERASE,LEVEL=01,SOURCE=0
./     NUMBER   NEW1=10,INCR=10
//ERASE    EXEC PGM=IEHPROGM
//DD1      DD   UNIT=190,DISP=(OLD,KEEP),VOLUME=SER=111111
//SYSPRINT DD   SYSOUT=A
./     ENDUP
/*
```

The control statements are discussed below:

- SYSUT1 and SYSUT2 DD define the SYS1.PROCLIB data set, which is assumed to be cataloged.
- SYSIN DD defines the control data set. 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 is to be listed in the message data set.
- The **NUMBER** Detail statement indicates that the new procedure is assigned sequence numbers. The first record of the procedure is assigned sequence number 10; the remaining two records are assigned sequence numbers 20 and 30.,

The ERASE EXEC, DD1, and SYSPRINT DD statements are placed in the cataloged procedure library, SYS1.PROCLIB, as a result of this job.

Note: The resulting procedure can be executed with an EXEC statement specifying PROC = ERASE. For additional information on cataloged procedures, see *OS JCL Reference*, GC28-6704.

IEBUPDTE Example 2

In this example, a three-member, partitioned library is to be created. The input data is contained solely in the control data set.

The example follows:

```
//UPDATE   JOB  09#770,SMITH
//         EXEC PGM=IEBUPDTE,PARM=NEW
//SYSPRINT DD   SYSOUT=A
//SYSUT2   DD   DSNAME=OUTLIB,UNIT=2314,DISP=(NEW,KEEP),
// VOLUME=SER=111112,SPACE=(TRK,(100,,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 discussed below:

- 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. 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 is to be listed in the message data set.

- The Data statements contain the data to be placed in the output partitioned data set.
- ENDUP signals the end of control data set input.

Note: Because sequence numbers (other than blank numbers) are included within the data statements, no **NUMBER** Detail statements are included in the example.

IEBUPDTE Example 3

In this example, a four-member, partitioned data set (NEWMCLIB) is to be created. The data set is to contain:

- Two members (ATTACH and DETACH) copied from an existing partitioned data set (SYS1.MACLIB).
- One replacement member (BLDL) for an existing member of the input partitioned data set.
- A new member (EXIT) The new member (EXIT) is contained in the control data set.

The example follows:

```
//UPDATE   JOB  09#770,SMITH
//         EXEC PGM=IEBUPDTE,PARM=MOD
//SYSPRINT DD   SYSOUT=A
//SYSUT1   DD   DSNAME=SYS1.MACLIB,DISP=SHR,UNIT=2314
//SYSUT2   DD   DSNAME=NEWMCLIB,VOLUME=SER=111112,UNIT=2314,
// DISP=(NEW,KEEP),SPACE=(TRK,(100,,10)),
// DCB=(RECFM=F,LRECL=80,BLKSIZE=80)
//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 cards for EXIT member)

```
./        REPL   NAME=BLDL,LEVEL=01,SOURCE=1,LIST=ALL
./      NUMBER   NEW1=10,INCR=100
```

(Data cards to replace BLDL member)

```
./       ENDUP
/*
```

The control statements are discussed below:

- SYSUT1 DD defines the input partitioned data set SYS1.MACLIB, which is assumed to be cataloged.
- SYSUT2 DD defines the output partitioned data set OUTLIB. Enough space is allocated to allow for subsequent modifications without creating a new master data set.
- SYSIN DD defines the control data set.
- The **REPRO** Function statements identify the existing input members to be copied onto the output data set. These members are also listed in the message data set.
- The **ADD** Function statement indicates that records (subsequent Data statements) are to be placed as a member in the output partitioned data set. 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 records are incremented by 100.
- The **REPL** Function statement identifies a new member used as a replacement for an existing member. The subsequent **NUMBER** Detail statement assigns sequence numbers to the records in the new member.
- ENDUP signals the end of SYSIN data.

Note: The three named input members (ATTACH, DETACH, and BLDL) do not have to be specified in the order of their collating sequence in the old master.

IEBUPDTE Example 4

In this example, a member (MODMEMB) is to be updated within the space it actually occupies. Two existing logical records are to be replaced, and the entire member is to be renumbered.

The example follows:

```
//UPDATE   JOB  09#770,SMITH
//         EXEC PGM=IEBUPDTE,PARM=MOD
//SYSPRINT DD   SYSOUT=A
//SYSUT1   DD   DSNAME=PDS,UNIT=2314,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 discussed below:

- SYSUT1 DD defines the data set that is to be updated in place. (Note that the member name need not be specified in the DD statement.)
- SYSIN DD defines the control data set.
- The **CHANGE** Function statement indicates the name of the member to be updated and specifies the UPDATE = INPLACE operation. The entire member is to be listed in the message data set.
- 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 for successive sequence numbers.
- The Data statements replace existing logical records having sequence numbers of 20 and 35.

IEBUPDTE Example 5

In this example, a sequential master data set is to be created from partitioned input and selected logical records are to be deleted.

The example follows:

```
//UPDATE   JOB  09#770,SMITH
//         EXEC PGM=IEBUPDTE,PARM=MOD
//SYSPRINT DD   SYSOUT=A
//SYSUT1   DD   DSNAME=PARTDS,UNIT=3330,DISP=(SHR,KEEP),
// VOLUME=SER=111112
//SYSUT2   DD   DSNAME=SEQDS,UNIT=2400,LABEL=(2,SL),
// DISP=(,KEEP),VOLUME=SER=001234
// 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 discussed below:

- SYSUT1 DD defines the input partitioned data set PARTDS. Because no DCB parameters are specified in the SYSUT1 DD statement, the input data set is assumed to consist of 80-byte, unblocked records. A warning message is issued to inform the user that this assumption was made.
- SYSUT2 DD defines the output sequential data set. The data set is to be written as the second data set on a 9-track tape volume. The data set is written at 800 bits per inch density.
- SYSIN DD defines the control data set.
- CHANGE identifies the input member (OLDMEMB1) and indicates that the output is to be 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.

- The second Data statement is inserted in the proper sequence in the output data set.

Note: Only one member can be used as input when converting to sequential organization.

IEBUPDTE Example 6

In this example, a member of a partitioned data set is to be created from sequential input and existing logical records are to be updated.

The example follows:

```
//UPDATE   JOB  09#770,SMITH
//         EXEC PGM=IEBUPDTE,PARM=MOD
//SYSPRINT DD   SYSOUT=A
//SYSUT1   DD   DSNAME=OLDSEQDS,UNIT=2400,LABEL=(,SL),
// DISP=(OLD,KEEP),VOLUME=SER=001234
//SYSUT2   DD   DSNAME=NEWPART,UNIT=2314,DISP=(,KEEP),
// VOLUME=SER=111112,SPACE=(TRK,(20,5,5)),
// DCB=(RECFM=F,LRECL=80,BLKSIZE=80)
//SYSIN    DD   *
./     CHANGE   NEW=PO,MEMBER=PARMEM1,LEVEL=01,                    ¢C
./              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 discussed below:

- SYSUT1 DD defines the input sequential data set (OLDSEQDS). The data set was originally written at a density of 800 bits per inch on a 9-track tape volume.
- SYSUT2 DD defines the output partitioned data set. Enough space is allocated to provide for members that might be added in subsequent job steps.
- SYSIN DD defines the control data set.
- The **CHANGE** Function statement identifies the output member and indicates that a conversion from sequential input to partitioned output is to be made. The **SEQFLD** parameter indicates that a five-byte sequence number is located in columns 60 through 64 of each Data statement. The **COLUMN** parameter specifies the starting column of a field (within subsequent data statements) from which replacement information is obtained.
- 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 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 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 MEMB1 to the output member PARMEM1.

IEBUPDTE Example 7

In this example, a block of three logical records is to be inserted into an existing member, and the updated member is to be placed in the existing partitioned data set.

Figure 40 shows existing sequence numbers, new sequence numbers, and Data statements to be inserted.

Sequence Numbers and Data Statements to be 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

Figure 40. Sequence Numbers and Data Statements to Be Inserted

The example follows:

```
//UPDATE   JOB  09#770,SMITH
//         EXEC PGM=IEBUPDTE,PARM=MOD
//SYSPRINT DD   SYSOUT=A
//SYSUT1   DD   DSNAME=PDS,UNIT=2314,DISP=(OLD,KEEP),
// VOLUME=SER=111112
//SYSUT2   DD   DSNAME=PDS,UNIT=2314,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 discussed below:

- SYSUT1 and SYSUT2 DD define the partitioned data set PDS.
- SYSIN DD defines the control data set.
- The **CHANGE** Function statement identifies the input member RENUM. The entire member is to be 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, 30, etc. 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.
4. The remaining logical records in the member are renumbered.

IEBUPDTE Example 8

In this example, two blocks (three logical records per block) are to be inserted into an existing member, and the member is to be placed in the existing partitioned data set. A portion of the output member is to be renumbered.

Figure 41 shows existing sequence numbers, new sequence numbers, and Data statements to be inserted.

Sequence Numbers and Data Statements to be 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
Data statement 4	50
Data statement 5	55
Data statement 6	60
35	65
Data statement 7	70
40	75
50	80
150	150
155	155

Figure 41. Sequence Numbers and Seven Data Statements to Be Inserted

The example follows:

```
//UPDATE   JOB  09#770,SMITH
//         EXEC PGM=IEBUPDTE,PARM=MOD
//SYSPRINT DD   SYSOUT=A
//SYSUT1   DD   DSNAME=PDS,UNIT=2314,DISP=(OLD,KEEP),
// VOLUME=SER=111112
//SYSUT2   DD   DSNAME=PDS,UNIT=2314,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)
./     NUMBER   SEQ1=30,INSERT=YES
(Data statement 4)
(Data statement 5)
(Data statement 6)
(Data statement 7, sequence number 00000038)
/*
```

The control statements are discussed below:

- SYSUT1 and SYSUT2 DD define the partitioned data set PDS.
- SYSIN DD defines the control data set.
- The **CHANGE** Function statement identifies the input member RENUM. The entire member is to be 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 is a logical record to be inserted in the output member.

The existing logical records in this example have sequence numbers 10, 15, 20, 25, 30, 35, 40, 45, 50, 150, 155, 160, 165, etc. The insert and renumbering operations are performed 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.
4. Data statement 4 is assigned sequence number 50 and inserted. (The SEQ1 = 30 specification in the second NUMBER statement places this data statement after existing logical record 30, which has become logical record 45.)
5. Data statements 5 and 6 are assigned sequence numbers 55 and 60 and are inserted after Data statement 4.
6. Existing logical record 35 is assigned sequence number 65.
7. Data statement 7 is assigned sequence number 70 and is inserted.
8. The remaining logical records in the member are renumbered until logical record 150 is encountered. Because this record has a sequence number higher than the next number to be assigned, the renumbering operation is terminated.

IEBUPDTE Example 9

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.

The example follows:

```
//LABEL    JOB  ,MSGLEVEL=1
//CREATION EXEC PGM=IEBUPDTE,PARM=NEW
//SYSPRINT DD   SYSOUT=A
//SYSUT2   DD   DSNAME=LABEL,VOLUME=SER=123456,
// DISP=(NEW,KEEP),LABEL=(,SUL),
// SPACE=(TRK,(15,3)),UNIT=2314
//SYSIN    DD   *
./       LABEL
```

(First header label)

(Last header label)

```
./         ADD   LIST=ALL,OUTHDR=ROUTINE1,OUTTLR=ROUTINE2
```

(First input data record)

(Last input data record)

```
./       LABEL
```

(First trailer label)
(Last trailer label)

```
./       ENDUP
/*
```

The control statements are discussed below:

- SYSUT2 DD defines and allocates space for the output sequential data set, which resides on a 2314 volume.
- SYSIN DD defines the control data set. (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 to be placed in the output data set. The newly created data set is to be listed in the message data set. User output header and output trailer routines are to be given control prior to 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.

IEBUPDTE Example 10

In this example, IEBUPDTE is used to copy a sequential data set from one direct-access volume to another. User labels are processed by user exit routines.

The example follows:

```
//LABELS   JOB  ,MSGLEVEL=1
//         EXEC PGM=IEBUPDTE,PARM=(MOD,,MMMMMM)
//SYSPRINT DD   SYSOUT=A
//SYSUT1   DD   DSNAME=OLDMAST,DISP=OLD,LABEL=(,SUL),
// VOLUME=SER=111111,UNIT=2314
//SYSUT2   DD   DSNAME=NEWMAST,DISP=(NEW,KEEP),LABEL=(,SUL),
// UNIT=2314,VOLUME=SER=XB182.
// SPACE=(TRK,(10,10))
//SYSIN    DD   DSNAME=INPUT,DISP=OLD,LABEL=(,SUL),
// VOLUME=SER=222222,UNIT=2314
/*
```

(Input data set)

```
./      REPRO   LIST=ALL,INHDR=SSSSSS,INTLR=TTTTTT,          ¢C
./              OUTHDR=XXXXXX,OUTTLR=YYYYYY
./      ENDUP
```

The control statements are discussed below:

- SYSUT1 DD defines the input sequential data set, which resides on a 2314 volume.
- SYSUT2 DD defines the output sequential data set, which will reside on a 2314 volume.
- SYSIN DD defines the control data set.
- The REPRO Function statement indicates that the existing input sequential data set is to be copied to the output data set. This output data set is to be 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.

IEHATLAS Program

IEHATLAS is a system utility used when a defective track is indicated by a data check or missing address marker condition. (See "Introduction" for general system utility information.)

IEHATLAS can be used to locate and assign an alternate track to replace the defective track. Usable data records on the defective track are retrieved and transferred to the alternate track. The bad record from the defective track is then replaced on the alternate by a correct copy provided by the user.

In a simple application, IEHATLAS is used as a separate job after an abnormal termination of a problem program. Input data necessary for execution of IEHATLAS—the address of the defective track and replacement records—may be obtained from the dump and from backup data.

A more complex use of IEHATLAS may involve the preparation of a user's SYNAD routine, which reconstructs the necessary input data and invokes IEHATLAS dynamically.

When IEHATLAS is invoked, it attempts to write on the defective track. If the subsequent read-back check indicates that the attempt was successful, a message is issued on the SYSOUT device. If not, a supervisor call routine (SVC 86) is entered automatically.

The SVC routine locates and assigns an alternate track. (If a defective track already has an alternate and an error occurs on that alternate, the SVC routine assigns the next available alternate. All of the valid data records on the defective track are retrieved and transferred to the alternate track. The input record is written on the alternate track in the correct position to recover from the previous error.

When a READ error occurs and a complete recovery is desired, IEHDASDR can be used to produce a listing of error data on a track. Using this data, the input data record for IEHATLAS can be created. The *replace* function can then be performed by executing IEHATLAS.

The direct access device types supported by IEHATLAS are 2302, 2311, 2314, 2319, 2305, and 3330.

Input and Output

IEHATLAS uses the following input: (1) a description of the defective track, specifying the bin (or cell), cylinder, track, record, key, and data length (in hexadecimal notation), (2) an indication if WRITE Special is needed, and (3) a valid copy (in hexadecimal notation) of the bad record.

IEHATLAS produces as output: (1) a message, issued on the SYSOUT device, containing the user's control information, the input record, and diagnostics, (2) the input record, written on either the original (defective) track or on an alternate track containing the usable data taken from the defective track, and (3) the return parameter list (specifying a maximum of three error record numbers in hexadecimal when an unrecoverable error occurs).

Control

IEHATLAS is controlled by job control statements and utility control statements. The job control statements are used to execute or invoke IEHATLAS and to define the data sets used and produced by IEHATLAS.

A utility control statement is used to specify whether the bad record is a member of the volume table of contents or a member of some other data set. It is also used to indicate whether or not the WRITE Special command is to be used.

Job Control Statements

Table 34 shows the job control statements necessary for using IEHATLAS.

For detailed information on the minimum region size that can be specified for IEHATLAS, see *OS Storage Estimates*, GC28-6551.

Restrictions

- The block size for the SYSPRINT data set must be a multiple of 121. The block size for the SYSIN data set must be a multiple of 80. Any blocking factor can be specified.
- DISP = SHR must not be coded on the SYSUT1 DD statement.

Table 34. IEHATLAS Job Control Statements

Statement	Use
JOB	Initiates the job.
EXEC	Specifies the program name (PGM = IEHATLAS) or, if the job control statements reside in a procedure library, the procedure name.
SYSABEND DD	Defines a dump data set. It must include appropriate parameters for a basic sequential (BSAM) data set. The data set can be written to a system output device, a tape volume, or a direct access volume.
SYSPRINT DD	Defines a sequential data set that contains the output messages issued by IEHATLAS.
SYSUT1 DD	Defines the data set that contains the bad record.
SYSIN DD	Defines the control data set, which contains the utility control statement and a copy of the bad record.

Utility Control Statement

The TRACK or VTOC control statement is used to control IEHATLAS.

TRACK or VTOC Statement

The TRACK or VTOC statement is used to identify the defective record.

The format of the TRACK or VTOC statement is:

```
{TRACK = bbbbccccchhhhrrkkdddd[S] }
{VTOC = bbbbccccchhhhrrkkdddd     }
```

where:

TRACK =
specifies that an alternate track is to be assigned for a track that does not contain VTOC records.

VTOC =
specifies that an alternate track is to be assigned for a track that contains VTOC records.

bbbb
is the bin (or cell) number when the device specified in the SYSUT1 DD Statement is a 2321 data cell; if the device is other than a 2321 data cell, the number must be padded with zeros.

cccc
is the number of the cylinder in which the defective track was found.

hhhh
is the defective track number.

rrkk
is the record number and key length for the bad record.

dddd
is the data length of the bad record. (When a WRITE Special command is used, *dddd* is the length of the record segment.

S
is an optional byte of EBCDIC information that specifies that the WRITE Special command is to be used (when the last record on the track overflows and must be completed elsewhere).

Care should be taken to ensure that the input record data length does not exceed the track size. This is especially important when the WRITE Special command is specified because the error may not be recognized immediately by the system.

The TRACK or VTOC statement must not begin in column 1.

Input data (consisting of the hexadecimal replacement record) begins in column 1 immediately following the utility control data. Input data may continue through column 80. As many cards as necessary may be used to contain the replacement record. All columns (1 through 80) are used on the additional cards.

IEHATLAS is designed to replace an error record with a copy of that record. It cannot be used to replace a record with another of a different key and/or data length.

An end-of-file record cannot be changed; therefore, input for key and/or data fields are ignored.

IEHATLAS Examples

The following examples illustrate some of the uses of IEHATLAS. Table 35 can be used as a quick reference guide to IEHATLAS examples. The numbers in the "Example" column point to examples that follow.

Table 35. IEHATLAS Example Directory

Operation	Comments	Example
Get Alternate Tracks	Write Special is included because of a track overflow condition.	1
Get Alternate Track	Alternate track assigned for a bad end-of-file record.	2
Get Alternate Track	Alternate track assigned for a bad VTOC record.	3
Get Alternate Track	Replace defective record zero.	4

IEHATLAS Example 1

In this example, the data set defined by SYSUT1 contains the bad record. An alternate track on the specified unit and volume will be assigned to replace the defective track.

The example follows:

```
//JOBATLAS JOB  06#990,SMITH,MSGLEVEL=1
//STEP     EXEC PGM=IEHATLAS
//SYSPRINT DD   SYSOUT=A
//SYSUT1   DD   DSNAME=NEWSET,UNIT=3330,VOLUME=SER=333011,
// DISP=OLD
//SYSIN    DD   *
 TRACK=00000002000422020006S
F3F1C2C2F0F00000
/*
```

The control statements are discussed below:

- SYSPRINT DD defines the device to which the output messages can be written.
- SYSUT1 DD defines the data set that contains the bad record.
- SYSIN DD defines the control data set, which follows in the input stream.
- TRACK specifies the bin, cylinder, and track number for the defective track; and the record number, key length, and data length of the bad record. In this example, the device is a 3330, so the input record has a bin number of zero; the record is to be placed on cylinder 2, track 4, record 22; and it has a key length of 2 with a logical record length of 8. The WRITE Special (S) character is used because there is a track overflow condition.

The input record is a typical hexadecimal record as defined by the TRACK statement. The input record contains eight bytes (data length = 6, key length = 2).

IEHATLAS Example 2

In this example, an alternate track is assigned for a bad end-of-file record.

The example follows:

```
//JOBATLAS JOB  06#990,SMITH,MSGLEVEL=1
//STEP     EXEC PGM=IEHATLAS
//SYSPRINT DD   SYSOUT=A
//SYSUT1   DD   DSNAME=EOFSET,UNIT=2305,VOLUME=SER=333333,
// DISP=OLD
//SYSIN    DD   *
 TRACK=00000001000003000000
/*
```

The control statements are discussed below:

- SYSPRINT DD defines the device to which the output messages can be written.
- SYSUT1 DD defines the data set that contains the bad record.
- SYSIN DD defines the control data set, which follows in the input stream.
- TRACK defines an end-of-file record on cylinder one, track zero, record three. Input data other than the utility control statement is not required.

IEHATLAS Example 3

In this example, an alternate track is assigned for a bad VTOC record.

The example follows:

```
//JOBTALAS JOB  06#990,SMITH,MSGLEVEL=1
//STEP     EXEC PGM=IEHATLAS
//SYSPRINT DD   SYSOUT=A
//SYSUT1   DD   UNIT=3330,VOLUME=SER=222222,DISP=OLD
//SYSIN    DD   *
 VTOC=000000000005022C0060
D6C2D1C5C3E340404040404040404040404040404040404040404040404040404040404040
40404040F1F2F3F1F1F0F000014401360000000100000000000000000000000000000000000
000040008000000F000F000333333333333333310DDD000001000000A0000000A000000000000000
0000000000000000000000000000000000000000
/*
```

The control statements are discussed below:

- SYSPRINT DD defines the device to which the output messages can be written.
- SYSUT1 DD defines the data set that contains the bad record.
- SYSIN DD defines the control data set, which follows in the input stream.
- VTOC defines the location of the bad VTOC record as track five of cylinder zero. The record number is 2 with a key length of 44. Record length of the bad record is 96.

The input record in this example is a typical hexadecimal record as defined by the VTOC statement. The input record contains 140 bytes (data length = 96, key length = 44).

IEHATLAS Example 4

In this example, the replacement record is record zero.

The example follows:

```
//JOBATLAS  JOB   06#990,SMITH,MSGLEVEL=1
//STEP      EXEC  PGM=IEHATLAS
//SYSPRINT  DD    SYSOUT=A
//SYSUT1    DD    DSNAME=NEWSET,UNIT=3330,VOLUME=SER=333001,
// DISP=OLD
//SYSIN     DD    *
 TRACK=00000002000400000008
0000000000000000
/*
```

The control statements are discussed below:

- SYSPRINT DD defines the device to which the output messages can be written.
- SYSUT1 DD defines the data set that contains the bad record.
- SYSIN DD defines the control data set, which follows in the input stream.
- TRACK specifies the bin, cylinder, and track number for the defective track; and the record number, key length, and data length of the bad record. In this example, the device is a 3330, so the input record has a bin number of zero; the record is to be placed on cylinder 2, track 4, record 0; and it has a key length of zero with a logical record length of 8.

The input record is a typical hexadecimal record as defined by the TRACK statement. The input record contains eight bytes (data length = 8, key length = 0).

IEHDASDR Program

IEHDASDR is a system utility used to prepare direct access volumes for operating system use and to ensure that any permanent hardware errors (that is, defective tracks) incurred on direct access volumes do not seriously degrade the performance of those volumes. (See "Introduction" for general system utility information.)

In addition, IEHDASDR can be used to dump the entire contents or portions of a direct access volume to a volume or volumes of the same direct access device type, to a tape volume or volumes, or to a system output device. Data that is dumped to a magnetic tape volume is arranged so that it can subsequently be *restored* to its original organization by IEHDASDR. The direct access device types supported by IEHDASDR are: 2301, 2302, 2303, 2305, 2311, 2314, 2319, 2321, and 3330.

The program can be used to:

- Analyze tracks, assign alternate tracks for defective tracks, and perform housekeeping and formatting functions to make direct access volumes suitable for IBM System/360 Operating System use.

Note: Defective tracks are flagged when the 3330 disk storage volumes are initialized at the factory. An IEHDASDR job to initialize a 3330 does not perform a surface analysis. The ANALYZE option performs a "quick-DASDI" which includes:

1. Initialization of track zero, including IPL1, IPL2, volume label, and optional IPLTXT.
2. VTOC construction.

- Perform housekeeping and formatting functions on direct access volumes without analyzing tracks.
- Change the volume serial number of a formatted direct access volume.
- Assign alternate tracks for specified defective or questionable tracks on disk or data cell volumes.
- Create a backup or transportable copy of a direct access volume, or list the contents on a system output device.
- Copy *dumped* data from a tape volume to a direct access volume.

Initialize—With Recording-Surface Analysis

IEHDASDR can be used to analyze the recording surface of a direct access device to:

- Assign alternate tracks for any disk or data cell tracks found defective during an analysis, or for any track previously flagged defective. Each track can be analyzed from 1 to 255 times, at the discretion of the user. The test of looking for previously flagged tracks must be suppressed when a new or unformatted direct access volume is being initialized.
- *Standardize* each track by placing a standard home address and a record zero (R0) field on it. The remainder of the track is erased.
- Construct IPL bootstrap records (records 1 and 2 of track 0), a volume label record (record 3 of track 0), and a volume table of contents (VTOC), whose size and placement are determined by the user.
- Provide *owner* information in the volume label record.

Figure 42 shows a direct access volume after it has been prepared for IBM System/360 Operating System use. A direct access volume can be *initialized* in this manner using IEHDASDR.

Initialize—Without Recording-Surface Analysis

IEHDASDR can be used to prepare a direct access volume for use without performing a recording-surface analysis.

A volume might be prepared for system use without recording-surface analysis to:

- Check a direct access volume for previously flagged tracks (except for drum volumes). No formatting is performed on known defective tracks.
- Standardize each track by placing a standard home address and a record zero (R0) field on it. The remainder of the track is erased (except on 3330 volumes).
- Construct IPL bootstrap records (records 1 and 2 of track 0), a volume label record (record 3 of track 0), and a volume table of contents (VTOC), whose size and placement are determined by the user.

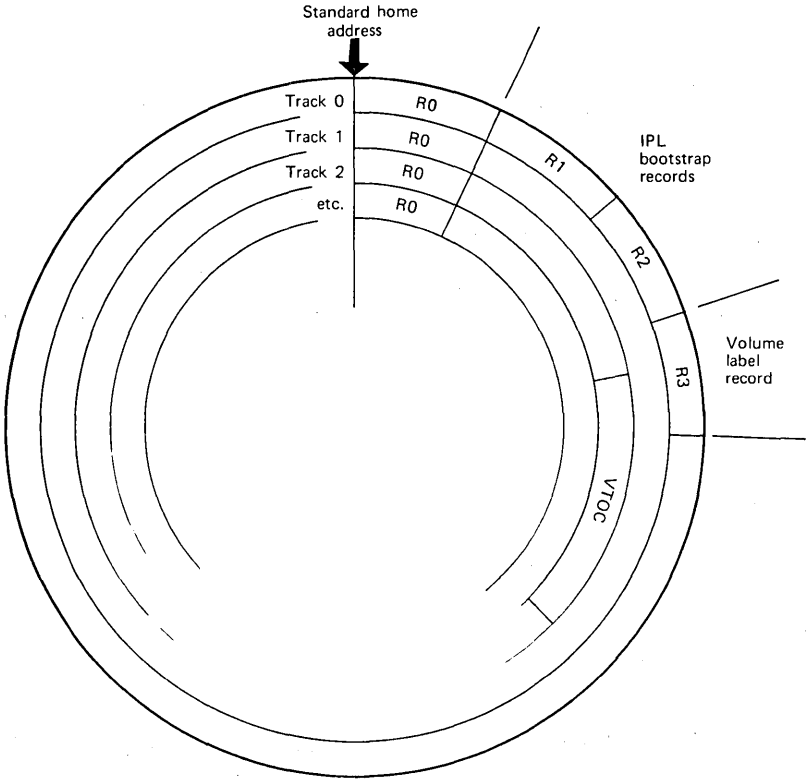


Figure 42. Direct Access Volume Initialized Using IEHDASDR

- Optionally write an IPL program record for 2301, 2303, 2305, 2311, 2314, 2319, and 3330 devices, and provide owner information in the volume label record.

Changing the Volume Serial Number of a Direct Access Volume

IEHDASDR can be used to change the volume serial number of an initialized direct access volume. Optionally, a one- to ten-character owner name can be placed in the volume label record (record 3 of track 0). If an owner name already exists, it is overwritten with the new name.

Note: All cataloged data sets residing on a volume whose label is changed must be recataloged, if the catalog reflects the old serial number.

Assigning Alternate Tracks for Specified Tracks

IEHDASDR can be used to assign an alternate track on a data cell or disk volume. An alternate track can be assigned for any track, whether it is defective or not. If the specified track is an alternate, a new alternate is assigned; if the specified track is an unassigned alternate, it is flagged to prevent its future use.

Note: If it is necessary to assign an alternate track on a drum volume, an IBM Programming Service Representative should be notified.

Creating a Backup, Transportable, or Printed Copy

IEHDASDR can be used to dump a direct access volume or a portion of a volume to any number of tape volumes or volumes of the same direct access device type, or to a system output device. The program can dump a single track, a group of tracks, or an entire volume.

When an entire volume is dumped:

- All primary tracks (for which no alternate tracks are assigned) are dumped.
- When a primary track is found to have an alternate track assigned, the alternate is dumped in place of the primary.

Each track to be dumped will have all of its data except the home address and the count field of record zero (R0) copied to the receiving volume.

A receiving direct access volume retains its own serial number unless the user specifies that it is to be assigned the serial number of the direct access volume being dumped.

Except for a printing operation, only data that is *owned* is dumped; IEHDASDR checks the first or only Free Space (Format 5) data set control block (DSCB) in the volume table of contents. The Free Space (Format 5) DSCB identifies unowned (unused) space on the direct access volume. Whenever an unowned track is encountered, a dummy record, containing a home address and record zero, is written on the receiving

volume. When data is dumped to a system output device, the entire range of specified tracks is dumped.

A printing operation prints each record in hexadecimal. In addition, all printable characters are also represented in EBCDIC.

Figure 43 shows the format of printed output. Each track is identified by its absolute track address (ccccchhhh). The R0 data field is printed on the same line as the track address. Each printed record is preceded by a count field that identifies the applicable track address (ccccchhhh), the record number of the record being printed (rr), and the key and data length (kk and dddd) of the record.

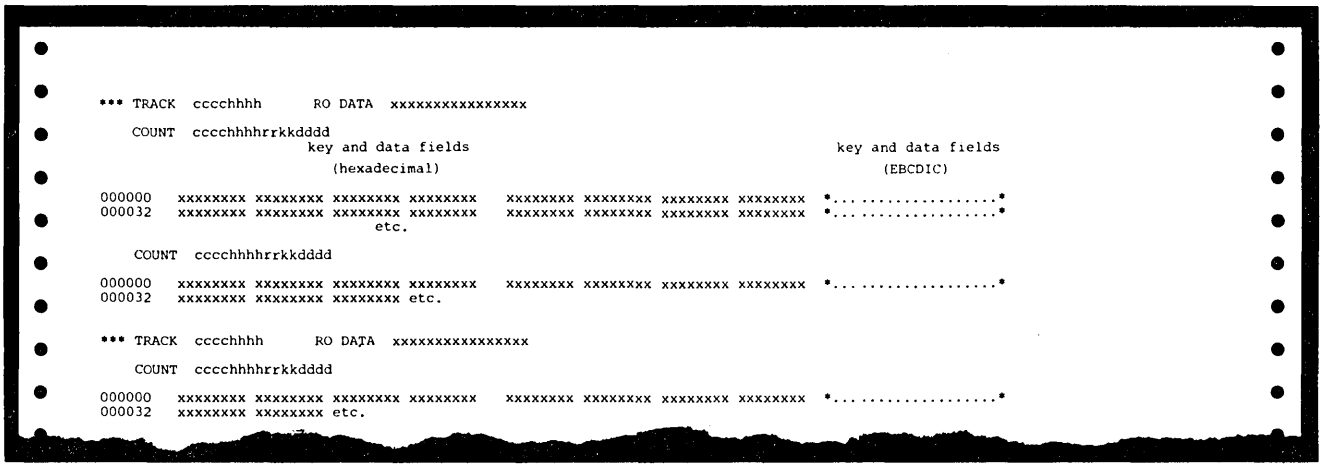


Figure 43. Format of a Direct Access Volume Dumped to a Printer Using IEHDASDR

If an alternate track is printed in place of a primary track, it is identified in the printout by the primary track address.

Copying Dumped Data to a Direct Access Volume

When a direct access volume is dumped to a tape volume, the data is placed in a format that is specially suited for the tape volume. IEHDASDR can be used to restore the format of the dumped data and place the data on the same type of direct access volume as the original volume; that is, data originally dumped from a 2311 volume can be *restored* to a 2311 volume, data dumped from a 2314 volume can be restored to a 2314 volume.

Identical copies of dumped data can be restored to any number of volumes of the same direct access device type as the original volume during the execution of a single restore operation. In addition, data that was dumped by IBCDMPRS can be restored.

A receiving direct access volume retains its own serial number unless the user specifies that it is to be assigned the serial number of the direct access volume originally dumped. If multiple direct access volumes are to be dumped to, and the user specifies that the serial number of the dumped volume is to be propagated, all receiving volumes are assigned that serial number.

Input and Output

IEHDASDR uses as input a control data set containing utility control statements and, optionally, IPL text.

The primary output or result of executing IEHDASDR is determined by the application.

A sequential message data set is created to list informational messages (for example, control statements used), dumped data (for a print operation), and any error messages.

IEHDASDR provides a return code to indicate the results of program execution. The return codes and their meanings are:

- 00, which indicates successful completion.
- 04, which indicates that an unusual condition was encountered; however, the overall result is successful. A warning message is issued.
- 08, which indicates that a specified operation did not complete successfully. An attempt is made to perform any additional operations.
- 16, which indicates that either an error occurred upon invoking IEHDASDR, or IEHDASDR was unable to open the input or message data set. The job step is terminated.

Control

IEHDASDR is controlled by job control statements and utility control statements. The job control statements are used to execute or invoke IEHDASDR and define the data sets used and produced by IEHDASDR.

The utility control statements are used to control the functions of the program.

Job Control Statements

Table 36 shows the job control statements necessary for using IEHDASDR.

Table 36. IEHDASDR Job Control Statements

Statement	Use
JOB	Initiates the job.
EXEC	Specifies the program name (PGM = IEHDASDR) or, if the job control statements reside in a procedure library, the procedure name. Additional information can be entered in the PARM parameter of the EXEC statement; see "PARM Information on the EXEC Statement" below.
SYSPRINT DD	Defines a sequential message data set. The data set can be written to a system output device, a tape volume, or a direct access device.
anyname DD	Defines a direct access device type.
tapename DD	Defines a magnetic tape drive.
SYSIN DD	Defines the control data set. The control data set usually resides in the input stream; however, it can be defined as a blocked or unblocked sequential data set or as a member of a procedure library.

For detailed information on the minimum region size that can be specified for IEHDASDR, see *OS Storage Estimates*, GC28-6551.

The "anyname" DD statement can be entered:

```
//anyname DD UNIT = xxxx,VOLUME = SER = xxxxxx,DISP = OLD
```

If more than one volume is to be processed on a single mountable device, deferred mounting can be specified in the "anyname" DD statement by entering:

```
//anyname DD  UNIT = (xxxx,,DEFER),VOLUME = (PRIVATE,...),
//           DISP = (OLD,KEEP)
```

The "anyname" DD statement is not used for an operation that analyzes an offline direct access volume.

If the volume serial number of a volume to be processed on line is not known, it may be possible to make a nonspecific, PRIVATE volume request on a specific unit; for example:

```
//anyname DD UNIT = (191,,DEFER),VOLUME = PRIVATE,DISP = (NEW,KEEP),
// SPACE = (TRK,(1,1))
```

In this case, the operator is asked to mount a scratch volume on that unit. See "Appendix C: Defining Mountable Devices" for the appropriate DD statement and for a discussion of how to make a nonspecific unit request.

If an IEHDASDR operation produces a volume serial number that is a duplicate of a volume serial number already allocated within the system, the volume to which the duplicate number is assigned is made unavailable to the system. The operator is asked to remove the applicable volume at the completion of the operation.

The "tapename" DD statement can be entered:

```
//tapename DD  UNIT = xxxx,VOLUME = SER = xxxxxx,LABEL = (...,...),
//            DISP = (...,KEEP),DCB = (TRTCH = C,DEN = x)
```

If more than one tape volume is to be processed on the same tape drive, deferred mounting can be specified by:

```
//tapename DD UNIT = (xxxx,,DEFER),VOLUME = (PRIVATE,...)
```

If standard labeled tapes are specified, the DSNAME should also be provided.

The "anyname" DD and "tapename" DD statements are referred to by utility control statements for program operation.

Both the SYSIN and the SYSPRINT data set can have a blocking factor of other than 1.

Data can be dumped from the system residence volume (the IPL volume); however, this is the only IEHDASDR operation that can be performed on that volume.

Because IEHDASDR can change serial numbers and existing data on a direct access volume, operating precautions must be followed by users who have two or more central processing units sharing the same direct access volume.

If IEHDASDR is run in a multiprogramming environment, the user must choose a combination of DD statements (defining mountable devices) that will ensure that volume integrity is maintained. Refer to "Appendix C: DD Statements for Defining Mountable Devices."

If password-protected data sets reside on volumes that are used by IEHDASDR, the following considerations must be made:

- When dumping from a volume containing password-protected data sets, each data set must be described in a separate DD statement having a unique ddname. When the program is executed, the operator must supply the correct password (in answer to a console message) for each password-protected data set.
- When dumping to a tape volume from a direct access volume containing password-protected data sets, the DD statement defining the tape volume must include a DSNAME parameter. In addition, the LABEL parameter must define a standard labeled tape, include a PASSWORD subparameter, and specify or imply a file number of 1.
- When restoring from a tape volume, a DSNAME parameter must be included in the DD statement defining the tape volume.
- During the DUMP, RESTORE, ANALYZE, and FORMAT functions (see "Utility Control Statements"), the direct access "TO" volume is checked for password-protected data sets. At this time the operator must supply the correct password for each password-protected data set encountered.

Refer to *OS Data Management for System Programmers*, GC28-6550, for additional information on password protection.

IEHDASDR can perform up to six concurrent operations of ANALYZE, FORMAT, DUMP or RESTORE operations (see "Utility Control Statements"). This feature, which can shorten the time required to execute the program, is controlled by (1) the number of devices defined for use and (2) the physical arrangement of utility control statements in the input stream. For example, assuming that the required devices are defined and available, a combination of six successive statements of the same type permits six concurrent operations to take place. However, if the utility control statements are arranged so that no operations of the same type appear in succession, no operations are performed concurrently, even though many devices might be defined for use.

Note: The number of concurrent operations allowed can be overridden by an EXEC statement PARM value.

Restrictions

- If IEHDASDR is used to change a volume serial number and a subsequent operation is performed on the newly labeled volume in the same job step, two "anyname" DD statements are required. The VOLUME parameter in the first statement includes the old volume serial number; the VOLUME parameter in the second statement specifies the new volume serial number. In addition, the second statement specifies unit affinity with the first.
- One "anyname" DD statement is required for each device to be used in the job step unless the device is to be processed off line.
- The "tapename" DD statement must be included if a data set is dumped to tape or if a previously dumped data set is to be restored to a direct access volume.
- If BLKSIZE is specified on the SYSIN DD statement, it must be a multiple of 80. If BLKSIZE is omitted from the statement, a block size of 80 bytes is assumed.
- If BLKSIZE is specified on the SYSPRINT DD statement, it must be a multiple of 121. If BLKSIZE is omitted or incorrectly specified, a block size of 121 bytes is assumed.
- SYSIN attributes must be identical if SYSIN data sets are to be concatenated.

PARM Information on the EXEC Statement

The EXEC statement for IEHDASDR can contain PARM information that is used by the program to control line density on output listings and to indicate the maximum number of operations of the same type that can be performed concurrently in the job step.

The EXEC statement can be coded:

```
// EXEC PGM = IEHDASDR    {,PARM = 'N = n'                }
                          {,PARM = 'LINECNT = xx'         }
                          {,PARM = 'LINECNT = xx,N = n'   }
```

The LINECNT value specifies the number of lines per page in the listing of the SYSPRINT data set. The number *xx* is a 2-digit decimal number ranging from 01 to 99. If LINECNT is omitted, the number of lines per page is 58.

The N value specifies a decimal number from one to six that represents the maximum number of like functions that can be performed concurrently by IEHDASDR, assuming that adequate system resources are available. See *OS Storage Estimates*, GC27-6551, for the storage required for each operation. If N is omitted, up to six ANALYZE, FORMAT, DUMP, or RESTORE operations are performed concurrently—according to the number of successive like statements in the input stream. (See "Utility Control Statements.")

System resources permitting, multiple output copies can be specified in any or all of the concurrent operations.

For example, if N = 2 and four DUMP statements appear in succession, the first two dump operations are performed concurrently. As each dump operation is completed and system resources become available, a new dump operation begins.

Utility Control Statements

The utility control statements used to control IEHDASDR are:

- ANALYZE statement, which is used to analyze the recording surface to test for defective tracks, assign alternates for any defective tracks found, and to format the volume to make it ready for use.
- FORMAT statement, which is used to make a volume ready for use without performing an analysis of the recording surface.
- LABEL statement, which is used to change the volume serial number of a direct access volume and, optionally, to update the owner field.
- GETALT statement, which is used to assign an alternate track for a specified disk track or data cell.
- DUMP statement, which is used to dump a single track, a group of tracks, or an entire direct access volume.
- RESTORE statement, which is used to restore a previously dumped direct access volume to a direct access device.
- IPLTXT statement, which signals the beginning of IPL program text statements.

For most operations, multiple copies of a source volume can be made. The program can also perform from two to six ANALYZE, FORMAT, DUMP, or RESTORE operations concurrently, according to the number of successive like statements in the input stream; that is, up to six direct access volumes can be analyzed or formatted, or dumped simultaneously, or up to six magnetic tape (restore) volumes can be processed simultaneously.

ANALYZE Statement

The ANALYZE statement is used to analyze the recording surface of a direct access device. Bit patterns are written on a track, read, and tested for defects. If no defects are found, the track is formatted to make it ready for system use.

The format of the ANALYZE statement is:

```
[label] ANALYZE TODD = {(cuu,...)     }
                       {(ddname,...) }
                ,VTOC = xxxxx
                ,EXTENT = xxxxx
                [,NEWVOLID = serial]
                [,IPLDD = ddname]
                ,FLAGTEST = {YES }
                            {NO  }
                [,PASSES = {n     }]
                           {0     }
                [,OWNERID = name]
                [,PURGE = {YES }]
                          {NO  }
```

where:

TODD =
specifies the volume to be processed. If multiple volumes are specified in an ANALYZE statement and an abnormal completion of the ANALYZE operation occurs, the operation is terminated on all volumes. These values can be coded:

(cuu,...)
specifies the channel and unit address of a direct access device containing a volume to be initialized. This value is used only for the analysis of a volume off line, which includes the first analysis of a volume. If the volume to be processed is a 2321 volume, TODD = cuu/b is specified, where *cuu* is the channel and unit address of the device, and *b* is the bin number of the volume; for example, TODD = 150/0 specifies bin 0 with a channel and unit address of 150. If this value is coded, no DD statement defining a mountable device is required. When this volume is coded, the specified devices must be varied off line (by use of the VARY OFFLINE command) prior to the execution of the job step.

(ddname,...)
specifies the ddname of a job control statement defining a direct access device containing a volume to be analyzed, formatted, and labeled. Multiple ddnames specifying additional job control statements can be included unless the **TODD** device is a 2321, in which case only one DD statement can be referred to.

VTOC = *xxxxx*
specifies a one- to five-byte decimal relative track address representing a primary track on which the volume table of contents is to begin. The VTOC cannot occupy track 0, and it cannot occupy track 1 if IPL text is written for 2303 and 2311 volumes.

EXTENT = *xxxxx*
specifies the decimal length of the VTOC in tracks. The VTOC cannot extend into the alternate track area or onto a second volume.

NEWVOLID = *serial*
specifies a one- to six-character serial number. The serial number is assigned to all direct access volumes processed through the use of this control statement. If **NEWVOLID** is omitted, all direct access volumes retain their own serial numbers. This parameter is required for the analysis of a volume off line.

IPLDD = *ddname*
specifies the ddname of a DD statement defining the data set containing the IPL program. The IPL program can be included in the SYSIN (input stream) data set, or it can be defined as a sequential data set or a member of a partitioned data set. If IPL text is included in the input stream, an IPLTXT statement is used to separate the ANALYZE statement from the IPL program text statements. The maximum IPL record size is 3625 bytes. **IPLDD** applies to 2301, 2303, 2305, 2311, 2314, 2319, and 3330 volumes.

Note: Caution should be used to ensure that the DD statement being referred to is coded correctly.

FLAGTEST =
specifies whether a check is to be made for previously flagged tracks. **FLAGTEST** is applicable to disk and data cell volumes, not to drum volumes. When a volume is initialized off line, no check is made. These values can be coded:

YES
specifes that each track is to be checked to see if it was previously flagged as defective. If **FLAGTEST** is omitted, **YES** is assumed.

NO
specifies that the program is not to check for previously flagged tracks on the volume.

PASSES =
specifies the number of passes to be made in analyzing a recording surface. These values can be coded:

n
specifies the number of times a bit pattern test is to be performed. The *n* value is a decimal number from 1 to 255. If **PASSES** is omitted, the bit pattern test is performed once on each track. This value is applicable to disk and drum volumes, with the exception of 3330 volumes.

0
specifies that the ANALYZE function is to bypass all surface analysis and track formatting, writing only a VTOC, track zero records (IPL bootstrap and volume label records), and IPL text if requested. The 0 value applies to all direct access volumes supported by IEHDASDR. If the device is a 3330, only the 0 specification is allowed, and is forced regardless of the **PASSES** parameter. This value is applicable to all previously initialized direct access devices supported by IEHDASDR while on line. The offline "quick-DASDI" feature is only supported for 2314, 2305, and 3330 volumes.

OWNERID = *name*
specifies a one- to ten-character name or other identifying information to be placed in the volume label record. **OWNERID** is specified as an EBCDIC character string with the exclusion of the blank and the comma characters (these terminate the control card scan of a field or an entire card).

PURGE =
specifies whether the ANALYZE operation is to be terminated when an unexpired data set is encountered. If **PURGE** is omitted, and an unexpired data set is encountered, the ANALYZE operation is terminated. These values can be coded:

YES
indicates that all unexpired data sets on the volume can be overwritten provided that the operator signals his concurrence when the first unexpired data set is encountered.

NO
specifies that the ANALYZE operation is to be terminated if an unexpired data set is encountered. This is the default.

If PURGE = YES is coded and an unexpired data set is encountered, the operator is prompted. The operator replies are:

- U, which indicates that all unexpired data sets on this volume can be overwritten. (The ANALYZE operation continues.)
- T, which indicates that this volume contains unexpired data sets that must not be overwritten. (The ANALYZE operation is terminated.)

The **PURGE** parameter does not apply to password-protected data sets; that is, the operator must always respond with the proper password for each password-protected data set encountered. If he is unable to do so, the ANALYZE operation is terminated.

Note: If the device is on line and a volume label and VTOC are present, they are read, and the information contained in them is used to initialize the volume. If the device is off line, the volume label and VTOC are ignored.

FORMAT Statement

The FORMAT statement is used to prepare a volume for operating system use. Except for flag testing, no analysis is made prior to formatting a track. Previously flagged disk tracks remain flagged; alternate tracks are assigned, where applicable. Because the FORMAT statement does not cause recording-surface analysis, it should not be used for the first initialization of a volume.

Output produced as a result of using the FORMAT statement includes a list of defective tracks and their assigned alternates.

The FORMAT statement is not applicable to the 2321 data cell. The ANALYZE function should be used when initializing a 2321 volume. (If FORMAT is specified for a 2321 volume, the ANALYZE function is automatically performed.)

Note: If a command reject is detected while a FORMAT operation is performed on an assigned alternate track on an IBM 2305 Fixed Head Storage volume, processing continues as if no alternate track existed. No action need be taken if message IEH4000I is typed out on the operator's console in response to this condition. If FORMAT cannot read a home address, it flags the defective track and assigns an alternate track.

The format of the FORMAT statement is:

```
[label] FORMAT  TODD = (ddname,...)
                ,VTOC = xxxxx
                ,EXTENT = xxxxx
                [,NEWVOLID = serial]
                [,IPLDD = ddname]
                [,OWNERID = name]
                [,PURGE = {YES }]
                          {NO  }
```

where:

TODD = (*ddname*,...)
specifies the ddname of a job control statement defining a direct access device containing a volume to be formatted. Multiple ddnames specifying additional job control statements can be included. If multiple volumes are specified in a FORMAT statement and an abnormal completion of the FORMAT operation occurs, the operation is terminated on all volumes.

VTOC = *xxxxx*
specifies a one- to five-byte decimal relative track address representing a primary track on which the volume table of contents is to begin. The VTOC cannot occupy track 0, or track 1 if IPL text is written for 2303 and 2311 volumes.

EXTENT = *xxxxx*
specifies the decimal length of the VTOC in tracks. The VTOC cannot extend into the alternate track area or to a second volume.

NEWVOLID = *serial*
specifies a one- to six-character serial number. The serial number is assigned to all direct access volumes processed through the use of this control statement. If **NEWVOLID** is omitted, the direct access volumes retain their own serial numbers.

IPLDD = *ddname*
specifies the ddname of a DD statement defining the data set containing the IPL program. The IPL program can be included in the SYSIN (input stream) data set, or it can be defined as a sequential data set or a member of a partitioned data set. If IPL text is included in the input stream, an IPLTXT statement is used to separate the FORMAT statement from the program text statements. Maximum IPL record size is 3625 bytes. This parameter applies to 2301, 2303, 2305, 2311, 2314, 2319, and 3330 volumes.

Note: Caution should be used to ensure that the DD statement being referred to is coded correctly. An incorrectly DD statement results in a loss of the time required to perform the FORMAT function.

OWNERID = *name*
specifies a one- to ten-character name or other identifying information to be placed in the volume label record. **OWNERID** is specified as an EBCDIC character string with the exclusion of the blank and the comma characters (these terminate the control card scan of a field or an entire card).

PURGE =
specifies whether the FORMAT operation is to be terminated when an unexpired data set is encountered. If **PURGE** is omitted and an unexpired data set is encountered, the FORMAT operation is terminated. These values can be coded:

YES
indicates that all unexpired data sets on the volume can be overwritten provided that the operator signals his concurrence when the first unexpired data set is encountered.

NO
specifies that the FORMAT operation is to be terminated when an unexpired data set is encountered. This is the default.

If PURGE = YES is coded and an unexpired data set is encountered, the operator is prompted. The operator replies are:

- U, which indicates that all unexpired data sets on this volume can be overwritten. (The FORMAT operation continues.)
- T, which indicates that this volume contains unexpired data sets that must not be overwritten. (The FORMAT operation is terminated.)

The **PURGE** parameter does not apply to password-protected data sets; that is, the operator must always respond with the proper password for each password-protected data set encountered. If he is unable to do so, the FORMAT operation is terminated.

LABEL Statement

The LABEL statement is used to change the serial number of a direct access volume and, optionally, to update the owner field in record 3 of track 0. One LABEL statement must be included for each volume that is to have its label changed.

The format of the LABEL statement is:

```
[label] LABEL  TODD = {cuu     }
                      {ddname }
               ,NEWVOLID = serial
               [,OWNERID = name]
```

where:

TODD =
specifies the volume to be processed. These values can be coded:

cuu
specifies the channel and unit address of a direct access device containing a volume whose serial number is to be changed. This value is used only for labeling an off line volume. If the volume to be processed is a 2321 volume, TODD = cuu/b is specified, where *cuu* is the channel and unit address of the device and *b* is the bin number of the volume. If this volume is coded, no DD statement defining a mountable device is required. When this volume is coded, the specified device must be varied off line (by use of the VARY OFFLINE command) prior to the execution of the job step.

ddname
specifies the ddname of a job control statement defining a direct access device containing a volume whose serial number is to be changed.

NEWVOLID = *serial*
specifies a one- to six-character serial number. The serial number is assigned to the direct access volume processed through the use of this control statement.

OWNERID = *name*
specifies a one- to ten-character name or other identifying information. If **OWNERID** is omitted, the old owner information, if any, is retained. **OWNERID** is specified as an EBCDIC character string with the exclusion of the blank, the dash, and the comma characters (these terminate the control card scan of a field or an entire card).

GETALT Statement

The GETALT statement is used to assign an alternate track for a specified data cell or disk track if the volume was previously initialized.

The format of the GETALT statement is:

```
[label] GETALT  TODD = ddname
                ,TRACK = ccchhhh
```

where:

TODD = *ddname*
specifies the ddname of a job control statement defining a data cell or disk device containing a volume on which an alternate track is to be assigned.

TRACK = *ccchhhh*
specifies in hexadecimal the cylinder number, cccc, and head number, *hhhh*, of a track for which an alternate track is requested. (When referring to a 2321 volume, cccc is the subcell and strip address, and *hhhh* is the cylinder and head address. **TRACK** cannot specify track 0 or the first track occupied by the VTOC.

Note: Flags set by the GETALT statement for defective 3330 tracks cannot be reset by IEHDASDR.

DUMP Statement

The DUMP statement dumps a single track, a group of consecutive tracks, or an entire direct access volume to one or more direct access volumes of the same device type, to one or more tape volumes, or to a system output device (printer assumed).

A detailed description of the tape volume format is available in the *IBM System/360 Operating System: Utilities, Program Logic Manual*, GY28-6614.

The format of the DUMP statement is:

```
[label] DUMP  FROMDD = ddname
              ,TODD = (ddname,...)
              [,CPYVOLID = {YES}
                           {NO}]
              [,BEGIN = ccchhhh]
              [,END = ccchhhh]
              [,PURGE =    {YES}
                           {NO}]
```

where:

FROMDD = *ddname*
specifies the ddname of the DD statement defining the device containing the direct access volume from which a copy or copies are to be made.

TODD = (*ddname*,...)
specifies the ddname of the system output device (SYSPRINT) or specifies the ddnames of the DD statements defining the devices containing the direct access or tape volumes on which copies are to be made. If TODD = SYSPRINT is coded, the direct access volume described by **FROMDD** is dumped to the system output device. If a permanent data check or missing address marker is encountered while reading the direct access volume, the defective records are identified and printed. Output may exceed the expected data size due to a data check in the count field of the error record. When dumping from a 2305 or 3330 to SYSPRINT, the SPACE parameter must be used on the SYSPRINT DD card. Allocate sufficient space, for example, "5,(5)", to allow large capacity devices to dump to SYSPRINT.

CPYVOLID =
specified whether receiving direct access volumes are to be assigned the serial number of the dumped volume. If **CPYVOLID** is omitted, receiving volumes keep their own serial numbers. These values can be coded:

YES
specifies that all receiving direct access volumes are to be assigned the serial number of the dumped volume.

NO
specifies that receiving volumes are to keep their own serial numbers. This is the default.

BEGIN = *ccchhhh*
specifies in hexadecimal a cylinder number, *cccc* and head number, *hhhh*, that identify the first track to be dumped. (When referring to a 2321 volume, *cccc* is the subcell and strip address, and *hhhh* is the cylinder and head address.) If **BEGIN** is omitted, the dump operation begins with track 0.

END = *ccchhhh*
specifies in hexadecimal a cylinder number, *cccc*, and head number, *hhhh*, that identify the last track to be dumped. If only one track is to be dumped, both **BEGIN** and **END** specify that track address. (When referring to a 2321 volume, *cccc* is the subcell and strip address, and *hhhh* is the cylinder and head address.) If **END** is omitted, the last primary track of the volume is the last track to be copied. (Alternate tracks are not dumped unless they are assigned as alternates.)

PURGE =
specifies whether the dump operation is to be terminated when an unexpired data set is encountered. If **PURGE** is omitted, the dump operation is terminated when an unexpired data set is encountered. **PURGE** does not apply when dumping to a restore tape. These values can be coded:

YES
indicates that all unexpired data sets on a receiving direct access volume can be overwritten, provided that the operator signals his concurrence when the first unexpired data set is encountered.

NO
specifies that the dump operation is to be terminated when an unexpired data set is encountered. This is the default.

If PURGE = YES is coded and an unexpired data set is encountered, the operator is prompted. The operator replies are:

- U, which indicates that all unexpired data sets on the receiving direct access volume can be overwritten. (The DUMP operation continues.)
- T, which indicates that the receiving direct access volume contains unexpired data sets that must not be overwritten. (The DUMP operation is terminated.)

The **PURGE** parameter does not apply to password-protected data sets; that is, the operator must always respond with the proper password for each password-protected data set encountered. If he is unable to do so, the dump operation is terminated.

An extra input/output error (data check) message is generated at the console when the dump to SYSPRINT function encounters one of the following conditions:

- Missing address marker.
- Data check in count and key fields and/or data field.
- Input/output error on a search command.
- Missing address marker and no record found.

The additional data check message printed at the console is generated by the dump function's error recovery procedure. However, the additional message is not reflected by a SYNADAF message in the SYSPRINT data set. If a missing address marker is encountered during a space count command, the function terminates with a return code of 8.

Note: If multiple output volumes are specified in a DUMP statement and an abnormal completion of the DUMP operation occurs, the operation is terminated on all output volumes.

Do not dump a volume and restore new data to that volume in the same job step. IEHDASDR does not *flush* the input stream if an operation is unsuccessful; that is, the program attempts to perform any remaining functions after encountering an error. Thus, if a dump operation is unsuccessful, data is lost if a subsequent restore operation places new data on the dumped volume.

Partial dumps of direct access volumes should be used with extreme caution. Because only those tracks that are dumped are placed on the receiving volume, the partially dumped data may not be usable. When partially dumped data is subsequently restored, it is placed on the same tracks that it originally occupied.

When using the DUMP statement, do not specify the same ddname in more than one **TODD** parameter in a single job step, except when the ddname is SYSPRINT.

When space permits, more than one direct access volume can be dumped to a restore tape. However, IEHDASDR creates two files for each volume of data that is dumped. Therefore, the LABEL parameter sequence number in the DD statement defining the restore volume must be coded as 3, 5, 7, etc. for the second, third, fourth, etc. volume dumped to the restore tape.

In the case of an IPL restore tape, the LABEL parameter sequence number must be coded as 2, 4, 6, etc. for the first, second, third, etc. volume dumped to the restore tape.

The files are referred to in the same manner when restoring data to a direct access device.

When processing an unlabeled tape before a dump operation, the IEHDASDR writes an end-of-file record (tapemark) and continues processing.

When dumping to or restoring from a tape, specified as standard label or "BLP", a disposition of KEEP should be specified in the DD statement for the tape. Unlabeled tapes may have other disposition parameters.

When restoring from a restore file on a tape, the same file sequence number and tape label format used in the dump operation must be used.

Intermixing of restore files with system data sets is not recommended because of the unique format of the restore file. C

RESTORE Statement

The RESTORE statement is used to restore a direct access volume or volumes from a tape volume on which a dumped copy was previously placed.

Note: When a standard labeled restore tape created by IBCDMPRS is restored by IEHDASDR, the DD card describing the tape for IEHDASDR can specify LABEL = (2,BLP). Bypass label processing must have been system generated by

specifying OPTIONS = BYLABEL on the SCHEDULR control card. If bypass label processing is not available, any standard labeled tape created by IBCDMPRS can be restored by IEHDASDR, by providing appropriate DCB parameters on the DD statement for the tape (RECFM = U,BLKSIZE = track length).

The format of the RESTORE statement is:

```
[label] RESTORE  TODD = (ddname,...)
                 ,FROMDD = ddname
                 [,CPYVOLID = {YES }]
                              {NO  }
                 [,PURGE = {YES }]
                           {NO  }
```

where:

TODD = (*ddname*,...)
specifies the ddnames of the DD statements defining the devices containing the direct access volumes to be restored. If multiple output volumes are specified in a RESTORE statement and an abnormal completion of the restore operation occurs, the operation is terminated on all output volumes.

FROMDD = *ddname*
specifies the ddname of the DD statement defining the tape volume containing the data to be restored. If more than one tape volume is to be used as input, the DD statement for the tape must indicate multiple volumes.

CPYVOLID
specifies whether restored direct access volumes are to be assigned the serial number of the dumped direct access device. If **CPYVOLID** is omitted, receiving volumes keep their own serial numbers. These values can be coded:

YES
specifies that all restored direct access volumes are to be assigned the serial number of the dumped direct access volume.

NO
specifies that receiving volumes are to keep their own serial numbers. This is the default.

PURGE =
specifies whether the restore operation is to be terminated when an unexpired data set is encountered. If **PURGE** is omitted, the restore operation is terminated when an unexpired data set is encountered.

YES
specifies that all unexpired data sets on the receiving direct access volume can be overwritten provided that the operator signals his concurrence when the first unexpired data set is encountered.

NO
specifies that the restore operation is to be terminated if an unexpired data set is encountered. This is the default.

If PURGE = YES is coded and an unexpired data set is encountered, the operator is prompted. The operator replies are:

- U, which indicates that all unexpired data sets on this volume can be overwritten. (The restore operation continues.)
- T, which indicates that this volume contains unexpired data sets that must not be overwritten. (The restore operation is terminated.)

The **PURGE** parameter does not apply to password-protected data sets; that is, the operator must always respond with the proper password for each password-protected data set encountered. If he is unable to do so, the restore operation is terminated.

IPLTXT Statement

The IPLTXT statement is used to mark the beginning of IPL program text statements. The IPL text must follow the first statement referring to it.

IPL text need be included only once in the input stream; that is, IEHDASDR refers to the first copy of IPL text encountered when performing multiple functions in a single job step.

The format for the IPLTXT statement is:

```
[label] IPLTXT
```

IEHDASDR Examples

The following examples illustrate some of the uses of IEHDASDR. Table 37 can be used as a quick reference guide to IEHDASDR examples. The numbers in the "Example" column point to examples that follow.

Table 37. IEHDASDR Example Directory

Operation	Device	Comments	Example
INITIALIZE	2314 Disk	Volume is to be initialized for the first time; therefore, recording serface is analyzed. IPL text is included in the input stream.	1
INITIALIZE	2311 Disks	Three previously initialized volumes are to be initialized; their volume serial numbers are to be changed. Surface analysis is to be performed at the same time.	2
GETALT and LABEL	2321 Disk	Get alternate tracks for a previously initialized volume and change its volume serial number.	3
DUMP	2314 Disks	Dump a copy of one volume to three other volumes.	4
DUMP	2311 Disk, system output device	Dump a group of tracks to the system output device, which is assumed to be a printer.	5
DUMP	2314 Disk, 9-track tape	Dump a disk volume to magnetic tape. Only one tape volume is required.	6
RESTORE	2311 Disks, 7-track tape	A 2311 disk volume, previously dumped to tape, is to be restored to direct access.	7
DUMP and RESTORE	2314 Disks, 9-track tape	Dump operations are to be performed concurrently to minimize input/output time. Restore operations are to be performed concurrently to minimize input/output time.	8
RESTORE	9-track tape, 2314 Disk	A 2314 volume, previously dumped to two tape volumes, is to be restored to disk.	9
FORMAT	3330 Disk	Format a disk by writing an R0 on each track. IPL text is included in the input stream. The volume serial is changed.	10
FORMAT	3330 Disks	Format two disks.	11
Quick-DASDI	3330 Disk	Use the ANALYZE function to build the VTOC and change the volume serial number.	12

IEHDASDR Example 1

In this example, a blank 2314 volume is to be analyzed and formatted for the first time. Because this example deals with a blank volume, two considerations must be made:

1. The **TODD** parameter in the ANALYZE statement must specify a channel and unit address, rather than a ddname.
2. The selected device (in this example, unit 190) must be varied offline by the operator; that is, before the job is executed, the operator must use the VARY OFFLINE command.

The example follows:

```
//DASDR1   JOB
//         EXEC PGM=IEHDASDR
//SYSPRINT DD   SYSOUT=A
//SYSIN    DD   *
  ANALYZE   TODD=190,VTOC=00004,EXTEND=00010,                    ¢C
                NEWVOLID=231400,OWNERID=SMITH,                   ¢C
                IPLDD=SYSIN,FLAGTEST=NO
    IPLTXT
 TXT
```

(IPL text)

```
 TXT
 END
/*
```

The control statements are discussed below:

- SYSIN DD defines the control data set, which follows in the input stream.

- ANALYZE defines a mountable device on which a blank 2314 volume is to be mounted. This statement defines the starting location and extent of a volume table of contents, specifies a serial number and owner identification, indicates that no flag testing is to be performed, and indicates that IPL text is included in the input stream.
- IPLTXT signals the start of IPL text.
- END signals the end of IPL text.

IEHDASDR Example 2

In this example, three previously initialized 2311 volumes are to be initialized and assigned new serial numbers.

The example follows:

```
//DASDR2   JOB
//         EXEC PGM=IEHDASDR
//SYSPRINT DD   SYSOUT=A
//VOL1     DD   UNIT=(2311,,DEFER),DISP=OLD,
// VOLUME=(PRIVATE,,SER=(231100))
//VOL2     DD   UNIT=(2311,,DEFER),DISP=OLD,
// VOLUME=(PRIVATE,,SER=(231101))
//VOL3     DD   UNIT=(2311,,DEFER),DISP=OLD,
// VOLUME=(PRIVATE,,SER=(231102))
//SYSIN    DD   *
  ANALYZE   TODD=VOL1,VTOC=00003,EXTENT=00010,                  %C
               OWNERID=SMITH,NEWVOLID=DISK01,FLAGTEST=NO
  ANALYZE   TODD=VOL2,VTOC=00006,EXTENT=00010,                  %C
               OWNERID=SMITH,NEWVOLID=DISK02,FLAGTEST=NO
  ANALYZE   TODD=VOL3,VTOC=00004,EXTENT=00010,                  %C
               OWNERID=SMITH,NEWVOLID=DISK03,FLAGTEST=NO
/*
```

The control statements are discussed below:

- VOL1, VOL2, and VOL3 DD define three 2311 devices on which the volumes to be initialized are mounted.
- SYSIN DD defines the control data set, which follows in the input stream.
- The ANALYZE statements indicate the ddnames of DD statements defining devices on which the three 2311 volumes (231100, 231101, and 231102) are to be mounted. The ANALYZE statements also define starting locations and extents of the three volume tables of contents, specify new owner names and serial numbers (DISK01, DISK02, and DISK03), and indicate that no flag testing is to be performed on these volumes.

IEHDASDR Example 3

In this example, alternate tracks are to be assigned for three suspected defective tracks on a 2321 volume.

The example follows:

```
//DASDR3   JOB
//         EXEC PGM=IEHDASDR
//SYSPRINT DD   SYSOUT=A
//VOLUME1  DD   UNIT=(2321,,DEFER),DISP=OLD,
// VOLUME=(PRIVATE,,SER=(232100))
//SYSIN    DD   *
       GETALT   TODD=VOLUME1,TRACK=05070310
       GETALT   TODD=VOLUME1,TRACK=0A06020F
       GETALT   TODD=VOLUME1,TRACK=0A070311
        LABEL   TODD=VOLUME1,NEWVOLID=DISK00,OWNERID=SMITH
/*
```

The control statements are discussed below:

- VOLUME1 DD defines a device that is to contain the 2321 volume (232100).
- SYSIN DD defines the control data set, which follows in the input stream.
- The GETALT statements specify the ddname of the DD statement defining the device on which the 2321 volume is mounted. The GETALT statements specify the relative track addresses of the tracks for which alternates are to be assigned.
- LABEL specifies the ddname of the DD statement defining the device on which the 2321 volume is mounted. The LABEL statement changes the serial number of the 2321 volume from 232100 to DISK00.

IEHDASDR Example 4

In this example, a copy of an entire volume (231400) is to be dumped onto three volumes (231401, 231402, and 231403).

The example follows:

```
//DASDR4   JOB
//         EXEC PGM=IEHDASDR
//SYSPRINT DD   SYSOUT=A
//DUMPFROM DD   UNIT=(2314,,DEFER),DISP=OLD,
// VOLUME=(PRIVATE,,SER=(231400))
//DUMPTO1  DD   UNIT=(2314,,DEFER),DISP=OLD,
// VOLUME=(PRIVATE,,SER=(231401))
//DUMPTO2  DD   UNIT=(2314,,DEFER),DISP=OLD,
// VOLUME=(PRIVATE,,SER=(231402))
//DUMPTO3  DD   UNIT=(2314,,DEFER),DISP=OLD,
// VOLUME=(PRIVATE,,SER=(231403))
//SYSIN    DD   *
 DUMP   FROMDD=DUMPFROM,TODD=(DUMPTO1,DUMPTO2,DUMPTO3),      ℛC
              PURGE=YES
/*
```

The control statements are discussed below:

- DUMPFROM DD defines a mountable device that is to contain a source volume.
- DUMPTO1, DUMPTO2, and DUMPTO3 DD define mountable devices that are to contain the three receiving volumes.
- DUMP specifies the dump operation and identifies the DD statements defining the applicable devices. All receiving volumes are to retain their own serial numbers.

IEHDASDR Example 5

In this example, a copy of tracks 0 through 60 is to be dumped from a disk volume (231100) to a system output device.

The example follows:

```
//DASDR5   JOB
//         EXEC PGM=IEHDASDR
//SYSPRINT DD   SYSOUT=A
//DEV      DD   UNIT=(2311,,DEFER),DISP=OLD,
// VOLUME=(PRIVATE,,SER=(231100))
//SYSIN    DD   *
 DUMP   FROMDD=DEV,TODD=SYSPRINT,BEGIN=00000000,END=00050009
/*
```

The control statements are discussed below:

- DEV DD defines a mountable device that is to contain the source volume.
- DUMP specifies the dump operation, identifies the DD statements defining the source and receiving devices, and identifies the tracks that are to printed.

IEHDASDR Example 6

In this example, a 2314 volume (231400) is to be dumped to a 9-track, 800 bits per inch, tape volume (240000).

The example follows:

```
//DASDR6   JOB
//         EXEC PGM=IEHDASDR
//SYSPRINT DD   SYSOUT=A
//SOURCE   DD   UNIT=(2314,,DEFER),DISP=OLD,
// VOLUME=(PRIVATE,,SER=(231400))
//RECEIVE  DD   UNIT=(2400,,DEFER),DISP=NEW,DSNAME=TAPE1,
// VOLUME=(PRIVATE,,SER=(240000))
//SYSIN    DD   *
        DUMP   FROMDD=SOURCE,TODD=RECEIVE
/*
```

Note: This example assumes that only one tape volume is required. If more than one is required, code the volume serial numbers of the additional volumes in the VOLUME parameter of the DD statement that defines the magnetic tape device. For unlabeled tapes, include a volume count in the DD statement.

The control statements are discussed below:

- SOURCE DD defines a mountable device that is to contain the source volume.
- RECEIVE DD defines a 9-track tape drive that is to contain the receiving tape volume.
- DUMP specifies the dump operation and identifies the DD statements defining the source and receiving devices.

IEHDASDR Example 7

In this example, three disk volumes (231100, 231101, and 231102) are to be restored from a 7-track, 556 bits per inch, standard labeled, tape volume.

The example follows:

```
//DASDR7   JOB
//         EXEC PGM=IEHDASDR
//SYSPRINT DD   SYSOUT=A
//TAPE     DD   UNIT=(2400-2,,DEFER),DISP=OLD,DCB=(TRTCH=C,
// ,DEN=1),DSNAME=TAPE1,VOLUME=(PRIVATE,,SER=(240000))
//DIRACC1  DD   UNIT=(2311,,DEFER),DISP=OLD,
// VOLUME=(PRIVATE,,SER=(231100))
//DIRACC2  DD   UNIT=(2311,,DEFER),DISP=OLD,
// VOLUME=(PRIVATE,,SER=(231101))
//DIRACC3  DD   UNIT=(2311,,DEFER),DISP=OLD,
// VOLUME=(PRIVATE,,SER=(231102))
//SYSIN    DD   *
     RESTORE   TODD=(DIRACC1,DIRACC2,DIRACC3),FROMDD=TAPE
/*
```

The control statements are discussed below:

- TAPE DD defines a 7-track tape drive that is to contain the source tape volume.
- DIRACC1, DIRACC2, and DIRACC3 DD define mountable devices that are to contain the three receiving volumes.
- RESTORE specifies the restore operation and identifies the DD statements defining the source and receiving devices. The receiving volumes retain their own serial numbers.

IEHDASDR Example 8

In this example, two direct access volumes are to be dumped concurrently to two receiving volumes in one operation; two direct access volumes are to be restored concurrently from two 9-track, 800 bits per inch, standard labeled, tape volumes in another operation.

The example follows:

```
//DASDR8   JOB
//         EXEC PGM=IEHDASDR
//SYSPRINT DD   SYSOUT=A
//SOURCE1  DD   UNIT=(2314,,DEFER),DISP=OLD,
// VOLUME=(PRIVATE,,SER=(231400))
//SOURCE2  DD   UNIT=(2314,,DEFER),DISP=OLD,
// VOLUME=(PRIVATE,,SER=(231401))
//TO1      DD   UNIT=2314,VOLUME=SER=231402,DISP=OLD
//TO2      DD   UNIT=2314,VOLUME=SER=231403,DISP=OLD
//SOURCE3  DD   UNIT=(2400,,DEFER),DISP=OLD,LABEL=(,NL),
// VOLUME=(PRIVATE,,SER=(240000))
//SOURCE4  DD   UNIT=(2400,,DEFER),DISP=OLD,LABEL=(,NL),
// VOLUME=(PRIVATE,,SER=(240001))
//TO3      DD   UNIT=AFF=TO1,VOLUME=SER=231404,DISP=OLD
//TO4      DD   UNIT=AFF=TO2,VOLUME=SER=231405,DISP=OLD
//SYSIN    DD   *
         DUMP  FROMDD=SOURCE1,TODD=TO1
         DUMP  FROMDD=SOURCE2,TODD=TO2
      RESTORE  TODD=TO3,FROMDD=SOURCE3
      RESTORE  TODD=TO4,FROMDD=SOURCE4
/*
```

The control statements are discussed below:

- SOURCE1 and SOURCE2 DD define devices on which the source volumes for the dump operation are to be mounted.
- TO1 and TO2 DD define devices on which the receiving volumes for the dump operation are to be mounted.
- SOURCE3 and SOURCE4 DD define devices on which the source tape volumes for the restore operation are to be mounted.
- TO3 and TO4 DD define devices on which the receiving direct access volumes for the restore operation are to be mounted. The receiving volumes for the restore operation are to be mounted on the same devices as the receiving volumes for the dump operation were mounted.

IEHDASDR Example 9

In this example, a 2314 volume previously dumped to tape is to be restored. Because a completely filled 2314 volume requires more space than is available on a single reel of 9-track, 800 bits per inch tape, two tape volumes were used in the dump operation.

The example follows:

```
//DASDR9   JOB  00#990,SMITH
//         EXEC PGM=IEHDASDR
//SYSPRINT DD   SYSOUT=A
//TAPE     DD   UNIT=2400,VOL=(,,,2,SER=(240000,240001)),
// DISP=OLD
//DISK     DD   UNIT=2314,VOL=SER=231400,DISP=OLD
//SYSIN    DD   *
   RESTORE  FROMDD=TAPE,TODD=DISK
/*
```

The control statements are discussed below:

- TAPE DD defines the 9-track tape volumes that contain the data to be restored to disk.
- DISK DD defines the 2314 volume to which data is to be restored.
- RESTORE specifies that data is to be restored from the tape volumes defined in the TAPE DD statement to the 2314 volume defined in the DISK DD statement.
- For unlabeled tapes, use the external volume identification and the LABEL = (,NL) parameter on the associated tape DD statement.

IEHDASDR Example 10

In this example, a 3330 volume is formatted and assigned a new serial number.

The example follows:

```
//DASDR10  JOB
//         EXEC PGM=IEHDASDR
//SYSPRINT DD   SYSOUT=A
//DISK     DD   UNIT=3330,DISP=OLD,VOL=(PRIVATE,,SER=(333000))
//SYSIN    DD   *
 FORMAT    TODD=DISK,VTOC=00006,EXTENT=00005,                    %C
             NEWVOLID=333001,PURGE=YES,IPLDD=SYSIN
 IPLTXT
 TXT
 (IPL text)
 TXT
 END
/*
```

The control statements are discussed below:

- DISK DD defines the 3330 device on which the volume (333000) is mounted.
- SYSIN DD defines the control data set which follows in the input stream.
- FORMAT defines a starting location and extent of a volume table of contents, specifies a new serial number, and indicates that the IPL text is included in the input stream. Record 0 of each track is rewritten.
- IPLTXT signals the start of IPL text.
- END signals the end of IPL text.

IEHDASDR Example 11

In this example, two 3330 volumes are formatted.

The example follows:

```
//DASDR11  JOB
//         EXEC PGM=IEHDASDR
//SYSPRINT DD   SYSOUT=A
//DISK01   DD   UNIT=3330,DISP=OLD,VOL=(PRIVATE,,SER=(333001))
//DISK02   DD   UNIT=3330,DISP=OLD,VOL=(PRIVATE,,SER=(333002))
//SYSIN    DD   *
 FORMAT    TODD=(DISK01,DISK02),VTOC=00010,EXTENT=00010
/*
```

The control statements are discussed below:

- DISK01 and DISK02 DD statements define the 3330 devices on which the volumes (333001, 333002) are mounted.
- FORMAT defines a starting location and extent of a volume table of contents. The Record 0 of each track is rewritten.

IEHDASDR Example 12

In this example, a 3330 volume is initialized with a VTOC and volume serial number, or "quick-DASDI".

The example follows:

```
//DASDR12  JOB
//         EXEC PGM=IEHDASDR
//SYSPRINT DD   SYSOUT=A
//DISK     DD   UNIT=3330,DISP=OLD,VOL=(PRIVATE,,SER=(333000))
//SYSIN    DD   *
 ANALYZE   TODD=DISK,VTOC=00005,EXTENT=00010,NEWVOLID=333333
/*
```

The control statements are discussed below:

- DISK DD defines the 3330 device on which the volume (333000) is mounted.
- ANALYZE defines the starting location and extent of a volume table of contents. For 3330 devices, PASSES = 0 is the default so that only a "quick-DASDI" is performed.

IEHINITT Program

IEHINITT is a system utility used to place IBM volume label sets written in EBCDIC, in BCD, or in ASCII (American Standard Code for Information Interchange) on magnetic tapes mounted on one or more tape drives. (See "Introduction" for general system utility information.) Each volume label set created by the program contains:

- A standard volume label with user specified serial number and owner identification.
- An 80-byte dummy header label. For IBM standard labels, this record consists of HDR1 followed by zeros. For ANS labels, this record consists of HDR1 followed by zeros in the remaining positions, with the exception of position 54, which contains an ASCII space.
- A tapemark.

Note: When a labeled tape is subsequently used as a receiving volume: (1) the tape mark created by IEHINITT is overwritten, (2) the dummy HDR1 record created by IEHINITT is filled in with operating system data and device dependent information, (3) a HDR2 record, containing data set characteristics, is created, (4) user header labels are written if exits to user label routines are provided, (5) a tapemark is written, and (6) data is placed on the receiving volume.

Figure 44 shows an IBM standard label group after a volume is used to receive data. Refer to *OS Data Management Services Guide*, GC26-3746, for a discussion of volume labels.

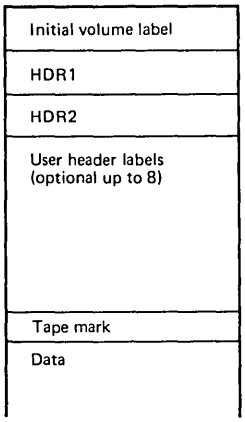


Figure 44. IBM Standard Label Group After Volume Receives Data

Placing a Standard Label Set on Magnetic Tape

IEHINITT can be used to write BCD labels on 7-track tape volumes and EBCDIC or ASCII labels on 9-track tape volumes. Any number of 7-track and/or 9-track tape volumes can be labeled in a single execution of IEHINITT.

Tape volumes are labeled in sequential order by specifying a serial number to be written on the first tape volume. The serial number 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 alphameric. If more than one volume is to be labeled, the serial numbers must be specified as six numeric characters.

The user can provide additional information, such as owner name, rewind or unload specifications, and whether the label is to be written in ASCII.

The user 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.

If any errors are encountered while attempting to label a tape, the tape is left unlabeled. IEHINITT attempts to label any tapes remaining to be processed.

For information on creating routines to write standard or nonstandard labels, refer to *OS Data Management for System Programmers*, GC28-6550.

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.

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: (1) utility program identification, (2) initial volume label information for each successfully labeled tape volume, (3) contents of utility control statements, and (4) any error messages.

IEHINITT produces a return code to indicate the results of program execution. The return codes and their meanings are:

- 00, which indicates successful completion. A message data set was created.
- 04, which indicates successful completion. No message data set was defined by the user.
- 08, which indicates that the program completed its operation, but error conditions were encountered during processing. A message data set was created.
- 12, which indicates that the program completed its operation, but error conditions were encountered during processing. No message data set was defined by the user.
- 16, which indicates that the program terminated 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.

Control

IEHINITT is controlled by job control statements and utility control statements. The job control statements are used to execute or invoke IEHINITT and to define data sets used and produced by IEHINITT. Utility control statements are used to specify applicable label information.

Job Control Statements

Table 38 shows the job control statements necessary for using IEHINITT.

Table 38. IEHINITT Job Control Statements

Statement	Use
JOB	Initiates 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 PARM information; see "PARM Information on the EXEC Statement."
SYSPRINT DD	Defines a sequential message data set.
anyname DD	Defines a tape drive to be used in a labeling operation; more than one tape drive 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 as a sequential data set outside the input stream.

The minimum region size that can be specified for IEHINITT is 14K.

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 drives to be used for the labeling operation, and deferred mounting. The name "anyname" must be identical to a name specified in a utility control statement to relate the specified drive(s) to the appropriate utility control statement.

Restrictions

- The SYSPRINT data set must have a logical record length of 121 bytes. It must consist of fixed length records with an ASA control character in the first byte of each record. Any blocking factor can be specified.
- The SYSIN data set must have a block size that is a multiple of 80. Any blocking factor can be specified.
- ANS labels can not be put on 7-track tape volumes.

PARM Information on the 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, as follows:

```
PARM = LINECNT = nn
```

If PARM is omitted, 60 lines are printed between headings.

If IEHINITT is invoked, 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 page count can be passed in a six-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 B: Invoking Utility Programs from a Problem Program."

Utility Control Statement

IEHINITT uses a utility control statement to provide control information for a labeling operation.

INITT Statement

The INITT statement provides control information for the IEHINITT program.

Any number of INITT utility control statements can be included for a given execution of the program. An identically named DD statement must exist for a utility control statement in the job step.

The format of the INITT statement is:

```
name INITT  SER = xxxxxx
            [,OWNER = 'cccccccccc[cccc]']
            [,NUMBTAPE = n]
            ,DISP = {REWIND }
                    {UNLOAD }
            [,LABTYPE = AL]
```

where:

name
specifies a name that is identical to a ddname in the name field of a DD statement defining a tape drive or drives. This name must begin in column 1.

SER = *xxxxxx*
specifies the volume serial number of the first or only tape to be labeled. The serial number cannot contain blanks, commas, apostrophes, equal signs, or special characters. A specified serial number is incremented by one for each additional tape to be labeled. (Serial number 999999 is incremented to 000000.) When processing multiple tapes, the volume serial number must be all numeric.

OWNER = 'cccccccccc[cccc]'
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 ANS volume labels. The delimiting apostrophes can be omitted if no blanks, commas, apostrophes, equal signs, or other special characters (except periods or hyphens) are included. If an apostrophe is included, it must be written as two consecutive apostrophes.

NUMBTAPE = *n*
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 **NUMBTAPE** is omitted, one tape volume is labeled.

DISP =
specifies whether a tape is to be rewound or unloaded. These values can be coded:

REWIND
specifies that a tape is to be rewound (but not unloaded) after the label has been written. If DISP = REWIND is not specified, the tape volume is rewound and unloaded.

UNLOAD
specifies that a tape is to be unloaded after the label has been written. This is the default.

LABTYPE = **AL**
specifies that an ANS volume label is to be created. If **LABTYPE** is not specified, the tape is written in EBCDIC for 9-track tape volumes and in BCD for 7-track tape volumes.

Figure 45 shows a printout of a message data set including the INITT statement and initial volume label information. In this example, one INITT statement was used to place serial numbers 001122 and 001123 on two tape volumes. VOL100112200 and VOL10011230 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.
- 0 is the Volume Security field.

No errors occurred during processing.

```
SYSTEM SUPPORT UTILITIES       IEHINITT
ALL      INITT   SER=001122,NUMBTAPE=2,OWNER='P.T.BROWN',     %C
                 DISP=REWIND
VOL10011220                     P.T.BROWN
VOL10011230                     P.T.BROWN
```

Figure 45. Printout of INITT Statement Specifications and Initial Volume Label Information

IEHINITT Examples

The following examples illustrate some of the uses of IEHINITT. Table 39 can be used as a quick reference guide to IEHINITT examples. The numbers in the "Example" column point to examples that follow.

Table 39. IEHINITT Example Directory

Operation	Comments	Example
LABEL	Three 9-track tapes are to be labeled.	1
LABEL	A 9-track tape is to be labeled.	2
LABEL	Two groups of 9-track tape volumes are to be labeled.	3
LABEL	9-track tape volumes are to be labeled. Sequence numbers are to be incremented by 10.	4
LABEL	Three 9-track tape volumes are to be labeled. An alphameric label is to be placed on a 2400 volume; numeric labels are placed on the 2400-4 volumes.	5

IEHINITT Example 1

In this example, serial numbers 001234, 001235, and 001236 are to be placed on three tape volumes; the labels are to be written in EBCDIC at 800 bits per inch. Each volume to be labeled is mounted, when it is required, on a single 9-track tape drive.

The example follows:

```
//LABEL1   JOB  09#990,BROWN,MSGLEVEL=(1,1)
//         EXEC PGM=IEHINITT
//SYSPRINT DD   SYSOUT=A
//LABEL    DD   DCB=DEN=2,UNIT=(2400,1,DEFER)
//SYSIN    DD   *
LABEL   INITT   SER=001234,NUMBTAPE=3
/*
```

IEHINITT Example 2

In this example, serial number 001001 is to be placed to one ASCII tape volume; the label is to be written at 800 bits per inch. The volume to be labeled is mounted, when it is required, on a 9-track tape drive.

The example follows:

```
//LABEL2    JOB  09#990,BROWN,MSGLEVEL=(1,1)
//          EXEC PGM=IEHINITT
//SYSPRINT  DD   SYSOUT=A
//ASCIILAB  DD   DCB=DEN=2,UNIT=(2400,1,DEFER)
//SYSIN     DD   *
ASCIILAB INITT   SER=001001,OWNER='SAM A. BROWN',LABTYPE=AL
/*
```

IEHINITT Example 3

In this example, two groups of serial numbers (001234, 001235, 001236, and 001334, 001335, 001336) are placed on six tape volumes. The labels are to be written in EBCDIC at 800 bits per inch. Each volume to be labeled is mounted, when it is required, on a single 9-track tape drive.

The example follows:

```
//LABEL3   JOB  09#990,BROWN,MSGLEVEL=(1,1)
//         EXEC PGM=IEHINITT
//SYSPRINT DD   SYSOUT=A
//LABEL    DD   DCB=DEN=2,UNIT=(2400,1,DEFER)
//SYSIN    DD   *
LABEL   INITT   SER=001234,NUMBTAPE=3
LABEL   INITT   SER=001334,NUMBTAPE=3
/*
```

IEHINITT Example 4

In this example, serial numbers 001234, 001244, 001254, 001264, 001274, etc., are to placed on eight tape volumes. The labels are to be written in EBCDIC at 800 bits per inch. Each volume to be labeled is mounted, when it is required, on one of four 9-track tape drives.

The example follows:

```
//LABEL4   JOB  09#990,BROWN,MSGLEVEL=(1,1)
//         EXEC PGM=IEHINITT
//SYSPRINT DD   SYSOUT=A
//LABEL    DD   DCB=DEN=2,UNIT=(2400,4,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
/*
```

IEHINITT Example 5

In this example, serial number TAPE1, is to be placed on a 2400 tape drive and serial numbers 001234 and 001235 are to be placed on two 2400-4 tape drives. The labels are to be written in EBCDIC at 800 and 1600 bits per inch.

The example follows:

```
//LABEL5   JOB  09#990,BROWN,MSGLEVEL=(1,1)
//         EXEC PGM=IEHINITT
//SYSPRINT DD   SYSOUT=A
//LABEL1   DD   DCB=DEN=2,UNIT=(2400,1,DEFER)
//LABEL2   DD   DCB=DEN=3,UNIT=(2400-4,1,DEFER)
//SYSIN    DD   *
LABEL1   INITT  SER=TAPE1
LABEL2   INITT  SER=001234,NUMBTAPE=2
/*
```

Note: If 2400 tape drives are not available for allocation and the system configuration includes 3400 tape drives, a 3400 tape drive may be allocated by default.

IEHIOSUP Program

IEHIOSUP is a system utility used to update TTR entries in the transfer control tables of the supervisor call library (SVC library). (See "Introduction" for general system utility information.) Because of the way SVC routines are loaded, it is necessary to update TTR entries after changing or replacing a module. IEHIOSUP automatically updates the TTR entries. IEHIOSUP must be used after:

- The SVC library is moved.
- The OPEN, CLOSE, TCLOSE, EOV, FEOV, SCRATCH, ALLOCATE, IEHATLAS, SETPRT, STOW, or any Machine Check Handler (MCH) recovery management module is changed or replaced in the SVC library.

Input and Output

IEHIOSUP uses as input an object data set (SYS1.SVCLIB) that contains the transfer control tables that are to be updated.

IEHIOSUP produces as output a message data set that contains any error messages generated during the execution of the program.

IEHIOSUP produces a return code to indicate the results of program execution. The return codes and their interpretations are:

- 00, which indicates successful completion.
- 12, which indicates an unrecoverable error. The job step is terminated.

Control

IEHIOSUP is executed or invoked with job control statements. Utility control statements are not used.

Job Control Statements

Table 40 shows the job control statements necessary for using IEHIOSUP.

Table 40. IEHIOSUP Job Control Statements

Statement	Use
JOB	Initiates the job.
EXEC	Specifies the program name (PGM = IEHIOSUP) or, if the job control statements reside in a procedure library, the procedure name.
SYSPRINT DD	Defines a sequential message data set.
SYSUT1 DD	Defines the object data set (SYS1.SVCLIB). The DSNAME, DISP, UNIT, and VOLUME parameters should be included.

The minimum region size that can be specified for IEHIOSUP is 10K.

If the SYS1.SVCLIB data set is cataloged, the UNIT and VOLUME parameters are not required on the SYSUT1 DD statement.

Restrictions

- PARM = TSO must be coded on the EXEC statement if the data set to be updated is from a TSO system.
- The block size for the SYSPRINT data set must be a multiple of 121. Any blocking factor can be specified.

IEHIOSUP Examples

The following examples illustrate some of the uses of IEHIOSUP. Table 41 can be used as a quick reference guide to IEHIOSUP examples. The numbers in the "Example" column point to examples that follow.

Table 41. IEHIOSUP Example Directory

Operation	Data Set Organization	Device	Comments	Example
UPDATE	Partitioned, Sequential	2314 Disk, system output device	SVC library is to be updated. SYS1.SVCLIB is not cataloged. System output device is a printer.	1
UPDATE	Partitioned, Sequential	3330 Disk, system output device	SVC library is to be updated. SYS1.SVCLIB is cataloged. System output device is a printer.	2
UPDATE	Partitioned, Sequential	2314 Disk, system output device	SVC library (TTRs) is to be updated on a TSO pack. SYS1.SVCLIB is cataloged. System was generated for TSO.	3

IEHIOSUP Example 1

In this example, the TTR entries in the SVC library are to be updated.

The example follows:

```
//TTRUPDTE JOB
//         EXEC PGM=IEHIOSUP
//SYSUT1   DD   DSNAME=SYS1.SVCLIB,DISP=OLD,UNIT=2314,
// VOLUME=SER=111111
//SYSPRINT DD   SYSOUT=A
//
```

The control statements are discussed below:

- SYSUT1 DD defines the object data set (the SYS1.SVCLIB data set).
- SYSPRINT DD defines the message data set.

IEHIOSUP Example 2

In this example, the TTR entries in the SVC library are to be updated.

The example follows:

```
//SVCUPDTE JOB
//         EXEC PGM=IEHIOSUP
//SYSUT1   DD   DSNAME=SYS1.SVCLIB,DISP=OLD
//SYSPRINT DD   SYSOUT=A
//
```

The control statements are discussed below:

- SYSUT1 DD defines the object data set (the SYS1.SVCLIB data set). Because the data set is cataloged, UNIT and VOLUME parameters are not required.
- SYSPRINT DD defines the message data set.

IEHIOSUP Example 3

In this example, the TTR entries in a TSO SVC library are to be updated.

The example follows:

```
//TSOSVCUP JOB
//         EXEC PGM=IEHIOSUP, PARM=TSO
//SYSUT1   DD   DSNAME=SYS1.SVCLIB,DISP=OLD
//SYSPRINT DD   SYSOUT=A
//
```

The control statements are discussed below:

- SYSUT1 DD defines the object data set (the SYSI.SVCLIB data set). Because the data set is cataloged, UNIT and VOLUME parameters are not required.
- SYSPRINT DD defines the message data set.

IEHLIST is a system utility used to list entries in a catalog, entries in the directory of one or more partitioned data sets, or entries in a volume table of contents. (See "Introduction" for general system utility information.) Any number of listings can be requested in a single job.

Listing Catalog Entries

IEHLIST lists all catalog entries that are part of the structure of a fully-qualified data set name. Figure 46 shows an index structure for which IEHLIST lists fully-qualified names A.B.D.W, A.B.D.X, A.B.E.Y, and A.B.E.Z. Because A.C.F does not represent a cataloged data set (that is, the lowest level of qualification has been deleted), it is not a fully-qualified name, and it is not listed.

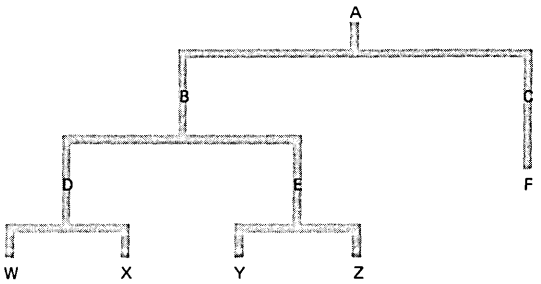


Figure 46. Index Structure—Listed by IEHLIST

Listing a Partitioned Data Set Directory

IEHLIST can list up to ten partitioned data set directories in a single application of the program. A partitioned directory is composed of variable length records blocked into 256-byte blocks. Each directory block can contain one or more entries which reflect member (and/or alias) names and other attributes of the partitioned members in edited and unedited format.

Figure 47 shows a directory block as it exists in storage.

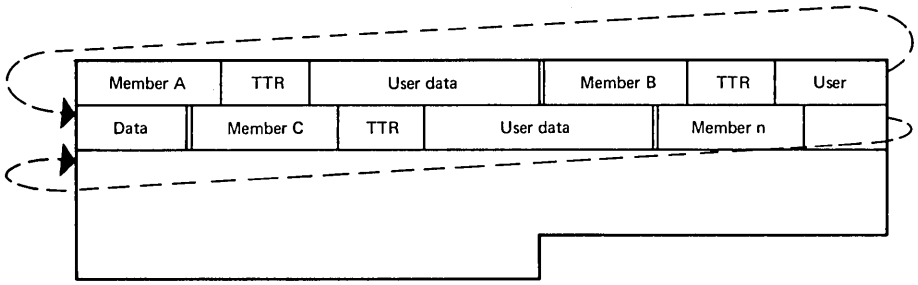


Figure 47. Sample Directory Block

Edited Format

IEHLIST optionally provides the following information, which is obtained from the applicable partitioned data set directory, when an edited format is requested:

- Member name
- Entry point
- Attributes
- Relative address of start of member
- Relative address of start of text
- Contiguous main storage requirements
- Length of first block of text
- Origin of first block of text
- System status indicators
- Other information

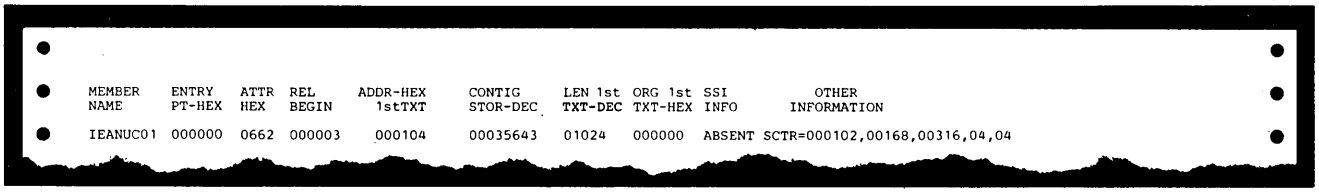
Before printing the directory entries on the first page, an index is printed explaining the asterisk (*) following a member name, the *attributes* (field 3) and *other information* (field 10). Under the ATTRIBUTE INDEX, the meaning of each attribute bit is

explained; under the OTHER INFORMATION INDEX, scatter and overlay format data is described, positionally, as it appears in the listing.

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.

Note: The FORMAT option applies only to a partitioned data set whose members have been created by the Linkage Editor (that is, the directory entries are at least 34 bytes long). If a directory entry is less than 34 bytes, a message is issued and the entry is printed in unedited format; if the entry is longer than 34 bytes, it is assumed that it is created by the Linkage Editor.

Figure 48 shows an edited entry for a partitioned member (IEANUC01). The entry is shown as it is listed by the IEHLIST program.



```
MEMBER    ENTRY   ATTR  REL     ADDR-HEX     CONTIG     LEN 1st  ORG 1st SSI          OTHER
NAME      PT-HEX  HEX   BEGIN     1stTXT     STOR-DEC   TXT-DEC  TXT-HEX INFO       INFORMATION

IEANUC01  000000  0662  000003    000104     00035643   01024    000000  ABSENT SCTR=000102,00168,00316,04,04
```

Figure 48. Edited Partitioned Directory Entry

Unedited (Dump) Format

The user may choose the unedited format. If this is the case, IEHLIST lists each member separately.

Figure 49 shows how the information in Figure 47 is listed.

Note: A listing such as that shown in Figure 47 can also be obtained by using IEBPTPCH (see "IEBPTPCH Program").

```
MEMB A    TTR     USER DATA
MEMB B    TTR     USER DATA
MEMB C    TTR     USER DATA
MEMB n    TTR     USER DATA
```

Figure 49. Sample Partitioned Directory Listing

To correctly interpret user data information, the user must know the format of the partitioned entry. The formats of directory entries are discussed in *OS System Control Blocks*, GC28-6628.

Listing a Volume Table of Contents

IEHLIST can be used to list, partially or completely, entries in a specified volume table of contents (VTOC). The program lists the contents of selected data set control blocks (DSCBs) in edited or unedited form.

Edited Format

Two edited formats are available. One 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, in detail
- Option codes
- Record formats

A VTOC consists of as many as seven types of DSCBs, as follows:

- Identifier DSCB—Format 1
- Index DSCB—Format 2
- Extension DSCB—Format 3
- VTOC DSCB—Format 4
- Free Space DSCB—Format 5
- Shared Extent DSCB—Format 6

- Free VTOC DSCB—Format 0

The first DSCB in the VTOC (and on your listing) is always the VTOC (Format 4) DSCB, which defines the VTOC data set itself and the attributes of the volume on which it resides. The VTOC DSCB is always followed by the first Free Space (Format 5) DSCB, which describes the space that is available on the volume for allocation. More than one Free Space DSCB may be required to describe the available space on a volume because each Free Space DSCB describes only 26 extents. The first Free Space DSCB can be followed, in any order, by Format 1, 2, 3, or 6 DSCBs or by other Free Space DSCBs if there are more than 26 available extents on the volume.

Identifier (Format 1) DSCBs describe a data set, its attributes, and its first three extents.

Index (Format 2) DSCBs describe the index of a data set with indexed sequential organization. The Index DSCB is always chained from the Format 1 DSCB for the data set.

Extension (Format 3) DSCBs describe the fourth through the sixteenth extents of a data set. The Extension DSCB is always chained from a Format 1 or a Format 2 DSCB.

Shared Extent (Format 6) DSCBs describe an extent of one or more contiguous cylinders being shared by two or more data sets. The first Shared Extent DSCB is chained from the Format 4 DSCB. Subsequent Shared Extent DSCBs are chained from each previous Format 6 DSCB.

Free VTOC records (called Format 0 DSCBs) indicate that space is available for another DSCB. Free VTOC DSCBs, which are 140-byte records that consist of binary zeros, are written over with Format 1, 2, 3, and 6 DSCBs when a new data set is allocated, and with Format 5 DSCBs when space is released. The Free VTOC DSCB is not listed by IEHLIST.

Figure 50 shows a sample listing of the edited format. This sample illustrates how each DSCB will appear on a listing, although in many cases the VTOC may not contain all possible types. The information is in columns, with the values or numbers appearing below each item's heading.

```
                          SYSTEMS SUPPORT UTILITIES---IEHLIST                          PAGE    1

               CONTENTS OF VTOC ON VOL EXAMPL

 FORMAT 4 DSCB NO AVAIL/MAX DSCB /MAX DIRECT  NO AVAIL NEXT ALT    FORMAT 6    LAST FMT 1     VTOC EXTENT      THIS DSCB
               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)
                 154        16          10        30    200                      5   0   5    5   0   5   9    5   0   1

 FORMAT 5 DSCB     A = NUMBER OF TRKS IN ADDITION TO FULL CYLS IN THE EXTENT
       TRK  FULL           TRK  FULL           TRK  FULL           TRK  FULL           TRK  FULL           TRK  FULL
      ADDR  CYLS   A      ADDR  CYLS   A      ADDR  CYLS   A      ADDR  CYLS   A      ADDR  CYLS   A      ADDR  CYLS   A
        17     3   3       110   189   0

               DSCB(C-H-R)   5   0   2


---------------DATA SET NAME---------------  ID  SER NO  SEQ NO  CREDT   EXPDT  NO EXT  DSORG  RECFM  OPTCD  BLKSIZE
EXAMPLE.OF.COMBINED.FORMATS.ONE.AND.TWO       1  EXAMPL      1   36699   27469     1     IS     F                100

            LRECL  KEYLEN  INITIAL ALLOC  2ND ALLOC/LAST BLK PTR(T-R-L)   USED PDS BYTES  FMT 2 OR 3(C-H-R)/DSCB(C-H-R)
              100      4     ABSTR              0                                          5   0   3     5   0   4

            EXTENTS  NO  LOW(C-H)   HIGH(C-H)
                      0     6   0      10   9

     2MIND(M-B-C-H)/3MIND(M-B-C-H)/L2MFN(C-H-R)/L3MIN(C-H-R)/CYLAD(M-B-C-H)/ADLIN(M-B-C-H)/ADHIN(M-B-C-H)/NOBYT/ NOTRK
       0 0   0   0    0 0   0   0    0   0   0    0   0   0    1 0  10   9    0 0   0   0    1 0  10   9     70      0

     LTRAD(C-H-R)/LCYAD(C-H-R)/LMSAD(C-H-R)/LPRAD(M-B-C-H-R) /NOLFV /CYLOV/ TAGDT/   PRCTR  / OVRCT/ RORG1/PTRDS(C-H-R)
       6   0   3    10   9   1    0   0   0   1 0   6   1  12    1       0             20          0      0
                              ----UNABLE TO CALCULATE EMPTY SPACE.


---------------DATA SET NAME---------------  ID  SER NO  SEQ NO  CREDT   EXPDT  NO EXT  DSORG  RECFM  OPTCD  BLKSIZE
EXAMPLE.OF.COMBINED.FORMATS.ONE.AND.THREE     1  EXAMPL      1   36699   27069    16     PS     V                3504

            LRECL  KEYLEN  INITIAL ALLOC  2ND ALLOC/LAST BLK PTR(T-R-L)   USED PDS BYTES  FMT 2 OR 3(C-H-R)/DSCB(C-H-R)
             3500            TRKS               1          15   1  1723                    5   0   6     5   0   5

            EXTENTS  NO  LOW(C-H)   HIGH(C-H)    NO  LOW(C-H)   HIGH(C-H)    NO  LOW(C-H)   HIGH(C-H)
                      0     0   1       0   1     1     0   2       0   2     2     0   3       0   3
                      3     0   4       0   4     4     0   5       0   5     5     0   6       0   6
                      6     0   7       0   7     7     0   8       0   8     8     0   9       0   9
                      9     1   0       1   0    10     1   1       1   1    11     1   2       1   2
                     12     1   3       1   3    13     1   4       1   4    14     1   5       1   5
                     15     1   6       1   6
                              ----ON THE ABOVE DATA SET,THERE ARE    0 EMPTY TRACK(S).

 THERE ARE  192 EMPTY CYLINDERS PLUS    3 EMPTY BRACKS ON THIS VOLUME
 THERE ARE  154 BLANK DSCBS IN THE VTOC ON THIS VOLUME
```

Figure 50. Sample Printout of a Volume Table of Contents

The second edited format is an abbreviated description of the data sets. It is provided by default when no format is requested specifically. It provides the following information:

- Data set name
- Creation date (dddyy)
- Expiration date (dddyy)
- Password indication
- Organization of the data set
- Extent(s)
- Volume serial number

The last line in the listing indicates how much space remains in the VTOC.

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 the user to know the various formats of applicable DSCBs. The VTOC overlay for IEHLIST listings of VTOCs in DUMP format (Order Number ZM08-0033) is useful in identifying the fields of the DSCBs.

Refer to *OS System Control Blocks*, GC28-6628 for a discussion of the various formats that data set control blocks can assume.

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(s) can be: (1) a VTOC data set, (2) a partitioned data set, or (3) a catalog data set (SYSCTLG).
- A control data set, which contains utility control statements that are used to control the functions of IEHLIST.

IEHLIST produces as output a message data set, which contains the result of the IEHLIST operations. The message data set includes the listed data and any error messages.

IEHLIST produces a return code to indicate the results of program execution. The return codes and their meanings are:

- 00, which indicates successful completion.
- 08, which indicates that an error condition caused a specified request to be ignored. Processing continues.
- 12, which indicates that a permanent input/output error occurred. The job is terminated.
- 16, which indicates that an unrecoverable error occurred while reading the data set. The job is terminated.

Control

IEHLIST is controlled by job control statements and utility control statements. The job control statements are used to execute or invoke IEHLIST and to define the data set 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 42 shows the job control statements necessary for using IEHLIST.

The minimum region size that can be specified for IEHLIST when an overlay structure is not used is 44K.

The "anyname1" DD statement can be entered:

```
//anyname1 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 deletion of the data set. This statement is arbitrarily assigned the ddname DD1 in the IEHLIST examples.

When deferred mounting is required, the "anyname2" DD statement can be entered:

```
//anyname2 DD UNIT = (xxx,,DEFER),VOLUME = (PRIVATE,...),DISP = OLD
```

See "Appendix C: DD Statements for Defining Mountable Devices" for information on defining mountable devices. This statement is arbitrarily assigned the ddname DD2 in the IEHLIST examples. Statements defining additional mountable devices are assigned ddnames DD3, DD4, etc.

Table 42. IEHLIST Job Control Statements

Statement	Use
JOB	Initiates 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. See "PARM Information on the EXEC Statement" below.
SYSPRINT DD	Defines a sequential message data set.
anyname1 DD	Defines a permanently mounted volume.
anyname2 DD	Defines a mountable device type.
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.

With the exception of the SYSIN and SYSPRINT DD statements, all DD statements in this table are used as device allocation statements, rather than as true data definition statements.

Restrictions

- The block size for the SYSPRINT data set must be a multiple of 121. The block size for the SYSIN data set must be a multiple of 80. Any blocking factor can be specified for these block sizes.
- An anyname1 DD statement must be included for each permanently mounted volume referred to in the job step. (The system residence volume is considered to be a permanently mounted volume.)
- An anyname2 DD statement must be included for each mountable device to be used in the job step.
- Because IEHLIST modifies the internal control blocks created by device allocation DD statements, IEHLIST job control statements must not include the DSNAME parameter. (All data sets are defined explicitly or implicitly by utility control statements.)
- When IEHLIST is dynamically invoked in a job step containing another program, the DD statements defining mountable devices for IEHLIST must be included in the job stream prior to DD statements defining data sets required by the other program.
- IEHLIST cannot support empty space calculations for data sets allocated in blocks when the blocksizes are approximately the same size or are larger than the tracksize. The empty block calculation gives only approximate indications of available space. When IEHLIST cannot supply an approximate number, the "Unable to Calculate" message is issued.
- IEHLIST specifications do not allow for protection 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 IEH108I) may be issued. In the case of LISTVTOC, the output produced by IEHLIST may be incorrect. When this happens, you should rerun the job.

PARM Information on the EXEC Statement

Additional information can be specified in the PARM parameter of the EXEC statement to control the number of lines printed per page. The PARM parameter can be coded:

```
PARM = 'LINECNT = xx'
```

The LINECNT parameter specifies the number of lines, xx, to be printed per page; xx is a decimal number from 01 through 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, or the default of 58 is used.

Utility Control Statements

IEHLIST is controlled by the following utility control statements:

- LISTCTLG statement, which is used to request a listing of all or part of a catalog.
- LISTPDS statement, which is used to request a directory listing of one or more partitioned data sets.
- LISTVTOC statement, which is used to request a listing of all or part of a volume table of contents.

LISTCTLG Statement

The LISTCTLG statement is used to request a listing of either the entire catalog or a specified portion of the catalog (SYSCTLG data set). The listing includes the fully-qualified name of each applicable cataloged data set and the serial number of the volume on which it resides. *Empty index levels are not listed.*

The format of the LISTCTLG statement is:

```
[label] LISTCTLG  [VOL = device = serial]
                  [,NODE = name]
```

where:

VOL = *device* = *serial*
specifies the device type and volume serial number of the control volume on which the specified portion of the catalog resides. If **VOL** is omitted, the catalog is assumed to reside on the system residence volume.

NODE = *name*
specifies a qualified name. All data set entries whose names are qualified by this name are listed. If **NODE** is omitted, all data set entries are listed.

Note: General catalog information (one-level data sets and CVOL pointers) is printed prior to the printing of an entire catalog or node.

LISTPDS Statement

The LISTPDS statement is used to request a directory listing of one or more partitioned data sets that reside on the same volume.

The format of the LISTPDS statement is:

```
[label] LISTPDS  DSNAME = (name[,name]...)
                 [,VOL = device = serial]
                 {,DUMP   }
                 {,FORMAT}
```

where:

DSNAME = (*name*[,*name*]...)
specifies the fully-qualified names of the partitioned data sets whose directories are to be listed. A maximum of ten names is allowed. If the list consists of a single name, the parentheses can be deleted.

VOL = *device* = *serial*
specifies the device type and volume serial number of the volume on which the partitioned data sets reside. If **VOL** is omitted, the data sets are assumed to reside on the system residence volume.

DUMP
specifies that the listing is to be in unedited, hexadecimal form.

FORMAT
specifies that the listing is to be edited for each directory entry.

Before printing the directory entries on the first page, an index is printed explaining the *attributes* (field 3) and *other information* (field 10). ATTRIBUTE INDEX explains each attribute bit; OTHER INFORMATION INDEX explains scatter and overlay format data as it appears in the listing.

Note: The LISTPDS statement 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.

LISTVTOC Statement

The LISTVTOC statement is used to request a partial or complete listing of the entries in a specified volume table of contents.

The format of the LISTVTOC statement is:

```
[label] LISTVTOC  {DUMP   }
                  {FORMAT}
                  [,DATE = dddyy]
                  [,VOL = device = serial]
                  [,DSNAME = (name[,name]...)
```

where:

DUMP
specifies that the listing is to be in unedited, hexadecimal form. If both **DUMP** and **FORMAT** are omitted, an abbreviated edited format is generated by default.

FORMAT
specifies that a comprehensive edited listing is to be generated. If both **FORMAT** and **DUMP** are omitted, an abbreviated edited format is generated by default.

DATE = *dddyy*
specifies that each entry that expires before this date is to be flagged with an asterisk (*) in the listing. This parameter applies only to the abbreviated edited format. The date is represented by *ddd*, the day of the year, and *yy*, the last two digits of the year. If **DATE** is omitted, no asterisks appear in the listing.

VOL = *device* = *serial*
specifies the device type and volume serial number of the volume whose table of contents is to be listed. If **VOL** is omitted, the system residence volume is assumed.

DSNAME = (*name*[,*name*]...)
specifies the fully-qualified names of the data sets whose entries are to be listed. A maximum of ten names is allowed. If **DSNAME** is omitted, the entire volume table of contents is listed.

IEHLIST Examples

The following examples illustrate some of the uses of IEHLIST. Table 43 can be used as a quick reference guide to IEHLIST examples. The numbers in the "Example" column point to examples that follow.

Table 43. IEHLIST Example Directory

Operation	Device	Comments	Example
LIST	2314 Disk, system output device	Source catalog is to be listed on the system output device.	1
LIST	2314 Disk, system residence device, system output device	Three catalogs and part of a fourth are to be listed on the system output device.	2
LIST	2314 or 2319 Disk,[1] 3330 Disk system output device	Three partitioned directories are to be listed on the system output device.	3
LIST	2314 Disk, system output device	Volume table of contents is to be listed in edited form; selected data set control blocks are listed in unedited form.	4

[1] Note that the 2319 disk is functionally equivalent to the 2314 disk; to use the 2319, specify 2314 in the control statement.

Note: In the IEHLIST examples, the EXEC statement and the SYSPRINT DD statement can be replaced with the following job control statement:

```
// EXEC PROC = LIST
```

The EXEC statement invokes the following IBM-supplied cataloged procedure:

```
//LIST EXEC    PGM = IEHLIST,REGION = 44K
//DDSRV DD     VOLUME = REF = SYS1.SVCLIB,DISP = OLD
//SYSPRINT DD  SYSOUT = A
```

IEHLIST Example 1

In this example a catalog residing on a 2314 volume (231400) is to be listed.

The example follows:

```
//LISTCAT  JOB  09#550,BLUE
//         EXEC PGM=IEHLIST
//SYSPRINT DD   SYSOUT=A
//DD2      DD   UNIT=2314,VOLUME=SER=231400,DISP=OLD
//SYSIN    DD   *
     LISTCTLG   VOL=2314=231400
/*
```

The control statements are discussed below:

- DD2 DD defines a mountable device on which the volume containing the source catalog is mounted.
- SYSIN DD defines the control data set, which follows in the input stream.
- LISTCTLG defines the source volume and specifies the list operation.

Note: The data set name of the catalog data set is SYSCTLG.

IEHLIST Example 2

In this example a catalog residing on the system residence volume, two catalogs residing on 2314 volumes, and a portion of a catalog residing on a 2314 volume, are to be listed.

The example follows:

```
//LISTCATS JOB  09#550,BLUE
//         EXEC PGM=IEHLIST
//SYSPRINT DD   SYSOUT=A
//DD1      DD   UNIT=3330,VOLUME=SER=111111,DISP=OLD
//DD2      DD   UNIT=(2314,,DEFER),DISP=OLD,
// VOLUME=(PRIVATE,,SER=(231400))
//SYSIN    DD   *
     LISTCTLG
     LISTCTLG   VOL=2314=231400
     LISTCTLG   VOL=2314=231401
     LISTCTLG   VOL=2314=231402,NODE=A.B.C
/*
```

The control statements are discussed below:

- DD1 DD defines a system residence device. (The first catalog to be listed resides on the system residence volume.)
- DD2 DD defines a mountable device on which each 2314 volume is mounted as it is required by the program.
- SYSIN DD defines the control data set, which follows in the input stream.
- The first LISTCTLG statement indicates that the catalog residing on the system control volume is to be listed.
- The second and third LISTCTLG statements identify two 2314 disk volumes containing catalogs to be listed.
- The fourth LISTCTLG statement identifies a 2314 volume containing a catalog that is to be partially copied. All data set entries whose beginning qualifiers are "A.B.C" are copied.

IEHLIST Example 3

In this example, a partitioned directory existing on the system residence volume is to be listed. In addition, two partitioned directories existing on a 2314 or 2319 volume are to be listed. Note that the 2319 disk is functionally equivalent to the 2314 disk; to use the 2319 specify 2314 in the control statement.

The example follows:

```
//LISTPDIR JOB  09#550,BLUE
//         EXEC PGM=IEHLIST
//SYSPRINT DD   SYSOUT=A
//DD1      DD   UNIT=3330,VOLUME=SER=111111,DISP=OLD
//DD2      DD   UNIT=2314,VOLUME=SER=231400,DISP=OLD
//SYSIN    DD   *
      LISTPDS   DSNAME=PARSET1
      LISTPDS   DSNAME=(PART1,PART2),VOL=2314=231400
/*
```

The control statements are discussed below:

- DD1 DD defines the system residence device.
- DD2 DD defines a mountable device on which a 2314 volume (231400) is to be mounted.
- SYSIN DD defines the control data set, which follows in the input stream.
- The first LISTPDS statement indicates that the partitioned data set directory belonging to data set PARSET1 is to be listed. This data set exists on the system residence volume.
- The second LISTPDS statement indicates that partitioned directories belonging to data sets PART1 and PART2 are to be listed. These data sets exist on a 2314 volume (231400).

IEHLIST Example 4

In this example, a volume table of contents in edited form, is to be listed. The edited listing is supplemented by an unedited listing of selected data set control blocks.

The example follows:

```
//LISTVTOC JOB  09#550,BLUE
//         EXEC PGM=IEHLIST
//SYSPRINT DD   SYSOUT=A
//DD2      DD   UNIT=2314,VOLUME=SER=231400,DISP=OLD
//SYSIN    DD   *
     LISTVTOC   FORMAT,VOL=2314=231400
     LISTVTOC   DUMP,VOL=2314=231400,DSNAME=(SET1,SET2,SET3)
/*
```

The control statements are discussed below:

- DD2 DD defines a mountable device on which the volume containing the specified volume table of contents is to be mounted.
- 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 2314 volume is to be listed in edited form.
- 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.

IEHMOVE Program

IEHMOVE is a system utility used to move or copy logical collections of operating system data. (See "Introduction" for general system utility information.)

IEHMOVE can be used to move or copy:

- A data set residing on from one to five volumes.
- A group of cataloged data sets.
- A catalog, or portions of a catalog.
- A volume of data sets.

The scope of a basic move or copy operation can be enlarged by:

- Merging members from two or more partitioned data sets.
- Including or excluding selected members.
- Renaming moved or copied members.
- Replacing selected members.
- Including or excluding data sets from a move or copy operation.

If, for some reason, IEHMOVE is unable to successfully move or copy specified data, an attempt is made 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 to a device that will support the data in its true form, the data is automatically reconstructed. For example, if the user attempts to move a partitioned data set to a tape volume, the data is unloaded onto that volume. The user can re-create the data set simply by moving the unloaded data set to a direct access volume.

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 source volume and the expiration data 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.

Space for a new data set on a direct access device can be allocated by the user in a previous job step or by IEHMOVE in an IEHMOVE job step.

Space can be allocated by the user in a previous job step by using a DD statement that specifies the amount of space required. If, however, a data set that contains location-dependent information (for instance, a data set with the *unmovable* attribute) is being moved or copied, the user should allocate space for the receiving data set using absolute track allocation. This ensures that the data is placed in the same relative location on the receiving volume as on the source volume (provided the device types of source and receiving volumes are the same). Unmovable data is moved or copied even when space allocation was not made before the IEHMOVE job step. No assumptions, however, can be made as to whether the location-dependent information is correct. (The IEHIOSUP program can be used to update the new version of a SYS1.SVCLIB after it has been moved or copied into a space which was not preallocated.)

Space for a new data set cannot be allocated by the user under the following circumstances:

- When the organization of the data set to be moved or copied is direct and the data set is not to be unloaded, IEHMOVE cannot determine if the new data set is empty.
- When a partitioned data set is being moved or copied as part of a move or copy volume operation and the data set is not to be unloaded. If the user does preallocate a partitioned data set in this case, no merging takes place.

If IEHMOVE performs the space allocation for the 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 direct access device, 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 all 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 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 attempts to allocate space on the assumption that all blocks in the data set are of the maximum length. This can cause the following situations when moving or copying between unlike devices:

- When moving or copying from a device with a relatively small block overhead to a device with a larger block overhead, too much space is allocated.
- When moving or copying from a device with a large block overhead to a device with a smaller block overhead, too little space might be allocated. In this case, the user should preallocate the data set with enough primary or secondary space.

If the data set to be moved or copied is on neither a direct access device nor an unloaded data set (for instance, a sequential data set on a tape volume), IEHMOVE uses a default allocation of approximately 72,500 bytes of primary space plus 36,250 bytes of secondary space. Any default space that is unused after the data set has been moved or copied is released.

Note: Data sets with direct organization and variable record format always have the same amount of direct access space allocated by IEHMOVE. This practice preserves any relative track addressing system that might exist within the data sets.

A move or copy operation results in: (1) a moved or copied data set, (2) no action, or (3) an unloaded version of the source data set. These results depend upon the compatibility of the source and receiving volumes with respect to:

- Size of the volumes.
- Data set organization (sequential, partitioned, or direct access).
- Movability of the source data set.
- Allocation of space on the receiving volume.

Two volumes are compatible with respect to size if (1) the source record size does not exceed the receiving track size or (2) the receiving volume supports the track overflow feature and the output is to be written with track overflow. (Refer to "Job Control Statements" for notes on the track overflow feature.) When using direct access organization, two volumes are compatible with respect to size if the source track capacity does not exceed the receiving track capacity. Direct access data sets moved or copied to a smaller device type or tape are unloaded. If the user wishes to load an unloaded direct access data set, it must be loaded to the same device type from which it originally was unloaded.

If the UNLOAD keyword is specified, a data set can be unloaded even though the receiving volume and data set organization would allow for a normal copy or move operation. (See "Utility Control Statements" in this chapter.)

Table 44 shows the results of move and copy operations when the receiving volume is a direct access volume that is compatible in size with the source volume. The organization of the source data set set is shown along with the characteristics of the receiving volume.

Table 45 shows the results of move and copy operations when the receiving volume is a direct access 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 46 shows the results of move and copy operations when the receiving volume is not a direct access volume. The organization of the source data set is shown along with the characteristics of the receiving volume.

Table 44. Move and Copy Operations—Direct Access Receiving Volume with Size Compatible with Source Volume

Receiving Volume Characteristics	*Sequential*	*Partitioned*	*Direct Access*
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 45. Move and Copy Operations—Direct Access Receiving Volume with Size Incompatible with Source Volume

Receiving Volume Characteristics	*Sequential*	*Partitioned*	*Direct Accessf*
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 46. Move and Copy Operations—Nondirect Access Receiving Volume

Receiving Volume Characteristics	Sequential	Partitioned	Direct Access
Movable data	moved or copied	unloaded	unloaded
Unmovable data	unloaded	unloaded	no action

IEHMOVE does not scratch data sets if the expiration date has not occurred. IEHPROGM can be used to scratch these data sets (see the chapter "IEHPROGM Program").

Note: When space is previously allocated for a data set that is to be unloaded, the data set should be sequentially organized, as is the case for unloaded data sets.

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 direct access 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, the user need move only the 105th member into the receiving partitioned data set.

When moving or copying a data set group or a volume containing password-protected data sets, the user 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. IEHMOVE does not take exits to a user's label processing routines.

When moving or copying a password-protected data set, the reproduced data set is protected in the same way as the source data set, unless a different protection level is established through preallocation.

It is your responsibility to update your system's PASSWORD data set. This can be done with the RENAME function of IEHMOVE.

Note: If a data set that has only user trailer labels is to be moved from a tape volume to a direct access volume, space must be previously allocated on the direct access volume to ensure that a track is reserved to receive the user labels.

Reblocking

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.

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), the user 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.

Moving or Copying a Data Set

IEHMOVE can be used to move or copy sequential, partitioned, and direct access data sets, as follows:

- A sequential data set can be: (1) moved from one direct access or nondirect access volume to another (or to the same volume provided that it is a direct access volume) or (2) copied from one direct access or nondirect access volume to another (or to the same volume provided that the data set name is changed and the receiving volume is a direct access volume).
- A partitioned data set can be: (1) moved from one direct access volume to another (or to the same volume) or (2) copied from one direct access volume to another (or to the same volume provided that the data set name is changed).
- A direct access data set can be moved or copied from one direct access volume to another provided that the receiving device type is the same device type or a larger device type and that the record size does not exceed 32K.

In addition, 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. 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 a utility control statement. (You must 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, follow the same procedure as for the *list* field of the TO = *device* = *list* parameter in the utility control statement.

For MOVE/COPY operations on a data set that resides on multiple volumes, all volume serial numbers involved should be given in the *list* field of the FROM/TO = *device* = *list* parameter of the utility control statement. The volume serial numbers must also appear in the *list* field of the VOLUME = SER = *list* parameter of the DD statement indicating where the text is to be placed. On that same DD statement, especially when operating on a BDAM data set, you should code the UNIT parameter as follows: UNIT = (*devicetype*,P).

A data set with the unmovable attribute can be moved or copied from one direct access volume to another or to the same volume provided that space has been previously allocated on the receiving volume. Change the name of a data set to move or copy it to the same volume. SVCLIB can be moved or copied to another location on the system residence volume, provided that space is available and that space has been previously allocated on that volume. IEHPROGM must be used immediately after such a move operation to rename the moved version SYS1.SVCLIB. After such a copy operation, IEHPROGM must be used to scratch the old version and to rename the copied version. In either case, IEHIOSUP must be used immediately after the IEHPROGM step to update the new version of SVCLIB.

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.

Table 47 shows basic and optional move and copy operations for sequential and partitioned data sets.

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.

Table 47. Moving and Copying Sequential and Partitioned Data Sets

Operation	Basic Actions	Optional Actions
Move Sequential	Move the data set. For direct access, scratch the source data. For cataloged to refer to the moved data set.	Prevent automatic cataloging of the moved data set. Rename the moved data set.
Move Partitioned	Move the data set. For direct access, scratch the source data. For cataloged data sets, update the catalog to refer to the moved data set.	Prevent automatic cataloging of the moved data set. Rename the moved data set. Re-allocate directory space. (Not possible if the space previously allocated is partially used.) Perform a merge operation using members from two or more data sets. Move only selected members. Replace members. Unload the data set.
Copy	Copy the data set. The source data set is not scratched. The catalog is not updated to refer to the copied data set.	Uncatalog the source data set. Catalog the copied data set on the receiving volume. Rename the copied data set.
Copy Partitioned	Copy the data set. The source data is not scratched. The catalog is not updated to refer to the copied data set.	Uncatalog the source data set. Catalog the copied data set on the receiving volume. Rename the copied data set. Re-allocate directory space. (Not possible if the space previously allocated is partially used.) Perform a merge operation using members from two or more data sets. Copy only selected members. Replace members. Unload the data set.

Figure 51 shows a copied partitioned data set. Note that the members are copied in the order in which they appear in the partitioned directory. IEBCOPY can be used to copy data sets whose members are not to be collated.

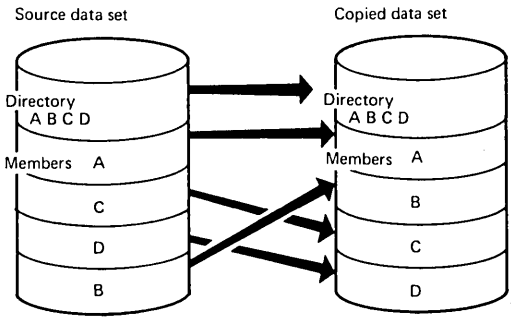


Figure 51. Partitioned Data Set Before and After an IEHMOVE Copy Operation

Members that are merged into an existing data set are placed, in collating sequence, after the last member in the existing data set.

Figure 52 shows members from one data set merged into an existing data set. Note that members A, C, and G from the existing data set are copied to the receiving volume before members B and F are copied from the source data set. Members B and F are copied in collating sequence.

Figure 53 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. Note that

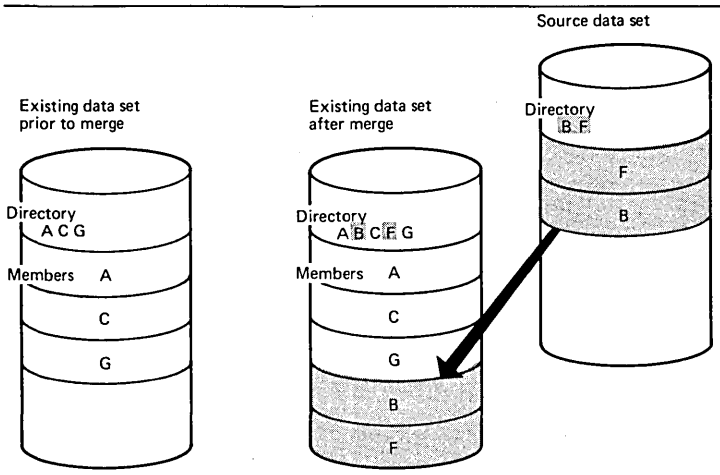


Figure 52. Merging Two Data Sets Using IEHMOVE

members A, C, and G are copied from the existing data set before any members are copied from the source data sets. Members F, B, D, and E from the source data sets are copied in collating sequence.

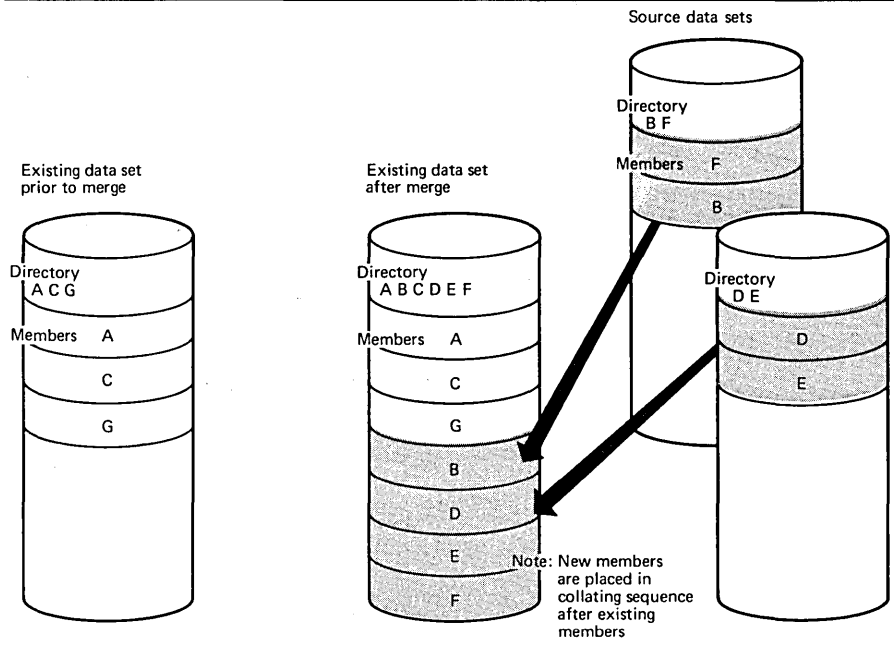


Figure 53. Merging Three Data Sets Using IEHMOVE

Moving or Copying a Group of Cataloged Data Sets

IEHMOVE can be used to move or copy a group of data sets that are cataloged on the same volume 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.

Additional data sets not belonging to the specified data set group can be included in the move or copy operation; data sets belonging to the group can be excluded.

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.

IEHLIST can be used to determine the structure of the catalog.

Table 48 shows basic and optional move and copy operations for a group of cataloged data sets.

Table 48. Moving and Copying a Group of Cataloged Data Sets

Operation	Basic Actions	Optional Actions
Move group of cataloged data sets	Move the data set group (excluding password-protected data sets) to the specified volumes. Scratch the source data sets (direct access only). The catalog is not updated to refer to moved data sets. Merging is not done.	Prevent updating of the catalog. Include password-protected data sets in the operation. Include additional data sets in the operation. Exclude data sets from the operation. Unload data sets.
Copy group of cataloged data sets	Copy the data set group (excluding password-protected data sets.). Source data sets are not scratched. The catalog is not updated to refer to copied data sets. Merging is not done.	Include password-protected data sets in the operation. Uncatalog the source data sets. Catalog the copied data sets on the receiving volumes. Include additional data sets in the operation. Unload a data set or sets.

Moving or Copying a Catalog

IEHMOVE can be used to move or copy a catalog or portions of a catalog without copying the data sets represented by the cataloged entries. The SYSCTLG (system catalog) data set need not be defined on the receiving volume before the operation. If, however, SYSCTLG was defined before the operation, the data set organization must not have been specified in the DCB field. Moved or copied entries are merged with any existing entries on the receiving volume. Note that the receiving volume must be a direct access volume unless the catalog is to be unloaded.

Table 49 shows basic and optional move and copy coperations for the catalog.

Table 49. Moving and Copying the Catalog

Operation	Basic Actions	Optional Actions
Move catalog	Move entries from the catalog to the specified direct access volume. Scratch the source volume.	Exclude selected entries from operation. Move an unloaded version of the catalog. Unload the catalog to the magnetic tape volume.
Copy catalog	Copy entries from the catalog to the specified direct access device. The source catalog is not scratched.	Exclude selected entries from the operation. Copy an unloaded version of the catalog. Unload the catalog to a tape volume.

Moving or Copying a Volume of Data Sets

IEHMOVE can be used to move or copy the data sets of an entire direct access 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. IEHPROGM can be used to uncatalog all of the cataloged data sets and recatalog them according to their new location.

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, 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, the user has 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 or copied has a name identical to a data set name on the receiving volume, the data set is not moved, copied, or merged onto the receiving volume.

Table 50 shows basic and optional move and copy operations for a volume of data sets.

Table 50. 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 direct access volumes. Scratch the source data sets for direct access volumes. The catalog is not updated.	Include password-protected data sets in the operation. Move to a tape volume.
COPY a volume of data sets	Copy all data sets not protected by a password to the specified direct access volume. The source data sets are not scratched.	Include password-protected data sets in the operation. Catalog all copied data sets on the receiving volume (direct access only). Copy to a tape volume.

Moving or Copying Direct Data Sets with Variable Spanned Records

IEHMOVE can be used to move or copy direct data sets with variable spanned records from one direct access volume to a compatible direct access volume, provided that the record size does not exceed 32K.

Because a direct access 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 direct access data sets occupy the same relative number of tracks that they occupied on the source device.

If a direct 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 a user wishes to create a direct 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 direct data sets should be at least two tracks. (See the "WRITE SZ" macro instruction in *OS Supervisor & Data Management Macro Instructions*, GC28-6647.)

Input and Output

IEHMOVE uses the following input:

- One or more data sets, which contain the data to be moved, copied, or merged into a 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 produces the following output:

- An output data set, which is the result of the move, copy, or merge operation.
- A message data set, which contains informational messages (for example, the names of moved or copied data sets) and error messages, if applicable.

IEHMOVE produces a return code to indicate the results of program execution. The return codes and their meanings are:

- 00, which indicates successful completion.
- 04, which indicates that a specified function was not completely successful. Processing continues.
- 08, which indicates a condition from which recovery is possible. Processing continues.
- 12, which indicates an unrecoverable error. The job step is terminated.

Control

IEHMOVE is controlled by job control statements and utility control statements. The job control statements are used to execute or invoke the program, define the devices and volumes used and produced by IEHMOVE, and prevent data sets from being deleted inadvertently.

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 51 shows the job control statements necessary for using IEHMOVE.

Table 51. IEHMOVE Job Control Statements

Statement	Use
JOB	Initiates 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 include optional PARM information; see "PARM Information on the EXEC Statement" below.
SYSPRINT DD	Defines a sequential message data set. 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 a work data set required by IEHMOVE is placed.
anyname1 DD	Defines a permanently mounted volume. (The system residence volume is considered to be a permanently mounted volume.)
anyname2 DD	Defines a mountable device type.
tape DD	Defines a tape volume to be used when moving or copying from or to a 7-track tape volume, a 9-track tape volume not having standard labels, or a 1600 bits per inch, 9-track tape volume on a single-density drive, or when copying to an 800 bits per inch tape on a dual-density drive.
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 an unblocked sequential data set or as a member of a procedure library.

The minimum region size that can be specified for the execution of IEHMOVE is 16K + b, where b is the largest block size in the job step rounded to the next higher 2K.

The SYSUT1 DD statement can be coded:

```
//SYSUT1 DD UNIT=xxxx,VOLUME=SER=xxxxxx,DISP=OLD
```

At least 80 contiguous tracks must be available for work space on the volume defined by the SYSUT1 DD statement. (This figure is based on a 2311 being the work volume. If a direct access device other than a 2311 is used, an equivalent amount of space must be available.)

When POWER = 2 is specified in the PARM field of the EXEC parameter, the number of contiguous tracks that must be available for work space on the volume is doubled; see "PARM Information on the EXEC Statement" below.

The anyname1 DD statement can be coded:

```
//anyname1 DD UNIT=xxxx,VOLUME=SER=xxxxxx,DISP=OLD
```

In the anyname1 DD statement, 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 anyname1 DD statement is arbitrarily assigned the ddname DD1 in the IEHMOVE examples.

The anyname2 DD statement can be coded:

```
//anyname2 DD UNIT=xxxx,VOLUME=SER=xxxxxx,DISP=OLD
```

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),VOLUME=(PRIVATE,...),
//             DISP=(...,KEEP)
```

See "Appendix C: DD Statements for Defining Mountable Devices" for information on defining mountable devices. The anyname2 DD statement is arbitrarily assigned the ddname DD2 in the IEHMOVE examples. DD statements defining additional mountable device types are assigned names DD3, DD4, etc.

The tape DD statement can be coded:

```
//tape DD  DSNAME = xxxxxxxx,UNIT = xxxx,VOLUME = SER = xxxxxx,
//         DISP = (...,KEEP),LABEL = (...,...),DCB = (TRTCH = C,DEN = x)
```

when 7-track tape is to be used. 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 onto a magnetic tape volume is automatically recorded in the HDR 1 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.

To define a sequence number for a data set on a tape volume, or to specify a specific device (for example, unit address 190), you must use a utility control statement instead of 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 a 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 a utility control statement. To copy to a unit record or unlabeled tape volume, specify any standard name or number in a 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).

With the exception of the SYSIN and SYSPRINT DD statements, all DD statements shown in Table 51 are used as device allocation statements, rather than as true data definition statements. Because IEHMOVE modifies the internal control blocks created by device allocation DD statements, these statements must not include the DSNAME parameter. (All data sets are defined explicitly or implicitly by utility control statements.)

A merge operation requires that one DD statement defining a mountable device be present for each source volume containing data to be included in the merge operation.

Prior space allocations can be made by specifying a dummy execution of IEHPROGM before the execution of IEHMOVE.

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. (For a discussion of user data TTRNs, refer to *OS Data Management Services Guide*, GC26-3746.)

Restrictions

- The block size for the SYSPRINT data set must be a multiple of 121. The block size for the SYSIN data set must be a multiple of 80. Any blocking factor can be specified for these block sizes.
- One anyname1 DD statement must be included for each permanently mounted volume referred to in the job step.
- One anyname2 DD statement must be included for each mountable device to be used in the job step.
- When IEHMOVE is dynamically invoked in a job step containing another program, the DD statements defining mountable devices for IEHMOVE must be included in the job stream prior to DD statements defining data sets required by the other program.

PARM Information on the EXEC Statement

The EXEC statement for IEHMOVE can contain PARM information that is used by the program to allocate additional work space and/or control line density on output listings. The EXEC statement can be coded, as follows:

```
// EXEC PGM = IEHMOVE[,PARM = {'POWER = n'                }
                              {'POWER = n,LINECNT = xx' }
                              {'LINECNT = xx'           }
```

The POWER = n parameter is used to request that the normal amount of space allocated for work areas be increased n times. 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. If, in the EXEC statement,

POWER = 3 is specified, the work space requirement is three times the basic requirements, etc.

The POWER parameter is used when 750 or more members are being moved or copied. The progression for the value of n is:

- POWER = 2 when 750 to 1,500 members are to be moved or copied.
- POWER = 3 when 1,501 to 2,250 members are to be moved or copied.
- POWER = 4 when 2,251 to 3,000 members are to be moved or copied.

If POWER = 2, the work space requirement on the SYSUT1 volume is two times the basic requirement; if POWER = 3, work space requirement is three times the basic requirement, etc. For example, if POWER = 2, 80 tracks on a 2314 must be available.

When moving or copying a catalog, the value of the POWER parameter can be calculated, as follows:

POWER = (10D + V + 20G)/4000

where D is the total number of data sets, aliases, and generation data set entries (which is the number of data set names printed by IEHLIST when LISTCTLG is specified); V is the total number of volumes used by these data sets (which is the number of lines printed by IEHLIST when LISTCTLG is specified); and G is the number of generated data sets. The progression of the value of n when moving or copying a catalog is:

- POWER = 2 when 350 to 700 data sets reside on the catalog.
- POWER = 3 when 701 to 1050 data sets reside on the catalog.
- POWER = 4 when 1051 to 1400 data sets reside on the catalog.

The LINECNT = xx parameter specifies the number of lines per page in the listing of the SYSPRINT data set; xx is a two-digit number in the range 04 through 99.

Job Control Language for the Track Overflow Feature

A data set containing track overflow records can be moved or copied if the source volume and the receiving volume are mounted on direct access devices that support the track overflow feature. (For BDAM data sets, the source and receiving devices must be the same device type.)

A data set that was written without track overflow can be moved or copied with or without track overflow or vice versa if the following conditions are met:

- Space was allocated for the data set prior to the request for a move or copy operation.
- The DD statement used for that allocation included the subparameter to specify the changed track overflow value and all other desired values. (The RECFM specifications assigned when the data set was originally created are overridden by the RECFM subparameter in this DD statement.)

If space has not been allocated, or if RECFM was not specified when space was allocated, the data set is moved or copied in accordance with RECFM specifications that were made when the data set was originally created.

The track overflow attribute is not retained for a sequential data set that is moved or copied to a device other than a direct access device.

Utility Control Statements

IEHMOVE is controlled by the following utility control statements:

- MOVE DSNAME statement, which is used to move a data set.
- COPY DSNAME statement, which is used to copy a data set.
- MOVE DSGROUP statement, which is used to move a group of cataloged data sets.
- COPY DSGROUP statement, which is used to copy a group of cataloged data sets.
- MOVE PDS statement, which is used to move a partitioned data set.
- COPY PDS statement, which is used to copy a partitioned data set.
- MOVE CATALOG, which is used to move cataloged entries.
- COPY CATALOG statement, which is used to copy cataloged entries.
- MOVE VOLUME statement, which is used to move a volume of data sets.
- COPY VOLUME statement, which is used to copy 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 DSGROUP, COPY DSGROUP, 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 DSGROUP, COPY DSGROUP, MOVE PDS, COPY PDS, MOVE CATALOG, or COPY CATALOG statement to exclude data from the 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.

MOVE DSNAME Statement

The MOVE DSNAME statement is used to move a data set. The source data set is scratched.

The format of the MOVE DSNAME statement is:

```
[label] MOVE  DSNAME = name
              ,TO = device = list
              [,FROM = device = list  ]
              [,CVOL = device = serial]
              [,UNCATLG]
              [,RENAME = name]
              [,FROMDD = ddname]
              [,TODD = ddname]
              [,UNLOAD]
```

where:

DSNAME = *name*
specifies the fully-qualified name of the data set to be moved.

TO = *device* = *list*
specifies the volume or volumes to which the data set is to be moved.

FROM = *device* = *list*
specifies the volume or volumes on which the data set currently resides, if it is not cataloged. If the data set is cataloged, **FROM** need not be written.

CVOL = *device* = *serial*
specifies the device type and volume serial number of the volume on which the catalog search for the data set is to begin. If neither **CVOL** nor **FROM** is written, the data set is assumed to be cataloged on the system residence volume.

UNCATLG
specifies that the catalog entry pertaining to the data set is to be removed. This parameter should be used only if the source data set is cataloged. **UNCATLG** is ignored if the volume is identified by **FROM**.

RENAME = *name*
specifies that the data set is to be renamed, and indicates the new name.

FROMDD = *ddname*
specifies the name of a DD statement from which DCB and LABEL information are obtained. This parameter is valid for data sets residing on magnetic tape volumes. See Table 51 and the tape DD statement under "Job Control Statements" for additional information.

TODD = *ddname*
specifies the name of a DD statement from which DCB and LABEL information are obtained. This parameter, which is valid for data sets to be moved to tape volumes, describes the mode and label of the volume where the moved data set is to reside. RECFM, LRECL, and BLKSIZE information is ignored.

UNLOAD
specifies that the data set is to be unloaded to the receiving volume(s).

If the data set is cataloged, the catalog is automatically updated unless **UNCATLG** is specified.

The **FROMDD** and **TODD** parameters can be omitted for 800 bits per inch, 9-track tape with standard labels on single-density drives or for 1600 bits per inch, 9-track tape with standard labels on dual-density drives.

COPY DSNAME Statement

The COPY DSNAME statement is used to copy a data set.

The format of the COPY DSNAME statement is:

```
[label] COPY  DSNAME = name
              ,TO = device = list
              [,FROM = device = list  ]
              [,CVOL = device = serial]
              [,UNCATLG]
              [,CATLG]
              [,RENAME = name]
              [,FROMDD = ddname]
              [,TODD = ddname]
              [,UNLOAD]
```

where:

DSNAME = *name*
specifies the fully-qualified name of the data set to be copied.

TO = *device* = *list*
specifies the volume or volumes on which the data set is to be copied.

FROM = *device* = *list*
specifies the volume or volumes on which the data set currently resides, if it is not cataloged. If the data set is cataloged, **FROM** should not be written.

CVOL = *device* = *serial*
specifies the device type and volume serial number of the volume on which the catalog search for the data set is to begin. If neither **CVOL** nor **FROM** is written, the data set is assumed to be cataloged on the system residence volume.

UNCATLG
specifies that the catalog entry pertaining to the source data set is to be removed. This parameter may only be used if the source data set is cataloged. **UNCATLG** is ignored if the volume is identified by **FROM**.

CATLG
specifies that the copied data set is to be cataloged on its receiving volume if it is a direct access volume. If a catalog does not exist on the receiving volume, a new catalog is created.

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 are obtained. DCB attributes for unloaded data sets are always RECFM = FB, LRECL = 80, and BLKSIZE = 800. This parameter is valid for data sets residing on tape volumes. See Table 51 and the tape DD statement under "Job Control Statements" for additional information.

TODD = *ddname*
specifies the name of a DD statement from which DCB and LABEL information are obtained. This parameter, which is valid for data sets to be copies onto magnetic tape volumes, describes the mode and label of the magnetic tape where the copied data set is to reside. RECFM, LRECL, and BLKSIZE information is ignored.

UNLOAD
specifies that the data set is to be unloaded to the receiving volume(s).

Note: The source data set, if cataloged, remains cataloged unless **UNCATLG** is specified.

The **FROMDD** and **TODD** parameters can be omitted for 800 bits per inch, 9-track tape with standard labels on single-density drives or for 1600 bits per inch, 9-track tape with standard labels on dual-density drives.

MOVE DSGROUP Statement

The MOVE DSGROUP statement is used to move groups of data sets that are cataloged on the same volume and whose names are partially qualified by one or more identical names. Source data sets are scratched. Data set groups to be moved must reside on direct access volumes.

INCLUDE and EXCLUDE statements, discussed later in this chapter, can be used to add to or delete data sets from the group.

The format of the MOVE DSGROUP statement is:

```
[label] MOVE  DSGROUP[ = name]
              ,TO = device = list
              [,CVOL = device = serial]
              [,PASSWORD]
              [,UNCATLG]
              [,TODD = ddname]
              [,UNLOAD]
```

where:

DSGROUP [= *name*]
specifies the cataloged data sets to be moved. If *name*, which is a qualified name, is not coded, all data sets cataloged on the specified volume are to be moved. If *name* is coded, all cataloged data sets whose names are qualified by this name are moved. If the name is a fully-qualified data set name, only that data set is moved.

TO = *device* = *list*
specifies the volume or volumes to which the specified group of data sets is to be moved.

CVOL = *device* = *serial*
specifies the device type and volume serial number of the volume on which the catalog search for the data sets is to begin. If **CVOL** is omitted, the specified group of data sets is assumed to be cataloged on the system residence volume.

PASSWORD
specifies that password-protected data sets contained in the group are to be moved. If **PASSWORD** is omitted, only data sets that are not protected are moved.

UNCATLG
specifies that the catalog entries pertaining to the specified group of data sets are to be removed.

TODD = *ddname*
specifies the name of a DD statement from which DCB and LABEL information are obtained. This parameter, which is valid for data set groups to be moved to magnetic tape volumes, describes the mode and label of the magnetic tape where the moved data sets are to reside. RECFM, LRECL, and BLKSIZE information is ignored. **TODD** can be omitted for 800 bits per inch, 9-track tape with standard labels on single-density drives or for 1600 bits per inch, 9-track tape with standard labels on dual-density drives.

UNLOAD
specifies that the data set is to be unloaded to the receiving volume(s).

MOVE DSGROUP operations cause the specified catalog to be updated automatically unless **UNCATLG** is specified.

COPY DSGROUP Statement

The COPY DSGROUP statement is used to copy groups of data sets that are cataloged on the same control volume and whose names are partially qualified by one or more identical names. Data set groups to be copied must reside on direct access volumes.

INCLUDE and EXCLUDE statements, discussed later in this chapter, can be used to add data sets to or delete data sets from the data set group to be copied.

The format of the COPY DSGROUP statement is:

```
[label] COPY  DSGROUP[ = name]
              ,TO = device = list
              [,CVOL = device = serial]
              [,PASSWORD]
              [,UNCATLG]
              [,CATLG]
              [,TODD = ddname]
              [,UNLOAD]
```

where:

DSGROUP[= *name*]
specifies the cataloged data sets to be copied. If *name*, which is a qualified name, is not coded, all cataloged data sets on the specified volume are to be copied. If

name is coded, all cataloged data sets whose names are qualified by this name are copied. If the name is a fully-qualified data set name, only that data set is copied.

TO = *device* = *list*
specifies the volume or volumes on which the specified group of data sets is to be copied.

CVOL = *device* = *serial*
specifies the device type and volume serial number of the volume on which the search for the catalog is to begin. If **CVOL** is omitted, the specified group of data sets is assumed to be cataloged on the system residence volume.

PASSWORD
specifies that password-protected data sets contained in the group are to be copied. If **PASSWORD** is omitted, only data sets that are not protected are copied.

UNCATLG
specifies that the catalog entries pertaining to the source group of data sets are to be removed.

CATLG
specifies that the copied data sets are to be cataloged on their receiving volumes if they are direct access volumes. If catalogs do not exist on the receiving volumes, they are created.

TODD = *ddname*
specifies the name of a DD statement from which DCB and LABEL information are obtained. This parameter, which is valid for data set groups to be copied to magnetic tape volumes, describes the mode and label of the magnetic tape on which the copied data sets are to reside. RECFM, LRECL, and BLKSIZE information is ignored. **TODD** can be omitted for 800 bits per inch, 9-track tapes with standard labels on single-density drives or for 1600 bits per inch, 9-track tape with standard labels on dual-density drives.

UNLOAD
specifies that the data set is to be unloaded to the receiving volume(s).

Note: The source data sets remain cataloged unless **UNCATLG** is specified.

MOVE PDS Statement

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 delete members.

If IEHMOVE is used to allocate space for an output partitioned data set, the MOVE PDS statement 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. The source data set is scratched if its expiration date has occurred.

The format of the MOVE PDS statement is:

```
[label] MOVE  PDS = name
              ,TO = device = serial
              [,FROM = device = serial]
              [,CVOL = device = serial]
              [,EXPAND = nn]
              [,UNCATLG]
              [,RENAME = name]
              [,FROMDD = ddname]
              [,TODD = ddname]
```

where:

PDS = *name*
specifies the fully-qualified name of the partitioned data set to be moved.

TO = *device* = *serial*
specifies the device type and volume serial number of the volume to which the partitioned data set is to be moved. The *list* parameter may be used when unloading a partitioned data set that must span tape volumes.

FROM = *device* = *serial*
specifies the device type and volume serial number of the volume on which the partitioned data set currently resides, if it is not cataloged. If the data set is cataloged, **FROM** need not be written. FROM = device = list may be used when loading a PDS.

CVOL = *device* = *serial*
specifies the device type and volume serial number of the volume on which the catalog search for the partitioned data set is to begin. If neither **FROM** nor **CVOL** is written, the partitioned data set is assumed to be cataloged on the system residence volume.

EXPAND = *nn*
specifies the number of 256-byte records (up to 99 decimal) to be added to the directory of the specified partitioned data set. **EXPAND** cannot be specified if space is previously allocated.

UNCATLG
specifies that the catalog entry pertaining to the source partitioned data set is to be removed. This parameter may only be used if the source data set is cataloged. If the volume is identified by **FROM**, **UNCATLG** is ignored.

RENAME = *name*
specifies that the data set is to be renamed, and indicates the new name.

FROMDD = *ddname*
specifies the name of a DD statement from which DCB and LABEL information are obtained. This parameter, which is valid for unloaded partitioned data sets on tape volumes, describes the mode and label of the magnetic tape and, in addition, must include the DCB attributes of the unloaded data set (RECFM = FB,LRECL = 80, BLKSIZE = 800). See Table 51 and the tape DD statement under "Job Control Statements" for additional information.

TODD = *ddname*
specifies the name of a DD statement from which DCB and LABEL information are obtained. This parameter, which is valid for partitioned data sets to be unloaded to tape volumes, describes the mode and label of the magnetic tape on which the unloaded data set is to reside and, in addition, must include the DCB attributes of the unloaded data set (RECFM = FB,LRECL = 80,BLKSIZE = 800).

FROMDD can be omitted for 800 bits per inch, 9-track tapes with standard labels on single-density drives or for 1600 bits per inch, 9-track tape with standard labels on dual-density drives.

Note: MOVE PDS causes the specified catalog to be updated automatically unless **UNCATLG** is specified.

COPY PDS Statement

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 delete members.

If IEHMOVE is used to allocate space for an output partitioned data set, the COPY PDS statement can be used to expand a partitioned directory.

If the receiving volume already contains a partitioned data set with the same name, the two are merged.

The format of the COPY PDS statement is:

```
[label] COPY  PDS = name
              ,TO = device = serial
              [,FROM = device = serial]
              [,CVOL = device = serial]
              [,EXPAND = nn]
              [,UNCATLG]
              [,CATLG]
              [,RENAME = name]
              [,FROMDD = ddname]
              [,TODD = ddname]
```

where:

PDS = *name*
specifies the fully-qualified name of the partitioned data set to be copied.

TO = *device* = *serial*
specifies the device type and volume serial number of the volume to which the partitioned data set is to be moved. The *list* value may be used when unloading a partitioned data set that must span tape volumes.

FROM = *device* = *serial*
specifies the device type and volume serial number of the volume on which the partitioned data set currently resides, if it is not cataloged. If the data set is cataloged, **FROM** need not be written. FROM = device = list may be used when loading a PDS.

CVOL = *device* = *serial*
specifies the device type and volume serial number of the volume on which the catalog search for the partitioned data set is to begin. If neither **FROM** nor **CVOL** is written, the partitioned data set is assumed to be cataloged on the system residence volume.

EXPAND = *nn*
specifies the number of 256-byte records (up to 99 decimal) to be added to the directory of the specified partitioned data set. **EXPAND** cannot be specified 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 partitioned data set is cataloged. **UNCATLG** is ignored if the volume is identified by **FROM**.

CATLG
specifies that the copied partitioned data set is to be cataloged on the receiving volume if it is a direct access volume. If a catalog does not exist on the receiving volume, a new catalog is created.

RENAME = *name*
specifies that the data set is to be renamed, and indicates the new name.

FROMDD = *ddname*
specifies the name of a DD statement from which DCB and LABEL information are obtained. This parameter, which is valid for unloaded partitioned data sets residing on magnetic tape volumes, describes the mode and label of the magnetic tape and, in addition, must include the DCB attributes of the unloaded data set (RECFM = FB, LRECL = 80,BLKSIZE = 800). See Table 51 and the tape DD statement under "Job Control Statements" for additional information.

TODD = *ddname*
specifies the name of a DD statement from which DCB and LABEL information are obtained. This parameter, which is valid for partitioned data sets to be unloaded to magnetic tape volumes, describes the mode and label of the magnetic tape on which the unloaded data set is to reside and, in addition, must include the DCB attributes of the unloaded data set (RECFM = FB,LRECL = 80,BLKSIZE = 800).

The **FROMDD** and **TODD** parameters can be omitted for 800 bits per inch, 9-track tapes with standard labels on single-density drives or for 1600 bits per inch, 9-track tape with standard labels on dual-density drives.

Note: The source partitioned data set remains cataloged unless **UNCATLG** is specified.

MOVE CATALOG Statement

The MOVE CATALOG statement is used to move the entries of a catalog without moving the data sets associated with those entries. Certain entries can be excluded from the operation by means of the EXCLUDE statement. If the receiving volume contains a catalog, the source catalog entries are merged with it.

The format of the MOVE CATALOG statement is:

```
[label] MOVE  CATALOG [ = name]
              ,TO = device = serial
              [,CVOL = device = serial]
              [,FROM = device = serial]
              [,FROMDD = ddname]
              [,TODD = ddname]
```

where:

CATALOG [= *name*]
specifies the catalog entries to be moved. If *name*, which is a fully-qualified name, is not coded, all entries in the catalog are to be moved. If *name* is coded, all catalog entries whose names are qualified by this name are moved. If the name is a fully-qualified data set name, only the catalog entry that corresponds to that data set is moved.

TO = *device* = *serial*
specifies the device type and volume serial number of the volume to which the specified catalog entries are to be moved.

CVOL = *device* = *serial*
specifies the device type and volume serial number of the volume on which the search for the catalog is to begin. If both **FROM** and **CVOL** are omitted, the catalog is assumed to reside on the system residence volume.

FROM = *device* = *serial*
specifies the device type and volume serial number of the volume on which an unloaded version of the catalog resides. If neither **FROM** nor **CVOL** is coded, the catalog is assumed to reside on the system residence volume.

FROMDD = *ddname*
specifies the name of a DD statement from which DCB and LABEL information are obtained. This parameter, which is valid for unloaded catalogs residing on magnetic tape volumes, describes the mode the label of the magnetic tape and, in addition, must include the DCB attributes of the unloaded catalog (RECFM = FB,LRECL = 80, BLKSIZE = 800). See Table 51 and the tape DD statement under "Job Control Statements" for additional information.

TODD = *ddname*
specifies the name of a DD statement from which DCB and LABEL information are obtained. This parameter, which is valid for catalogs to be unloaded to magnetic tape volumes, describes the mode and label of the magnetic tape on which the unloaded catalog is to reside.

The **FROMDD** and **TODD** parameters can be omitted for 800 bits per inch, 9-track tapes with standard labels on single-density drives or for 1600 bits per inch, 9-track tape with standard labels on dual-density drives.

COPY CATALOG Statement

The COPY CATALOG statement is used to copy the entries in a catalog without copying the data sets associated with these entries. Certain entries can be excluded from a copy operation with the EXCLUDE statement. If the receiving volume contains a catalog, the source catalog is merged with it.

The format of the COPY CATALOG statement is:

```
[label] COPY  CATALOG [ = name]
              ,TO = device = serial
              [,CVOL = device = serial]
              [,FROM = device = serial]
              [,FROMDD = ddname]
              [,TODD = ddname]
```

where:

CATALOG [= *name*]
specifies the catalog entries to be copied. If *name*, which is a fully-qualified name, is not coded, all entries in the catalog are to be copied. If *name* is coded, all catalog entries whose names are qualified by this name are copied. If the name is a fully-qualified data set name, only the catalog entry that corresponds to that data set is copied.

TO = *device* = *serial*
specifies the device type and volume serial number of the volume onto which the specified catalog entries are to be copied.

CVOL = *device* = *serial*
specifies the device type and volume serial number of the volume on which the search for the catalog is to begin.

FROM = *device* = *serial*
specifies the device type and volume serial number of the volume on which an unloaded version of the catalog resides. If neither **FROM** nor **CVOL** is written, the catalog is assumed to reside on the system residence volume.

FROMDD = *ddname*
specifies the name of a DD statement from which DCB and LABEL information is obtained. This parameter, which is valid for unloaded catalogs residing on tape volumes, describes the mode and label of the tape and, in addition, must include the DCB attributes of the unloaded catalog (RECFM = FB,LRECL = 80, BLKSIZE = 800). See Table 51 and the tape DD statement under "Job Control Statements" for additional information.

TODD = *ddname*
specifies the name of a DD statement from which DCB and LABEL information are obtained. This parameter, which is valid for catalogs to be unloaded onto tape volumes, describes the mode and label of the tape volume on which the unloaded catalog is to reside.

The **FROMDD** and **TODD** parameters can be omitted for 800 bits per inch, 9-track tapes with standard labels on single-density drives or for 1600 bits per inch, 9-track tape with standard labels on dual-density drives.

MOVE VOLUME Statement

The MOVE VOLUME statement is used to move all the data sets residing on a specified volume. Catalog entries associated with the data sets remain unchanged. Data sets to be moved must reside on direct access volumes.

The format of the MOVE VOLUME statement is:

```
[label] MOVE  VOLUME = device = serial
              ,TO = device = list
              [,PASSWORD]
              [,TODD = ddname]
              [,UNLOAD]
```

where:

VOLUME = *device* = *serial*
specifies the device type and volume serial number of the source volume.

TO = *device* = *list*
specifies the volume or volumes to which the data sets are to be moved.

PASSWORD
specifies that password-protected data sets are to be included in the operation. If **PASSWORD** is omitted, only data sets that are not protected are moved.

TODD = *ddname*
specifies the name of a DD statement from which DCB and LABEL information are obtained. This parameter, which is valid for data sets to be moved to tape volumes, describes the mode and label of the receiving tape volume. If the RECFM, BLKSIZE, and LRECL parameters were specified, they are ignored. **TODD** can be omitted for 800 bits per inch, 9-track tapes with standard labels on single-density drives or for 1600 bits per inch, 9-track tape with standard labels on dual-density drives.

UNLOAD
specifies that the data set is to be unloaded to the receiving volume(s).

COPY VOLUME Statement

The COPY VOLUME statement is used to copy all the data sets residing on a specified volume. Catalog entries associated with the data sets remain unchanged. Data sets to be copied must reside on direct access volumes.

The format of the COPY VOLUME statement is:

```
[label] COPY  VOLUME = device = serial
              ,TO = device = list
              [,PASSWORD]
              [,CATLG]
              [,TODD = ddname]
              [,UNLOAD]
```

where:

VOLUME = *device* = *serial*
specifies the device type and volume serial number of the source volume.

TO = *device* = *list*
specifies the volume or volumes to which the data sets are to be copied.

PASSWORD
specifies that password-protected data sets are to be included in the operation. If **PASSWORD** is omitted, only data sets that are not protected are copied.

CATLG
specifies that all copied data sets are to be cataloged in a SYSCTLG (system catalog) data set on the direct access receiving volume. If a catalog does not exist on the receiving volume, it is created.

TODD = *ddname*
specifies the name of a DD statement from which DCB and LABEL information are obtained. This parameter, which is valid for data sets to be copied to magnetic tape volumes, describes the mode and label of the receiving magnetic tape volume. If RECFM, BLKSIZE, and LRECL were specified, they are ignored. **TODD** can be omitted for 800 bits per inch, 9-track tapes with standard labels on single-density drives or for 1600 bits per inch, 9-track tape with standard labels on dual-density drives.

UNLOAD
specifies that the data set is to be unloaded to the receiving volume(s).

If **CATLG** is specified and the source volume contains a SYSCTLG data set, error messages indicating that an inconsistent index structure exists are issued when the source SYSCTLG data set entries are merged into the catalog on the receiving volume. (Because the SYSCTLG data set is the last to be copied, only those entries representing cataloged data sets not residing on the source volume are copied into a receiving volume's SYSCTLG data set; entries representing all data sets residing on the source volume have already been made in the receiving SYSCTLG data set.)

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. Any number of INCLUDE statements can modify a MOVE or COPY statement. The INCLUDE statement is invalid when data is unloaded or when unloaded data is moved or copied.

The format of the INCLUDE statement is:

```
[label] INCLUDE  DSNAME = name
                 [,MEMBER = membername]
                 [,FROM = device = list  ]
                 [,CVOL = device = serial]
```

where:

DSNAME = *name*
specifies the fully-qualified name of a data set. When the INCLUDE statement modifies a MOVE DSGROUP or COPY DSGROUP statement, the named data set is included in the group. When the INCLUDE statement modifies a MOVE PDS or COPY PDS statement, either the entire named partitioned data set or a member of the data set is included in the operation.

MEMBER = *membername*
specifies the name of a member of the partitioned data set named in the DSNAME parameter. This member is merged with the partitioned data set that is moved or copied. Its record characteristics must be the same as those of the data set with which it it being merged. The data set containing this member is not scratched, regardless of the operation. This parameter is valid and required only when the INCLUDE statement modifies a MOVE PDS or COPY PDS statement.

FROM = *device* = *list*
specifies the volume or volumes on which the data set resides, if the data set is not cataloged. If the data set is cataloged, **FROM** need not be specified. If both **FROM** and **CVOL** are omitted, the specified data set is assumed to be cataloged on the system residence volume.

CVOL = *device* = *serial*
specifies the device type and volume serial number of the volume on which the catalog search for the data set is to begin. If both **FROM** and **CVOL** are omitted, the specified data set is assumed to be cataloged on the system residence volume.

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 or catalog entries excluded from a MOVE DSGROUP or MOVE CATALOG operation remain available. Only one EXCLUDE statement can modify a MOVE DSGROUP, COPY DSGROUP, MOVE CATALOG, or COPY CATALOG statement. The EXCLUDE statement is invalid when data is unloaded or when unloaded data is moved or copied.

The format of the EXCLUDE statement is:

```
[label] EXCLUDE {DSGROUP = name       }
                {MEMBER = membername }
```

where:

DSGROUP = *name*
specifies a qualified name. If the EXCLUDE statement modifies a MOVE DSGROUP or COPY DSGROUP statement, all data sets whose names are qualified by this name are excluded from the operation. If the EXCLUDE statement modifies a MOVE CATALOG or COPY CATALOG statement, all catalog entries whose names are qualified by this name are excluded from the operation.

MEMBER = *membername*
identifies a member to be excluded from the partitioned data set being moved or copied when the EXCLUDE statement modifies a MOVE PDS or COPY PDS statement.

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.

The format of the SELECT statement is:

```
[label] SELECT  {MEMBER = (name[,name...)                          }
                {MEMBER = ((name,newname)[,(name,newname)]. . .) }
```

where:

MEMBER = (*name*[,*name*]...)
identifies the members to be moved or copied. These members belong to the partitioned data set identified in the preceding MOVE PDS or COPY PDS statement.

MEMBER = ((*name*,*newname*(,,(*name*,*newname*)]...)
identifies the members to be moved or copied and gives the new name for each member.

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 identical 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 format of the REPLACE statement is:

```
[label] REPLACE  DSNAME = name
                 ,MEMBER = name
                 [,FROM = device = serial]
                 [,CVOL = device = serial]
```

where:

DSNAME = *name*
specifies the fully-qualified name of the partitioned data set that contains the new member.

MEMBER = *name*
specifies the name of the member.

FROM = *device* = *serial*
specifies the device type and volume serial number of the volume that contains the partitioned data set named in the DSNAME parameter. If the partitioned data set is cataloged, **FROM** need not be specified.

CVOL = *device* = *serial*
specifies the device type and volume serial number of the control volume on which the catalog search for the partitioned data set containing the new member is to begin.

If neither **FROM** nor **CVOL** is specified, the partitioned data set is assumed to be cataloged on the system residence volume.

IEHMOVE Examples

The following examples illustrate some of the uses of IEHMOVE. Table 52 can be used as a quick reference guide to IEHMOVE examples. The numbers in the "Example" column point to the examples that follow.

Table 52. IEHMOVE Example Directory

Operation	Data Set Organization	Device	Comments	Example
MOVE	Sequential	3330 Disk, 2314 Disks	Source volume is demounted after job completion. Two mountable disks.	1
COPY	Sequential	2314 Disk, 3330 Disk, 2314 or 2319 Disks[1]	Three cataloged sequential data sets are to be copied. The 2314 or 2319 are mountable.	2
MOVE	Data set group	2311 Disk, 2301 Drum, 2314 Disk	Data set group is to be moved. The 2314 disks are mountable.	3
MOVE PDS	Partitioned	3330 Disk, 2311 Disk 2314 Disks	A partitioned data set is to be moved; a member from another PDS is to be merged with it.	4
MOVE and CATALOG	Catalog	2314 Disk, 3330 Disk, 2311 Disk	Catalog is to be moved from system residence volume to second volume. Source catalog is scratched from system residence volume.	5
MOVE	Catalog	3330 Disk, 2314 Disk	Selected catalog entries are to be moved from system residence to a second volume. SYSCTLG is scratched.	6
MOVE	Volume	3330 Disk, 2314 Disks	Volume of data sets is to be moved.	7
MOVE	Partitioned	2301 Drum, 2314 Disks	A data set is to be moved to a volume on which space was previously allocated.	8
MOVE	Partitioned	3330 Disk, 2311 Disk, 2314 Disk	Three data sets are to be moved and unloaded to a volume on which space was previously allocated.	9
MOVE	Sequential	2311 Disk, 2314 Disk, 2400 Tape	A sequential data set is to be unloaded to an unlabeled 9-track tape volume.	10
MOVE	Sequential	3330 Disk, 2314 Disk, 2400 Tape	Unloaded data sets are to be loaded from a single volume.	11
COPY	Sequential	2314 Disk, 2311 Disk, 2400 Tape	Data sets are to be copied from separate source volumes.	12
COPY	Partitioned	2400 Tape, 2314 Disks	Unloaded data sets are to be loaded from unlabeled tape to a specific device.	13

[1] Note that the 2319 disk is functionally equivalent to the 2314 disk; to use the 2319, specify 2314 in the control statement.

IEHMOVE Example 1

In this example, three data sets (SEQSET1, SEQSET2, and SEQSET3) are to be 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.

The example follows:

```
//MOVEDS   JOB  09#550,GREEN
//         EXEC PGM=IEHMOVE
//SYSPRINT DD   SYSOUT=A
//SYSUT1   DD   UNIT=3330,VOLUME=SER=333000,DISP=OLD
//DD1      DD   UNIT=2314,VOLUME=SER=111111,DISP=OLD
//DD2      DD   UNIT=(2314,,DEFER),DISP=OLD,
// VOLUME=(PRIVATE,,SER=(231400))
//DD3      DD   VOLUME=(PRIVATE,RETAIN,SER=(231411)),
// UNIT=2314,DISP=OLD
//SYSIN    DD   *
       MOVE   DSNAME=SEQSET1,TO=2314=231400,FROM=2314=231411
       MOVE   DSNAME=SEQSET2,TO=2314=231412,FROM=2314=231411
       MOVE   DSNAME=SEQSET3,TO=2314=231413,FROM=2314=231411
/*
```

The control statements are discussed below:

- SYSUT1 DD defines the device that is to contain the work data set.
- DD1 DD defines the system residence device.
- DD2 DD defines the mountable device on which the receiving volumes will be mounted as they are required.
- DD3 DD defines a mountable device on which the source volume is to be 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 231400, 231412, and 231413, respectively. The source data sets are scratched.

IEHMOVE Example 2

In this example, three cataloged data sets are to be copied to a 2314 or 2319 volume. Note that the 2319 is functionally equivalent to the 2314; to use a 2319, specify 2314 in the control statement. Space is allocated by IEHMOVE. The catalog is not updated. The source data sets are not scratched.

The example follows:

```
//COPYPDS  JOB  09#550,GREEN
//         EXEC PGM=IEHMOVE
//SYSPRINT DD   SYSOUT=A
//SYSUT1   DD   UNIT=2314,VOLUME=SER=231400,DISP=OLD
//DD1      DD   UNIT=3330,VOLUME=SER=111111,DISP=OLD
//DD2      DD   UNIT=2314,VOLUME=SER=231400,DISP=OLD
//DD3      DD   UNIT=2314,VOLUME=SER=231401,DISP=OLD
//SYSIN    DD   *
         COPY   DSNAME=SEQSET1,TO=2314=231401
         COPY   DSNAME=SEQSET3,TO=2314=231401
         COPY   DSNAME=SEQSET4,TO=2314=231401
/*
```

The control statements are discussed below:

- SYSUT1 DD defines the device that is to contain the work data set.
- DD1 DD defines the system residence device.
- DD2 DD defines a mountable device on which the source volume is mounted.
- DD3 DD defines a mountable device on which the receiving volume is mounted.
- SYSIN DD defines the control data set which follows in the input stream.
- COPY copies the source data sets onto volume 231401.

Note: This example implies that the cataloged source data sets all exist on a 2314 volume. If the data sets existed on a volume or volumes other than a 2314, the necessary DD statements would have to be included in this example to define the applicable mountable device(s).

IEHMOVE Example 3

In this example, the data set group A.B.C—which comprises data sets A.B.C.X, A.B.C.Y, and A.B.C.Z—is moved from two 2314 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.

The example follows:

```
//MOVEDSG  JOB  09#550,GREEN
//         EXEC PGM=IEHMOVE
//SYSPRINT DD   SYSOUT=A
//SYSUT1   DD   UNIT=2311,VOLUME=SER=231101,DISP=OLD
//DD1      DD   UNIT=2301,VOLUME=SER=111111,DISP=OLD
//DD2      DD   UNIT=2314,VOLUME=SER=231401,DISP=OLD
//DD3      DD   UNIT=2314,VOLUME=SER=231410,DISP=OLD
//DD4      DD   UNIT=2314,VOLUME=SER=231411,DISP=OLD
//SYSIN    DD   *
        MOVE   DSGROUP=A.B.C,TO=2314=231401
/*
```

The control statements are discussed below:

- SYSUT1 DD defines the device that is to contain the work data set.
- DD1 DD defines the system residence device.
- DD2 DD defines a mountable device on which the receiving volume is to be mounted.
- DD3 DD defines a mountable device on which one of the source volumes is to be mounted.
- DD4 DD defines a mountable device on which one of the source volumes is to be mounted.
- SYSIN DD defines the control data set, which follows in the input stream.
- MOVE moves the specified data sets to volume 231401.

Note: This example can be used to produce the same result without the use of the DD4 DD statement, using one less mountable 2314 device. With DD3 and DD4, both of the source volumes are mounted at the start of the job. With DD3 only, the 231410 volume is mounted at the start of the job. After the 231410 volume is processed, the utility requests that the operator mount the 231411 volume. In this case the DD3 statement is coded:

```
//DD3 DD  UNIT = (2314,,DEFER),DISP = OLD,VOLUME = (PRIVATE,,
//        SER = (231410))
```

IEHMOVE Example 4

In this example, a partitioned data set (PARTSET1) is to be moved to a disk volume. In addition, a member (PARMEM3) from another partitioned data set (PARTSET2) is to be merged with the source members on the receiving volume. The source partitioned data set (PARTSET1) is scratched. Space is allocated by IEHMOVE.

The example follows:

```
//MOVEPDS  JOB  09#550,GREEN
//         EXEC PGM=IEHMOVE
//SYSPRINT DD   SYSOUT=A
//SYSUT1   DD   UNIT=3330,VOLUME=SER=333000,DISP=OLD
//DD1      DD   UNIT=2311,VOLUME=SER=111111,DISP=OLD
//DD2      DD   UNIT=2314,VOLUME=SER=231400,DISP=OLD
//DD3      DD   UNIT=2314,VOLUME=SER=231410,DISP=OLD
//DD4      DD   UNIT=2314,VOLUME=SER=231420,DISP=OLD
//SYSIN    DD   *
     MOVE    PDS=PARTSET1,TO=2314=231420,FROM=2314=231400
  INCLUDE    DSNAME=PARTSET2,MEMBER=PARMEM3,FROM=2314=231410
/*
```

The control statements are discussed below:

- SYSUT1 DD defines the device that is to contain the work data set.
- DD1 DD defines the system residence device.
- The DD2, DD3, and DD4 DD statements define mountable 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.

IEHMOVE Example 5

In this example, the SYSCTLG data set is to be moved from the system residence volume to a mountable 2311 volume. Space is allocated by IEHMOVE. The source catalog is scratched from the system residence volume.

The example follows:

```
//MOVECAT1 JOB  09#550,GREEN
//         EXEC PGM=IEHMOVE
//SYSPRINT DD   SYSOUT=A
//SYSUT1   DD   UNIT=2314,VOLUME=SER=231400,DISP=OLD
//DD1      DD   UNIT=3330,VOLUME=SER=111111,DISP=OLD
//DD2      DD   UNIT=2311,VOLUME=SER=222222,DISP=OLD
//SYSIN    DD   *
         MOVE  CATALOG,TO=2311=222222
/*
```

The control statements are discussed below:

- SYSUT1 DD defines the device that is to contain the work data set.
- DD1 DD defines the system residence device. The system residence volume contains the catalog to be moved.
- DD2 DD defines the mountable device on which the receiving volume is to be mounted.
- SYSIN DD defines the control data set, which follows in the input stream.
- MOVE specifies the move operation and defines the receiving volume.

IEHMOVE Example 6

In this example, the data set group A.B.C—which comprises data sets entries A.B.C.X, A.B.C.Y, and A.B.C.Z—is to be moved from the SYSCTLG data set to a mountable 2314 volume. If no catalog exists on the 2314 volume, one is created; if a catalog does exist, the specified entries are merged into it. The source SYSCTLG data set is scratched. The work data set is deleted when the job step is completed.

The example follows:

```
//MOVECAT2 JOB  09#550,GREEN
//         EXEC PGM=IEHMOVE
//SYSPRINT DD   SYSOUT=A
//SYSUT1   DD   UNIT=2314,VOLUME=SER=231402,DISP=OLD
//DD1      DD   UNIT=3330,VOLUME=SER=111111,DISP=OLD
//DD2      DD   UNIT=2314,VOLUME=SER=231402,DISP=OLD
//SYSIN    DD   *
         MOVE  CATALOG=A.B.C,TO=2314=231402
/*
```

The control statements are discussed below:

- SYSUT1 DD defines the device that is to contain the work data set. (Because IEHMOVE deletes the work data set at the completion of the program, it can be contained on the receiving volume, provided there is room for it.)
- DD1 DD defines the system residence device. The system residence volume contains the entries to be moved.
- DD2 DD defines the mountable device on which the receiving volume is to be mounted.
- SYSIN DD defines the control data set, which follows in the input stream.
- MOVE specifies a move operation for selected entries and defines the receiving volume.

IEHMOVE Example 7

In this example, a volume of data sets is to be moved to a 2314 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. The work data set is deleted when the job step is completed.

The example follows:

```
//MOVEVOL  JOB  09#550,GREEN
//         EXEC PGM=IEHMOVE
//SYSPRINT DD   SYSOUT=A
//SYSUT1   DD   UNIT=2314,VOLUME=SER=231400,DISP=OLD
//DD1      DD   UNIT=3330,VOLUME=SER=111111,DISP=OLD
//DD2      DD   UNIT=2314,VOLUME=SER=231400,DISP=OLD
//DD3      DD   UNIT=2314,VOLUME=SER=231401,DISP=OLD
//SYSIN    DD   *
         MOVE  VOLUME=2314=231401,TO=2314=231400,PASSWORD
/*
```

The control statements are discussed below:

- SYSUT1 DD defines the device that is to contain the work data set. The work data set is removed from the receiving volume when the job step is completed.
- DD1 DD defines the system residence device.
- DD2 DD defines the mountable device on which the receiving volume is to be mounted.
- DD3 DD defines a mountable device on which the source volume is to be 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 to be included in the operation.

Note: IEHPROGM can be used to uncatalog catalog entries pertaining to source data sets and to catalog the moved versions of those data sets.

IEHMOVE Example 8

In this example, a partitioned data set is to be moved to a 2314 volume on which space has been previously allocated for the data set. The source data set is scratched. The work data set is deleted when the job step is completed.

The example follows:

```
//ALLOCATE JOB  09#550,GREEN
//         EXEC PGM=IEHPROGM
//SYSPRINT DD   SYSOUT=A
//SET1     DD   DSNAME=PDSSET1,UNIT=2314,DISP=(NEW,KEEP),
// VOLUME=SER=231401,SPACE=(TRK,(100,10,10)),
// DCB=(RECFM=FB,LRECL=80,BLKSIZE=2000)
//SYSIN    DD   *
/*
//         EXEC PGM=IEHMOVE
//SYSPRINT DD   SYSOUT=A
//SYSUT1   DD   UNIT=2314,VOLUME=SER=231401,DISP=OLD
//DD1      DD   UNIT=2301,VOLUME=SER=111111,DISP=OLD
//DD2      DD   UNIT=2314,VOLUME=SER=231401,DISP=OLD
//DD3      DD   UNIT=2314,VOLUME=SER=231402,DISP=OLD
//SYSIN    DD   *
         MOVE  PDS=PDSSET1,TO=2314=231401,FROM=2314=231402
/*
```

The IEHPROGM job step is used to allocate space for data set PDSSET1 on a 2314 volume.

The control statements are discussed below:

- SYSUT1 DD defines the device that is to contain the work data set. The data set is removed from the receiving volume at the completion of the program.
- 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 mountable device on which the source volume is to be 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.

IEHMOVE Example 9

In this example, three partitioned data sets are to be moved from three separate source volumes to a 2311 volume. The source data set PDSSET3 is unloaded. (The record size exceeds the track capacity of the receiving volume.) The work data set is deleted when the job step is completed.

The example follows:

```
//ALLOCATE JOB  09#550,GREEN
//         EXEC PGM=IEHPROGM
//SYSPRINT DD   SYSOUT=A
//SET1     DD   DSNAME=PDSSET1,UNIT=2311,DISP=(NEW,KEEP),
// VOLUME=SER=231101,SPACE=(TRK,(100,10,5)),
// DCB=(RECFM=FB,LRECL=80,BLKSIZE=1600)
//SET2     DD   DSNAME=PDSSET2,UNIT=2311,DISP=(NEW,KEEP),
// VOLUME=SER=231101,SPACE=(TRK,(50,5,5)),
// DCB=(RECFM=F,LRECL=80,BLKSIZE=80)
//SET3     DD   DSNAME=PDSSET3,UNIT=2311,DISP=(NEW,KEEP),
// VOLUME=SER=231101,SPACE=(TRK,(50,5)),
// DCB=(RECFM=U,BLKSIZE=5000)
//SYSIN    DD   *
/*
//         EXEC PGM=IEHMOVE
//SYSPRINT DD   SYSOUT=A
//SYSUT1   DD   UNIT=2311,VOLUME=SER=231101,DISP=OLD
//DD1      DD   UNIT=3330,VOLUME=SER=111111,DISP=OLD
//DD2      DD   UNIT=(2314,,DEFER),DISP=OLD,
// VOLUME=(PRIVATE,,SER=(231400))
//DD3      DD   UNIT=2311,VOLUME=SER=231101,DISP=OLD
//SYSIN    DD   *
         MOVE   PDS=PDSSET1,TO=2311=231101,FROM=2314=231400
         MOVE   PDS=PDSSET2,TO=2311=231101,FROM=2314=231401
         MOVE   PDS=PDSSET3,TO=2311=231101,FROM=2314=231402
/*
```

The IEHPROGM job step is used to allocate space for the partitioned data sets PDSSET1, PDSSET2, and PDSSET3 on the receiving volume. The SPACE parameter in the SET3 DD statement allocates space for a sequential data set. This is necessary to successfully unload the partitioned data set PDSSET3. 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 below:

- SYSUT1 DD defines the device that is to contain the work data set.
- DD1 DD defines the system residence device.
- DD2 DD defines a mountable device on which the source volumes are mounted as they are required.
- DD3 DD defines a mountable 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.

Note: For a discussion on estimating space allocations, refer to *OS Data Management Services Guide*, GC26-3746.

IEHMOVE Example 10

In this example, a sequential data set is to be unloaded onto a 9-track, unlabeled tape volume (800 bits per inch). The work data set resides on the source volume and is deleted when the job step is completed.

The example follows:

```
//UNLOAD   JOB  09#550,GREEN
//         EXEC PGM=IEHMOVE
//SYSPRINT DD   SYSOUT=A
//SYSUT1   DD   UNIT=2314,VOLUME=SER=231400,DISP=OLD
//DD1      DD   UNIT=2311,VOLUME=SER=111111,DISP=OLD
//DD2      DD   UNIT=2314,VOLUME=SER=231400,DISP=OLD
//TAPEOUT  DD   UNIT=2400,VOLUME=SER=SCRTCH,DISP=OLD,
// DCB=(DEN=2),LABEL=(,NL)
//SYSIN    DD   *
         MOVE   DSNAME=SEQSET1,TO=2400=SCRTCH,                  ¢C
               FROM=2314=231400,TODD=TAPEOUT
/*
```

The control statements are discussed below:

- SYSUT1 DD defines the device that is to contain the work data set.
- DD1 DD defines the system residence device.
- DD2 DD defines a mountable device on which the source volume is mounted.
- TAPEOUT DD defines a mountable 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 2314 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.

IEHMOVE Example 11

In this example, three unloaded sequential data sets are to be loaded from a labeled, 7-track tape volume (556 bits per inch) to a 2314 volume. Space is allocated by IEHMOVE. The example assumes that the 2314 volume is capable of supporting the data sets in their original forms.

The example follows:

```
//LOAD     JOB  09#550,GREEN
//         EXEC PGM=IEHMOVE
//SYSPRINT DD   SYSOUT=A
//SYSUT1   DD   UNIT=2314,VOLUME=SER=231400,DISP=OLD
//DD1      DD   UNIT=3330,VOLUME=SER=111111,DISP=OLD
//DD2      DD   UNIT=2314,VOLUME=SER=231400,DISP=OLD
//TAPESETS DD   DSNAME=UNLDSET1,UNIT=2400,VOLUME=SER=001234,
// DISP=OLD,LABEL=(1,SL),DCB=(DEN=1,TRTCH=C)
//SYSIN    DD   *
         MOVE   DSNAME=UNLDSET1,TO=2314=231400,                 ¢C
               FROM=2400-2=(001234,1),FROMDD=TAPESETS
         MOVE   DSNAME=UNLDSET2,TO=2314=231400,                 ¢C
               FROM=2400=(001234,2),FROMDD=TAPESETS
         MOVE   DSNAME=UNLDSET3,TO=2314=231400,                 ¢C
               FROM=2400=(001234,3),FROMDD=TAPESETS
/*
```

The control statements are discussed below:

- SYSUT1 DD defines the device that is to contain the work data set.
- DD1 DD defines the system residence device.
- DD2 DD defines a mountable device on which the receiving volume is mounted.
- TAPESETS DD defines a mountable device on which the source 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.

Note: 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** or **TO** parameter on the utility control statement.

IEHMOVE Example 12

In this example, two sequential data sets are to be copied from separate source volumes to a 2314 volume. Space is allocated by IEHMOVE. Only one 9-track tape drive is available for the operation.

The example follows:

```
//DEFER    JOB  09#550,GREEN
//         EXEC PGM=IEHMOVE
//SYSPRINT DD   SYSOUT=A
//SYSUT1   DD   UNIT=2314,VOLUME=SER=231400,DISP=OLD
//DD1      DD   UNIT=2311,VOLUME=SER=111111,DISP=OLD
//DD2      DD   UNIT=2314,VOLUME=SER=231400,DISP=OLD
//TAPE1    DD   DSNAME=SEQSET1,UNIT=2400,DISP=OLD,
// LABEL=(,SL),VOLUME=SER=01234,DCB=(DEN=2)
//TAPE2    DD   DSNAME=SEQSET9,UNIT=AFF=TAPE1,DISP=OLD,
// LABEL=(4,SL),VOLUME=SER=01235,DCB=(DEN=2)
//SYSIN    DD   *
         COPY   DSNAME=SEQSET1,TO=2314=231400,                    %C
               FROM=2400=(001234,2),FROMDD=TAPE1
         COPY   DSNAME=SEQSET9,TO=2314=231400,                    %C
               FROM=2400=(001235,4),FROMDD=TAPE2
/*
```

The control statements are discussed below:

- SYSUT1 DD defines the volume that is to contain the work data set.
- DD1 DD defines the system residence device.
- DD2 DD defines a mountable device on which the receiving volume is mounted.
- TAPE1 DD defines a mountable device on which the first volume to be processed is mounted. The source data set is the second data set on the volume, as specified in the utility control statement.
- TAPE2 DD defines a mountable device on which the second volume to be processed is mounted when it is required. The source data set is the fourth data set on the volume, as specified in the utility control statement.
- SYSIN DD defines the control data set, which follows in the input stream.
- COPY copies the data sets to the receiving volume.

Note: To copy 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** or **TO** parameter on the utility control statement.

IEHMOVE Example 13

In this example, three unloaded partitioned data sets residing on an unlabeled tape volume mounted on device 282 are copied to a 2314 volume mounted on device 191.

The example follows:

```
//LOAD     JOB  MEDDAUGH,PS40300439,MSGLEVEL=1
//         EXEC PGM=IEHMOVE
//SYSPRINT DD   SYSOUT=A
//SYSABEND DD   SYSOUT=A
//SYSUT1   DD   UNIT=191,VOLUME=SER=231400,DISP=OLD
//DD1      DD   UNIT=191,VOLUME=SER=231400,DISP=OLD
//TAPE1    DD   UNIT=282,VOLUME=SER=NLTAPE,DISP=OLD,
// LABEL=(,NL),DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)
//SYSIN    DD   *
 COPY PDS=DSET1,FROM=282=(NLTAPE,1),TO=191=231400,FROMDD=TAPE1
 COPY PDS=DSET2,FROM=282=(NLTAPE,2),TO=191=231400,FROMDD=TAPE1
 COPY PDS=DSET3,FROM=282=(NLTAPE,3),TO=191=231400,FROMDD=TAPE1
/*
```

The control statements are discussed below:

- SYSUT1 DD defines the work data set.
- TAPE1 DD defines the source data sets. They are, in the order in which they reside on the volume, DSET1, DSET2, and DSET3.
- DD1 DD defines the receiving volume.
- SYSIN DD defines the control data set, which follows in the input stream.
- COPY copies the unloaded partitioned data sets from the unlabeled tape to the receiving volume.

Note: To copy data sets from a nonlabeled tape, you must place a label in the *list* field of the **FROM** parameter of the utility control statement. Following this label, the sequence numbers of the data sets must also be included in the same field. The unit address must appear in the device field of the **FROM** or **TO** parameter whenever you want to move from or copy to a specific device.

IEHMOVE Example 14

In this example, a BDAM data set on two 2314 volumes is copied to a 3330 disk storage device and is renamed. Space is allocated by IEHMOVE.

The example follows:

```
//COPYBDAM    JOB  09#550,GREEN
//            EXEC PGM=IEHMOVE
//SYSPRINT DD   SYSOUT=A
//SYSUT1   DD   UNIT=3330,VOL=SER=333001,DISP=OLD
//DD1      DD   UNIT=2314,VOL=SER=111111,DISP=OLD
//DD2      DD   UNIT=(2314,2),VOL=SER=(231401,231402),DISP=OLD
//DD3      DD   UNIT=3330,VOL=SER=333001,DISP=OLD
//SYSIN    DD   *
 COPY FROM=2314=(231401,231402),TO=3330=333001,                  %C
               DSNAME=TWOVOL.BDAM,RENAME=ONEVOL.BDAM
/*
```

The control statements are discussed below:

- SYSUT1 DD defines the work data set.
- DD1 DD defines the system residence device on which the source volumes reside.
- DD2 DD defines the two mountable volumes to be copied. Both volumes must be mounted in parallel.
- DD3 DD defines the receiving volume.
- SYSIN DD defines the control data set, which follows in the input stream.
- COPY copies the data set to the receiving volume and renames it.

IEHPROGM Program

IEHPROGM is a system utility used to modify system control data and to maintain data sets at an organizational level. (See "Introduction" for general system utility information.)

IEHPROGM can be used to:

- Scratch a data set or a member.
- Rename a data set or a member.
- Catalog or uncatalog a data set.
- Build or delete an index or an index alias.
- Connect or release two volumes.
- Build and maintain a generation index.
- Maintain data set passwords.

At the completion or termination of the program, the highest return code encountered within the program is passed to the calling program.

Scratching a Data Set or Member

IEHPROGM can be used to scratch the following from a direct access volume or volumes:

- Sequential, indexed sequential, partitioned, or direct access data sets.
- Members of a partitioned data set.
- Password-protected data sets.
- Data sets named by the operating system.

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 re-allocation.

The space occupied by a data set residing on a device that operates in split-cylinder mode is not available for re-allocation until all data sets sharing the cylinder have been scratched.

A member is considered scratched when its name is removed from the directory of the partitioned data set that contains it. The space occupied by a scratched member is not available for re-allocation until the partitioned data set is scratched, or the data set is compressed in place. (When scratching a member of a partitioned data set, all aliases of that member should also be removed from the directory.)

Renaming a Data Set or Member

IEHPROGM can be used to rename a data set or member that resides on a direct access volume. In addition, the program can be used to change any member aliases.

Cataloging or Uncataloging a Data Set

IEHPROGM can be used to catalog or uncatalog a sequential, indexed sequential, partitioned, or BDAM data set.

A data set is cataloged when its fully-qualified name and volume identification are entered in one or more index levels of the SYSCTLG data set. The program catalogs a data set by generating an entry, containing the data set name and associated volume information, in the index of the catalog. If higher level indexes are necessary to catalog the data set, they are automatically created.

The catalog function is used to: (1) catalog a data set that was not cataloged when it was created or (2) satisfy, if necessary, the requirement that a higher level index or indexes be created. Figure 54 shows how data set A.F.G is cataloged on the system residence volume. Note that the level F index does not exist in the SYSCTLG data set before the catalog operation.

IEHPROGM uncatalogs a data set by removing the data set name and associated volume information from the lowest level index of the catalog.

The uncatalog function of the program differs from a DISP = (...,UNCATLG) specification in a DD statement in that the DISP = (...,UNCATLG) specification cannot remove an entry from the SYSCTLG data set on a volume other than the system residence volume unless the two volumes are properly connected.

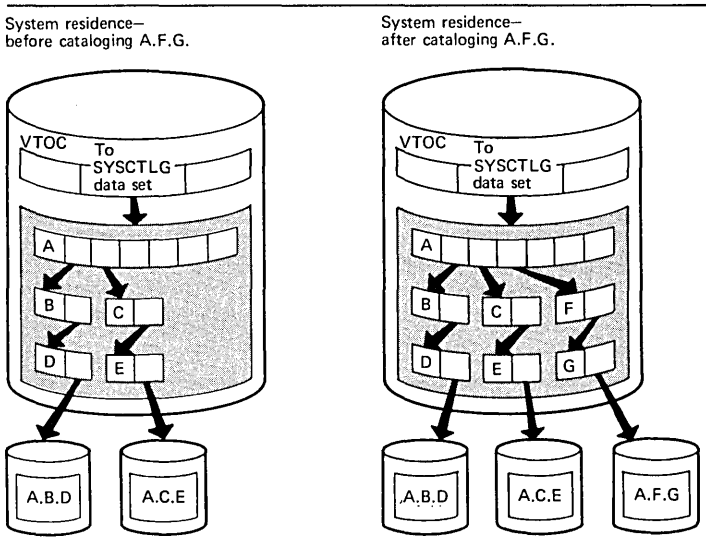


Figure 54. Cataloging a Data Set Using IEHPROGM

Figure 55 shows how data set A.F.G is uncataloged by the program. Prior to the operation, the fully qualified name and associated volume information are represented in the catalog. The uncatalog operation removes the lowest level entry from the index structure. Note that the structure A.F remains in the catalog.

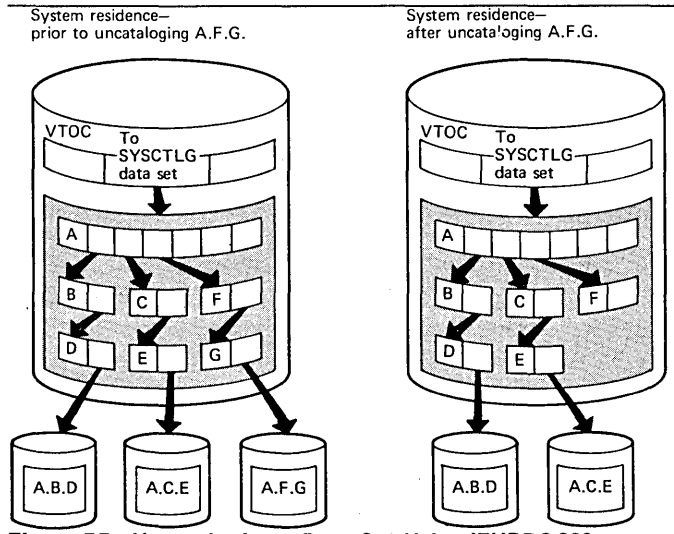


Figure 55. Uncataloging a Data Set Using IEHPROGM

IEHPROGM can be used to build a new index in the catalog or to delete an existing index. In building an index, the program automatically creates as many higher level indexes as are necessary to complete the specified structure.

IEHPROGM can be used to delete one or more indexes from an index structure; however, an index cannot be deleted if it contains any entries. That is, it cannot be deleted if it refers to a lower level index or if it is part of a structure indicating the fully-qualified name of a cataloged data set.

Figure 56 shows an index structure before and after a build operation. The left-hand portion of the figure shows two cataloged data sets, A.Y.YY and A.B.X.XX, before the build operation. The right-hand portion of the figure shows the index structure after the build operation, which was used to build index A.B.C.D.E. Note in the left-hand portion of the figure that index levels C and D do not exist before the build operation. These levels are automatically created when the level E index is built.

Note that when the level E index is subsequently deleted, the level C and D indexes are not automatically deleted by the program. To delete these index levels, delete: A.B.C.D.E, A.B.C.D, and A.B.C, in that order. The level B index cannot be deleted because data set A.B.X.XX and the X level index are dependent upon the level B index.

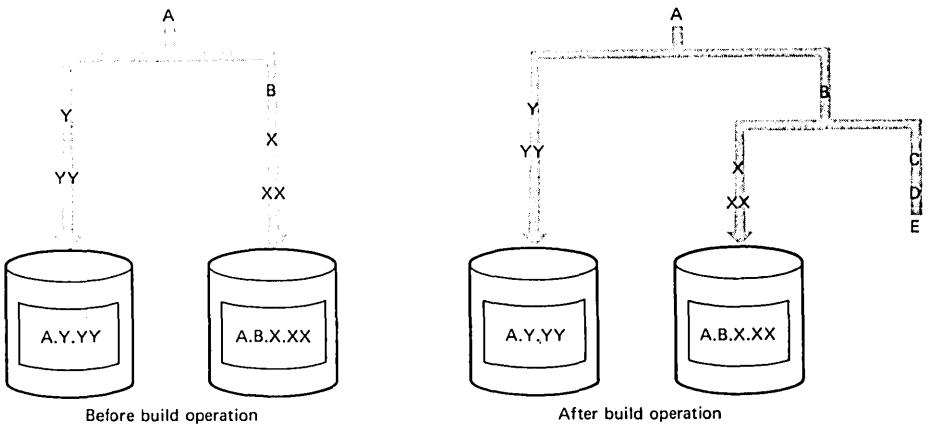


Figure 56. Index Structure Before and After an IEHPROGM Build Operation

IEHPROGM can be used to assign an alternative name (alias) to the highest level index of a catalog or to delete an alias previously assigned. An alias cannot, however, be assigned to the highest level of a generation index.

Figure 57 shows an alias, XX, that is assigned to index A (a high level index). The cataloged data set A.B.C can be referred to as either A.B.C or XX.B.C.

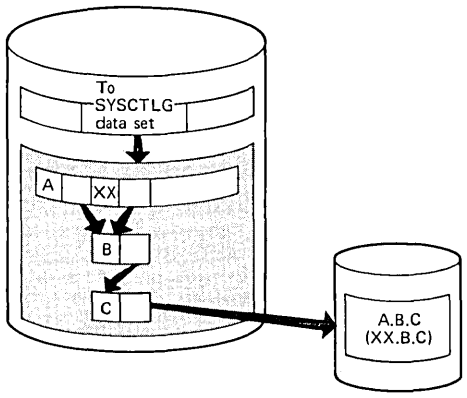


Figure 57. Building an Index Alias Using IEHPROGM

IEHPROGM can be used to *connect* a volume to a second volume by placing an entry (that contains an index name and the volume serial number and device type of the second volume) into a high level index on the first volume. The program can subsequently *release* the volumes by removing the entry from the high level index. If two volumes are connected:

- The catalog (SYSCTLG data set) can be extended to a second volume, if necessary.
- Normal JCL can be used to process (retrieve, uncatalog, etc.) data sets cataloged on the second volume (assuming that the first volume is the system residence volume).

If the SYSCTLG data set is extended to a second volume, it must be identified on that volume.

Figure 58 shows how the system residence volume can be connected to a second volume. Any subsequent index search for index X on the system residence volume is carried to the second volume.

Note: The index name of each high level index existing on the second volume must be present in the first volume; when a new high level index is placed on a second volume, the second volume should be connected to the first volume.

Figure 59 shows three volumes connected to the system residence volume. All volumes are accessible (through high level indexes X, Y, and Z) to the operating system.

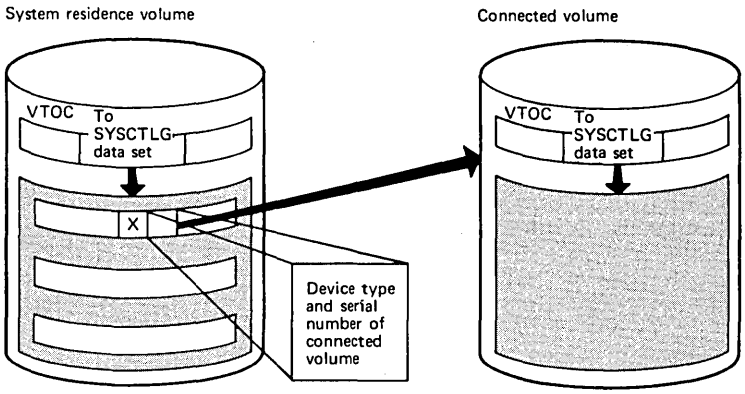


Figure 58. Connecting a Volume to a Second Volume Using IEHPROGM

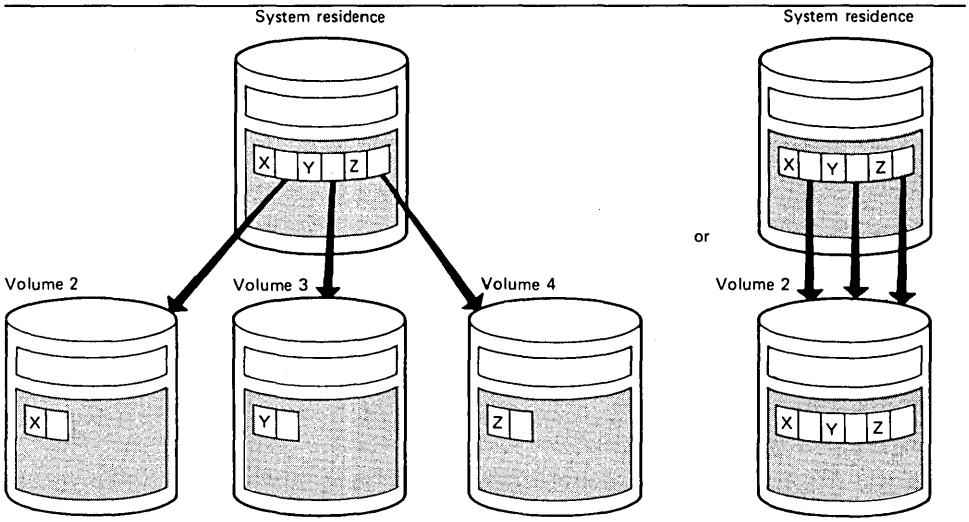


Figure 59. Connecting Three Volumes Using IEHPROGM

Building and Maintaining a Generation Index

IEHPROGM can be used to build an index structure for a generation data group and to control the action to be taken if the index overflows.

The lowest level index in the structure can contain up to 255 entries for successive generations of a data set. If the index overflows, the oldest entry is removed from the index, unless otherwise specified (in which case all entries are removed). If desired, the program can be used to scratch all generation data sets whose entries are removed from the index.

Figure 60 shows the index structure created for generation data group A.B.C. In this example, provision is made for up to five subsequent entries in the lowest level index.

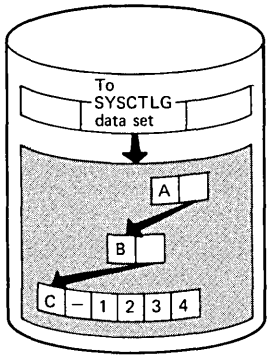


Figure 60. Building a Generation Index Using IEHPROGM

Note: Before a generation data group can be cataloged as such, a generation index must exist. Otherwise, a generation data set is cataloged as an individual data set, rather than as a generation.

When creating and cataloging a generation data set, the user can provide necessary DCB information. See "Providing DCB Attributes" in "Appendix D: Generation Data Groups" for a discussion of how DCB attributes are provided for a generation data group.

Maintaining Data Set Passwords

IEHPROGM can be used to maintain password entries in the PASSWORD data set and to alter the protection status of direct access data sets in the data set control block (DSCB). For a complete description of data set passwords and the PASSWORD data set *OS Data Management for System Programmers*, GC28-6550.

A data set can have one of three types of password protection, as indicated in the DSCB for direct access data sets and in the tape label for tape data sets (see *OS System Control Blocks*, GC28-6628, for a description of the DSCB and tape label). 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.

Note: If a system data set is password protected and a problem occurs on a system 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 61 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.

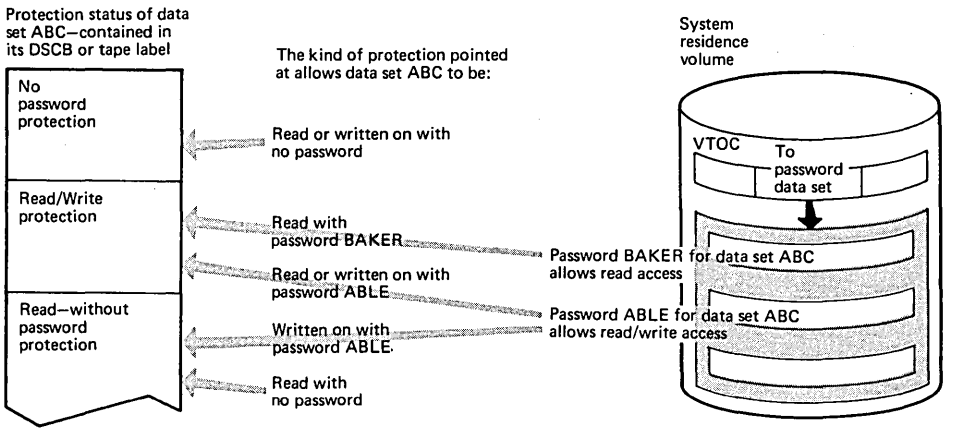


Figure 61. Relationship Between the Protection Status of a Data Set and Its Passwords

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. Control and secondary passwords are discussed under "Adding Data Set Passwords" below.

For direct access 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 direct access 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 control password for a data set is the initial password entry in the PASSWORD data set for that data set name.)
- The data set is on line.
- 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 via job control language.

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.

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.

If this is the first password added to the PASSWORD data set for a data set, the password 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. 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 a direct access, online data set is added, the protection status of the data set (read/write or read-without-password) is set in the DSCB. However, the data set to be protected must not be allocated within the same job as the one in which IEHPROGM is executed. If it is allocated, the DSCB cannot be accessed and the protection status is not set. If the data set to be protected is being created within the same job, use job control language to set the protection status 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 access counter of the password entry is set to zero when any information in the entry is replaced. The protection status of a data set can be changed by replacing the control entry for the data set.

If the control entry of a direct access, online data set is replaced, the DSCB is also reset to indicate any change in the protection status of the data set. Therefore, the user should ensure that the volume is on line when changing the protection status of a direct access 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, direct access 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 direct access data set, the user should ensure that the volume is on line. If the volume is not on line, 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 user must change the protection status in the tape label to indicate no protection; otherwise, the data set cannot be accessed. The tape label may be changed using the IEHINITT utility program.

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. Figure 62 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
```

Figure 62. Listing of a Password Entry

Input and Output

IEHPROGM uses as input a control data set that contains utility control statements used to control the functions of the program and to indicate those data sets or volumes that are to be modified.

IEHPROGM produces as output a modified object data set or volume(s) and a message data set, which contains any error messages and information from the PASSWORD data set.

IEHPROGM provides a return code to indicate the results of program execution. The return codes and their meanings are:

- 00, which indicates successful completion.
- 04, which indicates that a syntax error was found in the name field of the control statement or in the PARM field in the EXEC statement. Processing is continued.
- 08, which indicates that a request for a specific operation was ignored because of an invalid control statement or an otherwise invalid request. The operation is not performed.
- 12, which indicates that an input/output error was detected when trying to read from or write to SYSPRINT, SYSIN or the VTOC.
- 16, which indicates an unrecoverable error. The job step is terminated.

Control

IEHPROGM is controlled by job control statements and utility control statements.

Job control statements are used to:

- Execute or invoke the program.
- Define the control data set.
- Define volumes and/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 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 53 shows the job control statements necessary for using IEHPROGM.

Table 53. IEHPROGM Job Control Statements

Statement	Use
JOB	Initiates 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 line density on the output listing and to suppress printing of utility control statements. See "PARM Information on the EXEC Statement" below.
SYSPRINT DD	Defines a sequential message data set.
anyname1 DD	Defines a permanently mounted volume. (The system residence volume is considered to be a permanently mounted volume.)
anyname2 DD	Defines a mountable device type.
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.

The minimum region size that can be specified for the execution of IEHPROGM is 44K.

The anyname1 DD statement can be entered:

```
//anyname1 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 deletion of a data set. The anyname1 DD statement is arbitrarily assigned the ddname DD1 in the IEHPROGM examples.

The anyname2 DD statement can be coded in the following ways:

```
//anyname2 DD VOLUME = SER = xxxxxx,UNIT = xxxx,DISP = OLD

//anyname2 DD VOLUME = (PRIVATE,SER = xxxxxx),
//             UNIT = (xxxx,,DEFER),DISP = OLD
```

The second example can be used to specify deferred mounting when a large number of magnetic tapes or direct access volumes are to be processed in one application of the program. The anyname2 DD statement is arbitrarily assigned the ddname DD2 in the IEHPROGM examples. DD statements defining additional mountable devices are assigned names DD3, DD4, etc.

Refer to "Appendix C: DD Statements for Defining Mountable Devices" for instructions on defining mountable volumes.

Restrictions

- The block size for the SYSPRINT (message) data set must be a multiple of 121. The block size for the SYSIN (control) data set must be a multiple of 80. Any blocking factor can be specified for these block sizes.
- With the exception of the SYSIN and SYSPRINT DD statements, all DD statements in Table 53 are used as device allocation statements, rather than as true data definition statements. Because IEHPROGM modifies the internal control blocks created by device allocation DD statements, these statements must not include the DSNAME parameter. (All data sets are defined explicitly or implicitly by utility control statements.)
- One anyname1 DD statement must be included for each permanently mounted volume referred to in the job step.
- One anyname2 DD statement must be included for each mountable device to be used in the job step.
- When IEHPROGM is dynamically invoked in a job step containing a program other than IEHPROGM, the DD statements defining mountable devices for IEHPROGM must be included in the job stream prior to DD statements defining data sets required by the other program.
- For MVT applications, DD statements defining mountable devices must appear in the same order in the input stream as the utility control statements that refer to volumes mounted on those devices.
- When VOL = device = list specifies multiple volumes, no more than 50 volumes may be specified.

PARM Information on the EXEC Statement

Additional information can be specified in the PARM parameter of the EXEC statement to control line density on the output listing and to suppress printing of utility control statements. The EXEC statement can be coded:

```
//EXEC PGM = IEHPROGM[,PARM = 'LINECNT = xx,  {PRINT   }']
                                              {NOPRINT }
```

The LINECNT parameter specifies the number of lines per page in the listing of the SYSPRINT data set; xx is a 2-digit number, from 01 through 99. If LINECNT is omitted, or if an error is encountered in the LINECNT subparameter, the number of lines per page will be 45.

The PRINT value specifies that the utility control statements are to be written to the SYSPRINT data set. If neither PRINT nor NOPRINT is coded, PRINT is assumed.

The NOPRINT value specifies that utility control statements are not to be written to the SYSPRINT data set. 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.

Utility Control Statements

IEHPROGM is controlled by the following utility control statements:

- SCRATCH statement, which is used to scratch a data set or a member from a direct access volume.
- RENAME statement, which is used to change the true name or alias of a data set or member residing on a direct access volume.
- CATLG statement, which is used to generate an entry in the index of a catalog.
- UNCATLG statement, which is used to remove an entry from the lowest level index of the catalog.
- BLDX statement, which is used to create a new index in the catalog.
- DLTX statement, which is used to remove a low level index from the catalog.
- BLDA statement, which is used to assign an alias to an index at the highest level of the catalog.
- DLTA statement, which is used to deiete an alias previously assigned to an index at the highest level of the catalog.
- CONNECT statement, which is used to place an entry into an index at the highest level of the catalog.
- RELEASE statement, which is used to remove an entry from the high level index of a volume.
- BLDG statement, which is used to build an index for a generation data group and to establish the action to be taken should the index overflow.
- ADD statement, which is used to add a password entry in the PASSWORD data set.
- REPLACE statement, which is used to replace information in a password entry.
- DELETEP statement, which is used to delete an entry in the PASSWORD data set.
- LIST statement, which is used to format and print information from a password entry.

SCRATCH Statement

The SCRATCH statement is used to scratch a data set or member from a direct access volume. A data set or member is scratched only from the volumes designated in the SCRATCH statement. This function does not uncatalog scratched data sets.

The format of the SCRATCH statement is:

```
[label] SCRATCH {DSNAME = name }
                {VTOC          }
                ,VOL = device = list
                [,PURGE]
                [,MEMBER = name]
                [,SYS]
```

where:

DSNAME = *name*
specifies the fully-qualified name of either the data set to be scratched or the partitioned data set that contains the member to be scratched.

VTOC
specifies that all data sets on the specified volume, except those protected by a password or those whose expiration dates have not expired, are to be scratched. 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**.

VOL = *device* = *list*
specifies the volume or volumes that contain the data set or sets to be scratched. If **VTOC** is specified, **VOL** cannot specify more than one volume. Caution should be used when specifying **VTOC** if **VOL** specifies the system residence volume.

PURGE
specifies that each data set specified by **DSNAME** or **VTOC** be scratched, even if its expiration date has not elapsed. If **PURGE** is omitted, the specified data sets are scratched only if their expiration dates have elapsed.

MEMBER = *name*
specifies a member name or alias of a member to be removed from the directory of a partitioned data set. If **MEMBER** is omitted, the entire data set or volume of data sets is scratched.

SYS
specifies that data sets that are to be scratched have names that begin with "AAAAAAAA.AAAAAAAA.AAAAAAAA.AAAAAAAA." or "SYSnnnnn.T and F or V in position 19. These are names assigned to data sets by the operating system. This parameter is valid only when **VTOC** is specified.

If the name of the data set to be scratched begins with SYS, nnnn is the date and F or V in position 19 represents MFT or MVT.

When executing a SCRATCH operation, care should be taken to ensure that the data set or volume is not being used by a program executing concurrently.

RENAME Statement

The RENAME statement is used to change the true name or alias of a data set or member residing on a direct access volume. The name is changed only on the designated volume(s). The rename operation does not update the catalog.

The format of the RENAME statement is:

```
[label] RENAME  DSNAME = name
                ,VOL = device = list
                ,NEWNAME = name
                [,MEMBER = name]
```

where:

DSNAME = *name*
specifies the fully-qualified name of the data set to be renamed or specifies the data set that contains the member to be renamed.

VOL = *device* = *list*
specifies the volume or volumes that contain the data set or member whose name is to be changed. If **MEMBER** is specified, **VOL** cannot specify more than one volume.

NEWNAME = *name*
specifies the new fully-qualified name for the data set, or the new member or alias.

MEMBER = *name*
specifies the member name or alias for a member (in the named data set) that is to be renamed. If **MEMBER** is omitted, the specified data set name is changed.

CATLG Statement

The CATLG statement is used to generate an entry in the index of a catalog. If additional levels of indexes are required in the catalog, this function automatically creates them. When cataloging generation data sets, refer to "BLDG (Build Generation Index) Statement" for the action to be taken when the index is full.

The format of the CATLG statement is:

```
[label] CATLG  DSNAME = name
               ,VOL = device = list
               [,CVOL = device = serial]
```

where:

DSNAME = *name*
specifies the fully-qualified name of the data set to be cataloged. The qualified name must not exceed 44 characters, including delimiters.

VOL = *device* = *list*
specifies the volume or volumes that contain the data set to be cataloged. For either a sequential data set or an indexed sequential data set, 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. All serial numbers specified in **VOL** must represent the same device type.

CVOL = *device* = *serial*
specifies the device type and volume serial number of the control volume on which the catalog search for the index is to begin. If **CVOL** is omitted, the system residence volume is assumed. If the volumes are connected at the highest level of the index and the control volume is mounted, **CVOL** need not be specified.

Note: When *device* is represented by a group name (for example, SYSDA) instead of a generic name (for example, 2314 or 2400) in the **VOL** parameter, the catalog operation does not enter the device type code in the system catalog. Instead, it places a unique entry in the device type field of the catalog. The allocation of the device for this entry may not be satisfactory to the user. The generic name should be used if the group name was generated for one or more device types. When the system is subsequently generated, this entry may no longer be valid; that is, all such group name entries should be uncataloged and then recataloged after a subsequent generation of the system.

When cataloging data sets residing on tape, specify the data set sequence number and the volume serial number, as follows:

VOL = device = ({serial,seqno},...)

If a data set is created on a 9-track dual-density tape drive (2400-4), the data set can be cataloged with a device specification of 2400 for an 800 bits per inch tape or 2400-3 for a 1600 bits per inch tape. If a device specification of 2400-4 is made when the data set is cataloged, any subsequent retrieval of that data set is made on a dual-density drive.

UNCATLG Statement

The UNCATLG statement is used to remove an entry from the lowest level index of the catalog. This function does not delete higher level indexes from the index structure.

The format of the UNCATLG statement is:

```
[label] UNCATLG  DSNAME = name
                 [,CVOL = device = serial]
```

where:

DSNAME = *name*
specifies the fully-qualified name of the data set to be uncataloged.

CVOL = *device* = *serial*
specifies the device type and volume serial number of the control volume at which the search for the catalog entry is to begin. If **CVOL** is omitted, the system residence volume is assumed. If catalogs are properly connected at the highest level of the index and the control volume is mounted, **CVOL** need not be specified.

BLDX (Build Index) Statement

The BLDX statement is used to create a new index in the catalog. If the creation of an index requires that higher level indexes be created, this function automatically creates them.

The format of the BLDX statement is:

```
[label] BLDX  INDEX = name
              [,CVOL = device = serial]
```

where:

INDEX = *name*
specifies the qualified name of the index to be created. The qualified name must not exceed 44 characters, including delimiters.

CVOL = *device* = *serial*
specifies the device type and volume serial number of the control volume on which the search for the index is to begin. If **CVOL** is omitted, the system residence volume is assumed.

DLTX (Delete Index) Statement

The DLTX statement is used to remove an index from the catalog. Only an index that has no entries can be removed.

The format of the DLTX statement is:

```
[label] DLTX  INDEX = name
              [,CVOL = device = serial]
```

where:

INDEX = *name*
specifies the qualified name of the index to be deleted.

CVOL = *device* = *serial*
specifies the device type and volume serial number of the control volume on which the search for the index is to begin. If **CVOL** is omitted, the system residence volume is assumed.

Because this function does not delete higher level indexes, it must be used repetitively to delete an entire structure. For example, to delete index structure A.B.C, delete index A.B.C, index A.B, and index A.

BLDA (Build Index Alias) Statement

The BLDA statement is used to assign an alias to an index at the highest level of the catalog.

The format of the BLDA statement is:

```
[label] BLDA  INDEX = name
              ,ALIAS = name
              [,CVOL = device = serial]
```

where:

INDEX = *name*
specifies the unqualified index to which an alias name is to be assigned.

ALIAS = *name*
specifies an unqualified name to be assigned as the alias.

CVOL = *device* = *serial*
specifies the device type and volume serial number of the control volume on which the catalog entry is to be made. If **CVOL** is omitted, the system residence volume is assumed.

DLTA (Delete Index Alias) Statement

The DLTA statement is used to delete an alias previously assigned to an index at the highest level of the catalog.

The format of the DLTA statement is:

```
[label] DLTA  ALIAS = name
              [,CVOL = device = serial]
```

where:

ALIAS = *name*
specifies the unqualified index alias to be deleted.

CVOL = *device* = *serial*
specifies the device type and volume serial number of the control volume containing the catalog entry to be deleted. If **CVOL** is omitted, the system residence volume is assumed.

CONNECT Statement

The CONNECT statement is used to place an entry in the high level index of the catalog. The entry identifies a second volume by its device type and volume serial number. In addition, it contains an index name identifying the index to be searched for (during subsequent index searches) on the second volume.

Note: This function does not create an index on the second volume.

The format of the CONNECT statement is:

```
[label] CONNECT  INDEX = name
                 ,VOL = device = serial
                 [,CVOL = device = serial]
```

where:

INDEX = *name*
specifies the index name to be entered in the high level index on the first volume.

VOL = *device* = *serial*
specifies the device type and volume serial number of the second volume. This information is placed in the high level index on the first volume.

CVOL = *device* = *serial*
specifies the device type and serial number of the first volume. If **CVOL** is omitted, the system residence volume is assumed to be the first volume.

The CONNECT statement does not create a SYSCTLG data set on the connected volume. Before cataloging the first data set on a connected volume, the user must define a SYSCTLG data set on that volume. This can be done with the following DD statement:

```
//ddname DD  DSNAME=SYSCTLG,UNIT=xxxx,DISP=(,KEEP),
//             SPACE=(CYL,1),VOLUME=SER=xxxxxx
```

If a job requires an auxiliary control volume to complete a catalog search, the user need not have the auxiliary control volume mounted before the job is begun. (The user does not have to remember the volume on which a particular data set is cataloged.) The system directs the operator to mount an auxiliary control volume if it is needed. However, the auxiliary control volume must be connected to the system residence volume by means of the CONNECT verb, as modified for Release 17. If an auxiliary control volume was connected before Release 17, release the auxiliary control volume for all high level indexes on the system residence volume that point to that volume, and then use the current CONNECT verb to reconnect the auxiliary control volume with the system residence volume.

RELEASE (Disconnect) Statement

The RELEASE statement is used to remove an entry from the high level index of a volume. This effectively disconnects a second volume from the first volume. The RELEASE statement does not delete an index from the second volume.

The format of the RELEASE statement is:

```
[label] RELEASE  INDEX=name
                 [,CVOL=device=serial]
```

where:

INDEX = *name*
specifies the index name to be removed from the high level index of the first volume.

CVOL = *device* = *serial*
specifies the device type and volume serial number of the first volume. If **CVOL** is omitted, the system residence volume is assumed to be first volume.

BLDG (Build Generation Index) Statement

The BLDG statement is used to build an index for a generation data group, and to establish the action to be taken should the index overflow.

The format of the BLDG statement is:

```
[label] BLDG  INDEX=name
              ,ENTRIES=n
              [,CVOL=device=serial]
              [,EMPTY]
              [,DELETE]
```

where:

INDEX = *name*
specifies the 1- to 35-character name of the generation index.

ENTRIES = *n*
specifies the number of entries to be contained in the generation index; *n* must not exceed 255.

CVOL = *device* = *serial*
specifies the device type and volume serial number of the volume on which the catalog search for the index is to begin. If **CVOL** is omitted, the system residence volume is assumed.

EMPTY
specifies that all entries be removed from the generation index when it overflows. This effectively uncatalogs all of the generation data sets. If **EMPTY** is omitted, the entries with the largest generation numbers will be maintained in the catalog when the generation index overflows.

DELETE
specifies that generation data sets are to be scratched after their entries are removed from the index. If **DELETE** is omitted, the data sets are not scratched.

ADD (Add a Password) Statement

The ADD statement is used to add a password entry in the PASSWORD data set. When the control entry for a direct access, online data set is added, the indicated protection status is set in the DSCB; when a secondary entry is added, the protection status in the DSCB is not changed.

The format of the ADD statement is:

```
[label] ADD  DSNAME = name
             [,PASWORD2 = new-password]
             [,CPASWORD = control-password]
             [,TYPE = code]
             [,VOL = device = list]
             [,DATA = 'user-data']
```

where:

DSNAME = *name*
specifies the fully-qualified name of the data set to which the password is to be assigned.

PASWORD2 = *new-password*
specifies the password to be added. The password can consist of one- to eight-alphameric characters. If **PASWORD2** is omitted, the operator is prompted for a new password.

CPASWORD = *control-password*
specifies the control password for the data set. The control password must be specified unless this is the first password assigned to the data set.

TYPE = *code*
specifies the protection code of the password and, if a control password is being assigned to a direct access, online data set, specifies the protection status of the data set. 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. 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, read/write protection is set in the DSCB.

2
specifies that the password is to allow only read access to the data set; if a control password is being assigned, 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, read-without-password protection is set in the DSCB.

VOL = *device* = *list*
specifies the direct access volume or volumes that contain the data set to be protected. If omitted, the protection status of the data set is not set in the DSCB, unless the data set is cataloged. This parameter is not necessary for secondary password entries or if the desired protection status is already set in the DSCB.

DATA = *'user-data'*
specifies that user data is to be included in the password entry. The user data must be in single quotes and must not exceed 77 characters.

REPLACE (Replace a Password) Statement

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 a direct access, online 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 format of the REPLACE statement is:

```
[label] REPLACE  DSNAME = name
                 [,PASWORD1 = current-password]
                 [,PASWORD2 = new-password]
                 [,CPASWORD = control-password]
                 [,TYPE = code]
                 [,VOL = device = list]
                 [,DATA = 'user-data']
```

where:

DSNAME = *name*
specifies the fully-qualified name of the data set whose password entry is to be changed.

PASWORD1 = *current-password*
specifies the current password in the entry to be changed. If **PASWORD1** is omitted, the operator is prompted for the current password.

PASWORD2 = *new-password*
specifies the new password to be 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 one- to eight-alphameric characters. If **PASWORD2** is omitted, the operator is prompted for a new password.

CPASWORD = *control-password*
specifies the control password for the data set whose entry is to be changed. The control password must be specified unless the control entry is being changed. If the control entry is to be changed, the control password must be specified as **PASWORD1**.

TYPE = *code*
specifies the protection code of the password and, if a control password entry is to be changed for a direct access, online data set, specifies the protection status of the data set. If this parameter is omitted, the protection is not changed. 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 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 a control password is being 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 changed, read-without-password protection is set in the DSCB.

VOL = *device* = *list*
specifies the direct access volume or volumes that contain the data set whose protection status is to be changed. If omitted, the protection status of the data set is not changed in the DSCB, unless the data set is cataloged. This parameter is not necessary for secondary password entries or if the protection status of the data set is not to be changed.

DATA = '*user-data*'
specifies that user data is to be included in the password entry. The user data must be in single quotes and must not exceed 77 characters. If this parameter is omitted, the 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 a direct access, online data set is deleted, the protection status in the DSCB is set to indicate that the data set is no longer protected.

The format of the DELETEP statement is:

```
[label] DELETEP  DSNAME = name
                 [,PASWORD1 = current-password]
                 [,CPASWORD = control-password]
                 [,VOL = device = list]
```

where:

DSNAME = *name*
specifies the fully-qualified name of the data set whose password is to be deleted.

PASWORD1 = *current-password*
specifies the password to be deleted.

CPASWORD = *control-password*
specifies the control password for the data set whose password is to be deleted. The control password must be specified unless the control password is to be deleted. If the control password is to be deleted, the control password must be specified as **PASWORD1**.

VOL = *device* = *list*
specifies the direct access volume or volumes that contain the data set whose password is to be deleted. If omitted, the protection status of the data set is not changed in the DSCB, unless the data set is cataloged. This parameter is not necessary if a secondary password is to be deleted.

LIST (List Information from a Password) Statement

The LIST statement is used to format and print information from a password entry.

The format of the LIST statement is:

```
[label] LIST DSNAME = name
          ,PASWORD1 = current-password
```

where:

DSNAME = *name*
specifies the fully-qualified name of the data set whose password entry is to be listed.

PASWORD1 = *current-password*
specifies the password in the entry to be listed.

IEHPROGM Examples

The following examples illustrate some of the uses of IEHPROGM. Table 54 can be used as a quick reference guide to IEHPROGM examples. The numbers in the "Example" column point to the examples that follow.

Table 54. IEHPROGM Example Directory

Operation	Mountable Volumes	Comments	Example
SCRATCH	2314 Disk	VTOC is to be scratched.	1
SCRATCH UNCATLG, and DLTX	2314 Disk	Two data sets are to be scratched and uncataloged. An index structure is to be deleted from SYSCTLG.	2
RENAME, UNCATLG, DLTX, and CATLG	2314 Disks	A data set is to be renamed on two mountable devices; the old data set name and index structure are to be removed from the catalog. The data set is cataloged under its new name. Object data set resides on two mountable devices	3
UNCATLG and DLTX	2314 Disk	Three data sets are to be uncataloged; their supporting index structures are to be deleted from the catalog.	4
CONNECT and CATLG	2314 Disk	Connect system residence volume to a second volume. Catalog data sets on second volume. SYSCTLG was previously defined on the second volume.	5
BLDG	None	A generation index is to be built.	6
BLDG and CATLG	None	A generation index is to be built and three data sets are to be cataloged.	7
RELEASE and CONNECT	None	Auxiliary control volume is released and connected.	8
RENAME, DELETEP, and ADD	2314 Disk	The object data set exists on one mountable device.	9
LIST and REPLACE	2314 Disk	The object data set exists on two mountable devices.	10
RENAME	2314 Disk	Rename a member of a partitioned data set.	11

Note: In the IEHPROGM examples, the EXEC statement and the SYSPRINT DD statement can be replaced with the following job control statement:

```
//        EXEC PROC=MOD
```

which invokes the following IBM-supplied cataloged procedure:

```
//MOD EXEC    PGM=IEHPROGM,REGION=44K
//DDSRV DD    VOLUME=REF=SYS1.SVCLIB,DISP=OLD
//SYSPRINT DD SYSOUT=A
```

IEHPROGM Example 1

In the following example, data sets are to be scratched from the volume table of contents of a mountable volume. Because the system residence volume is not referred to, no DD1 DD statement is necessary in the job stream.

The example follows:

```
//SCRVTOC  JOB  09#550,BROWN
//         EXEC PGM=IEHPROGM
//SYSPRINT DD   SYSOUT=A
//DD2      DD   UNIT=2314,VOLUME=SER=231400,DISP=OLD
//SYSIN    DD   *
      SCRATCH   VTOC,VOL=2314=231400
/*
```

The SCRATCH statement, used in this example, indicates that all data sets (including those beginning with AAAAAA.AAAAAA.AAAAAA.AAAAAA) whose expiration dates have expired are to be scratched from the specified volume.

IEHPROGM Example 2

In this example, two data sets are to be scratched: SET1 is to be scratched on volume 231401, and A.B.C.D.E is to be scratched on volume 231402. Both data sets are to be uncataloged, and index structure A.B.C.D is to be deleted from the SYSCTLG data set. Because the system residence volume, which resides on a 3330 volume, is referred to through use of the UNCATLG and DLTX statements, a DD statement is included in the input stream.

The example follows:

```
//SCRDSETS JOB  09#550,BROWN
//         EXEC PGM=IEHPROGM
//SYSPRINT DD   SYSOUT=A
//DD1      DD   UNIT=3330,VOLUME=SER=111111,DISP=OLD
//DD2      DD   UNIT=2314,DISP=OLD,VOLUME=SER=(231400)
//SYSIN    DD   *
      SCRATCH   DSNAME=SET1,VOL=2314=231401
      UNCATLG   DSNAME=SET1
      SCRATCH   DSNAME=A.B.C.D.E,VOL=2314=231402
      UNCATLG   DSNAME=A.B.C.D.E
         DLTX   INDEX=A,B.C.D
         DLTX   INDEX=A.B.C
         DLTX   INDEX=A.B
         DLTX   INDEX=A
/*
```

IEHPROGM Example 3

In this example, the name of a data set is to be changed on two mountable volumes. The old data set name and index structure are to be removed from the catalog and the data set is to be cataloged under its new data set name.

The example follows:

```
//RENAMEDS JOB  09#550,BROWN
//         EXEC PGM=IEHPROGM
//SYSPRINT DD   SYSOUT=A
//DD1      DD   VOLUME=SER=111111,UNIT=3330,DISP=OLD
//DD2      DD   UNIT=(2314,,DEFER),DISP=OLD,
// VOLUME=(PRIVATE,SER=(231400,231401))
//SYSIN    DD   *
   RENAME  DSNAME=A.B.C,NEWNAME=NEWSET,                    ℛC
               VOL=2314=(231400,231401)
  UNCATLG  DSNAME=A.B.C
     DLTX  INDEX=A.B
     DLTX  INDEX=A
    CATLG  DSNAME=NEWSET,VOL=2314=(231400,231401)
/*
```

IEHPROGM Example 4

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 to be uncataloged and their supporting index structures deleted from the catalog. The system residence volume resides on a 2314 volume.

The example follows:

```
//DLTSTRUC JOB  09#550,BROWN
//         EXEC PGM=IEHPROGM
//SYSPRINT DD   SYSOUT=A
//DD1      DD   UNIT=2314,VOLUME=SER=111111,DISP=OLD
//SYSIN    DD   *
      UNCATLG   DSNAME=A.B.C.D.E.F.SET1
      UNCATLG   DSNAME=A.B.C.G.H.SET2
      UNCATLG   DSNAME=A.B.I.J.K.SET3
         DLTX   INDEX=A.B.I.J.K
         DLTX   INDEX=A.B.I.J
         DLTX   INDEX=A.B.I
         DLTX   INDEX=A.B.C.G.H
         DLTX   INDEX=A.B.C.G
         DLTX   INDEX=A.B.C.D.E.F
         DLTX   INDEX=A.B.C.D.E
         DLTX   INDEX=A.B.C.D
         DLTX   INDEX=A.B.C
         DLTX   INDEX=A.B
         DLTX   INDEX=A
/*
```

IEHPROGM Example 5

In this example, the system residence volume, which resides on a 3330 volume, is to be connected to a second volume. Any subsequent index search for index level X, Y, or Z will be carried to the second volume.

The example follows:

```
//CONNECT  JOB  09#550,BROWN
//         EXEC PGM=IEHPROGM
//SYSPRINT DD   SYSOUT=A
//DD1      DD   UNIT=3330,VOLUME=SER=111111,DISP=OLD
//DD2      DD   UNIT=2314,VOLUME=SER=231400,DISP=OLD
//SYSIN    DD   *
  CONNECT   INDEX=X,VOL=2314=231400
  CONNECT   INDEX=Y,VOL=2314=231400
  CONNECT   INDEX=Z,VOL=2314=231400
    CATLG   DSNAME=X.BB.CCC,VOL=2314=231401,CVOL=2314=231400
    CATLG   DSNAME=Y.BB.CC,VOL=2314=231401,CVOL=2314=231400
    CATLG   DSNAME=Z.BB.XT,VOL=2314=231401,CVOL=2314=231400
/*
```

The control statements are discussed below:

- The CONNECT statements identify the second volume. The specified index names, along with the volume identification, are placed on the system residence volume.
- The CATLG statements catalog three data sets (X.BB.CCC, Y.BB.CC, and Z.BB.XT) on the second volume. Because the volumes are connected before the catalog operations are performed, the **CVOL** parameters are not required in the CATLG statements; they are included to bypass the index search on the system residence volume.

IEHPROGM Example 6

In this example, a generation index is to be built in the catalog. The system residence volume resides on a 3330 volume.

The example follows:

```
//BLDGDGIX JOB  09#550,BROWN
//         EXEC PGM=IEHPROGM
//SYSPRINT DD   SYSOUT=A
//DD1      DD   UNIT=3330,VOLUME=SER=111111,DISP=OLD
//SYSIN    DD   *
         BLDG   INDEX=A.B.C,ENTRIES=10,EMPTY
/*
```

The BLDG statement specifies the generation data group A.B.C and makes provision for ten entries in the index. All entries are to be removed from the index when it overflows.

IEHPROGM Example 7

In this example, a generation index is to be built and three data sets are to be cataloged in the index. The system residence volume resides on a 3330 volume.

The example follows:

```
//CTLGDG   JOB  09#550,BROWN
//         EXEC PGM=IEHPROGM
//SYSPRINT DD   SYSOUT=A
//DD1      DD   UNIT=3330,VOLUME=SER=111111,DISP=OLD
//SYSIN    DD   *
          BLDG  INDEX=A.B.C,ENTRIES=20,EMPTY
        CATLG   DSNAME=A.B.C.G0001V00,VOL=2314=231400
        CATLG   DSNAME=A.B.C.G0002V00,VOL=2314=231400
        CATLG   DSNAME=A.B.C.G0003V00,VOL=2314=231400
/*
```

Figure 63 shows the index structure after the three generation data sets are cataloged.

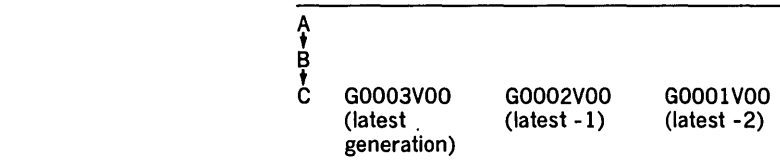


Figure 63. Index Structure After Generation Data Sets Are Cataloged

IEHPROGM Example 8

In this example, the RELEASE and CONNECT statements are used to disconnect the control volume, 231400, from the system residence catalog for the high level index A and to reconnect that same control volume for that index. This technique is necessary only if the auxiliary control volume was connected before Release 17. The system residence volume resides on a 2314 volume.

The example follows:

```
//RECONCT  JOB  09#550,BROWN
//         EXEC PGM=IEHPROGM
//SYSPRINT DD   SYSOUT=A
//DD1      DD   UNIT=2314,VOLUME=SER=111111,DISP=OLD
//SYSIN    DD   *
      RELEASE   INDEX=A
      CONNECT   INDEX=A,VOL=2314=231400
/*
```

IEHPROGM Example 9

In this example, a data set is to be renamed. The data set passwords assigned to the old data set name are to be deleted. Then two passwords are to be assigned to the new data set name.

Note: If the data set is not cataloged, a message indicating that the LOCATE macro instruction failed is issued. The return code is 8.

The example follows:

```
//ADDPASS  JOB  09#550,BROWN
           EXEC PGM=IEHPROGM,PARM='NOPRINT'
//SYSPRINT DD   SYSOUT=A
//DD1      DD   VOLUME=(PRIVATE,SER=231400),DISP=OLD,
// UNIT=(2314,,DEFER)
//SYSIN    DD   *
 RENAME DSNAME=OLD,VOL=2314=231400,NEWNAME=NEW
 DELETEP DSNAME=OLD,PASWORD1=KEY
 ADD DSNAME=NEW,PASWORD2=KEY,TYPE=1,                          %C
               DATA='SECONDARY IS READ'
 ADD DSNAME=NEW,PASWORD2=READ,CPASWORD=KEY,TYPE=2,            %C
               DATA='ASSIGNED TO J. DOE'
/*
```

The control statements are discussed below:

- DELETEP specifies that the entry for the password KEY is to be 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 to be 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.

Note: The operator is required to supply a password to rename the old data set.

IEHPROGM Example 10

In this example, information from a password entry is to be listed. Then the protection mode of the password, the protection status of the data set, and the user data are to be changed.

The example follows:

```
//REPLPASS JOB  09#550,BROWN
         EXEC PGM=IEHPROGM,PARM='NOPRINT'
//SYSPRINT DD   SYSOUT=A
//DD1      DD   UNIT=3330,VOLUME=SER=111111,DISP=OLD
//DD2      DD   VOLUME=(PRIVATE,SER=(231400,231401)),
// UNIT=(2314,,DEFER),DISP=OLD
//SYSIN    DD   *
     LIST   DSNAME=A.B.C,PASWORD1=ABLE
  REPLACE   DSNAME=A.B.C,PASWORD1=ABLE,                          %C
               PASWORD2=ABLE,TYPE=3,                             %C
               VOL=2314=(231400,231401),                         %C
               DATA='NO SECONDARIES; ASSIGNED TO DEPT 31'
/*
```

The control statements are discussed below:

- LIST specifies that the access counter, protection mode, and user data from the entry for password ABLE are to be 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 to be changed to write-only protection. The **VOL** parameter is required because the protection status of the data set is being changed and the data set, in this example, is not cataloged. Because this is a control password, the **CPASWORD** parameter is not required.

IEHPROGM Example 11

In this example, a member of a partitioned data set is to be renamed.

The example follows:

```
//REN    JOB  09#550,BROWN
//       EXEC PGM=IEHPROGM
//DD1    DD   VOL=SER=231411,DISP=OLD,UNIT=2314
//SYSIN  DD *
 RENAME  VOL=2314=231411,DSNAME=DATASET,NEWNAME=BC,MEMBER=ABC
/*
```

The control statements are discussed below:

- DD1 DD defines a permanently mounted volume.
- SYSIN DD defines the input data set, which immediately follows in the input stream.
- RENAME specifies that member ABC in the partitioned data set DATASET, which resides on a 2314 volume, is to be renamed BC.

IFHSTATR is a system utility used to select, format, and write information from type 21 (error statistics by volume) records when System Management Facilities (SMF) has been system generated into the system. (See "Introduction" for general system utility information.)

Figure 64 shows the format of the type 21 record.

4	Bytes of Record Descriptor Word			
0	System Indicator	Record Type	Time of Day	
4	Time of Day (continued)		Current Date	
8	Current Date (continued)		System Identification	
12	System Identifier		Length of rest of record including this field	
16	Volume Serial Number			
20	Volume Serial No. (cont.)		22	Channel/Unit Address
24	UCB Type			
28	Temporary Read Errors	Temporary Write Errors	Start I/O's	
32	Permanent Read Errors	Permanent Write Errors	Noise Blocks	Erase Gaps
36	Erase Gaps (continued)	Cleaner Actions		Tape Density
40	Block Size		Reserved	

Figure 64. Type 21 (ESV) Record Format

Error statistics by volume (ESV) records should be retrieved from the IFASMFDP tape or from SYS1.MAN (on tape). ESV can also be retrieved directly from SYS1.MANX or SYS1.MANY (on a direct access storage device); however, IFHSTATR does not clear the SYS1.MANX (or SYS1.MANY) data set and make it available for additional records.

Input and Output

IFHSTATR uses as input type 21 records, which contain information about errors on magnetic tape. IFHSTATR processes only type 21 records; if none are found, a message is written to the output data set.

IFHSTATR produces as output an output data set, which contains information selected from type 21 records. The output takes the form of 121-byte unblocked records, with an ASA control character in the first byte of each record.

Figure 65 shows a sample of printed output from IFHSTATR.

```
VOLUME          CPU  MOD   TIME     CHANNEL  TEMP  TEMP  PERM  PERM  NOISE  ERASE  CLEANER  USAGE  TAPE     BLOCK
SERIAL  .DATE    ID   NO  OF DAY    / UNIT   READ  WRITE READ  WRITE BLOCKS GAPS   ACTIONS (SIO's) DENSITY  LENGTH

001021 69/309   BB   40 15:55:07     181      1     0     0     0     1      0        0      10     0800       80
001022 69/309   AA   40 15:56:02     184     10     0     0     0     0      0        0      28     1600      121
000595 69/309   CC   50 15:56:20     283      0    10     0     0     0     10        0      28     0800       50
```

Figure 65. Sample Output from IFHSTATR

Control

IFHSTATR is controlled by job control statements. Utility control statements are not used.

Job Control Statements

Table 55 shows the job control statements necessary for using IFHSTATR.

Table 55. IFHSTATR Job Control Statements

Statement	Use
JOB	Initiates 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, UNIT, VOLUME, LABEL, DCB, and DISP parameters should be included.
SYSUT2 DD	Defines the sequential data set on which the output is to be written.

The minimum region size that can be specified for the execution of IFHSTATR is 4K.

The output data set can reside on any output device supported by BSAM.

Note: The 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.

IFHSTATR Example

This example shows the JCL needed to produce a report.

The example follows:

```
//          JOB
//          EXEC PGM=IFHSTATR
//SYSUT1    DD   UNIT=2400,DSNAME=SYS1.MAN,LABEL=( ,SL),
// VOLUME=SER=VOLID,DISP=OLD
//SYSUT2    DD   SYSOUT=A
/*
```

Appendix A: Exit Routine Linkage

Utility programs can be linked to user-supplied exit routines for additional processing.

Linking to an Exit Routine

Linking to an exit routine from a utility program is accomplished in one of the following ways:

- If the exit routine is for label processing or totaling, or if the exit routine is specified in the IEBTCRIN program by **OUTREC** or **ERROR**, linkage is performed by the BALR instruction.
- In all other cases, linkage is performed by using the LINK macro instruction.

The LINK macro instruction contains the symbolic name of the entry point of an exit routine and, if required, a list of parameters.

For further information on the use of the LINK macro instruction, see *OS Supervisor Services Guide*, GC28-6646, and *OS Supervisor & Data Management Macro Instructions*, GC28-6647.

At the time of the linkage operation:

- General register 1 contains the starting address of the parameter list, or contains zero to indicate end-of-file on the input data set for the IEBTCRIN **OUTREC** or **ERROR** exits.
- General register 13 contains the address of the register save area. This save area must not be used by user label processing routines or by the IEBGENER input/output error exit routines. See "Appendix E: Processing User Labels."
- General register 14 contains the address of the return point in the utility program.
- General register 15 contains the address of the entry point to the exit routine.

Registers 1 through 14 must be restored before control is returned to the utility program.

The exit routine must be contained in either the job library or the link library.

The parameter lists passed to label processing routines and parameter lists passed to nonlabel processing routines are described in the topics that follow.

Label Processing Routine Parameters

The parameters passed to a user's label processing routine are addresses of the 80-byte label buffer, the DCB being processed, the status information if an uncorrectable 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 the user's label processing routine. When the utility program has been requested to generate labels, the label processing routine constructs 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's routine. In such a case, the user's input label processing routine is entered with the buffer address parameter set to zero.

The format and content of the DCB are presented in *OS Supervisor & Data Management Macro Instructions*, GC28-6647.

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 MOD data set are being processed (when the data set is opened).

If an uncorrectable 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 low order three bytes of this parameter contain the address of standard status information as supplied for SYNAD routines. (The SYNAD routine is not entered.)

Nonlabel Processing Routine Parameters

Table 56 shows the program from which exits can be taken to nonlabel processing routines, the names of the exits, and the parameters available for each exit routine.

Table 56. 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).
IEBDG	OUTREC	Address of output record.
IEBCOMPR	ERROR	Address of DCB for SYSUT1; address of DCB for SYSUT2.[1]
	PRECOMP	Address of SYSUT1 record; length of SYSUT1 record, address of SYSUT2 record; length of SYSUT2 record.
IEBPTPCH	INREC	Address of input record; length of the input record.
	OUTREC	Address of output record; length of output record.
IEBTCRIN	ERROR	Address of the error record; address of a full word which contains the record length.
	OUTREC	Address of the normal record; address of a full word which contains the record length.

[1] The IOBAD pointer in the DCB points to a location that contains the address of the corresponding data event control block (DECB) for these records. The format of the DECB is illustrated as part of the BSAM READ macro instruction in *OS Supervisor & Data Management Macro Instructions*, GC28-6647.

Returning from an Exit Routine

An exit routine returns control to the utility program by means of the macro instruction in the exit routine.

The format of the RETURN macro instruction is:

```
[label] RETURN [(r1,r2)]
               ,RC = {n  }
                     {15 }
```

where:

(*r1*,*r2*)
specifies the range of registers to be reloaded by the utility program from the register save area. If this parameter is omitted, the registers are considered properly restored by the exit routine.

RC =
specifies a return code in register 15. If **RC** is omitted, register 15 is loaded as specified by (*r1*,*r2*). These values can be coded:

n
specifies a return code to be placed in the 12 low-order bits of register 15.

15
specifies that general register 15 already contains a valid return code.

The user's label processing routine must return a code in register 15 as shown in Table 57 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's label processing routine was entered after an uncorrectable output error occurred. In this case the system attempts to resume normal processing.

Table 57 shows the return codes that can be issued to utility programs by user exit routines. Slightly different return codes are used for the UPDATE = INPLACE option of the IEBUPDTE program. See the discussion of UPDATE = INPLACE in the chapter "IEBUPDTE Program."

Table 57. Return Codes Issued by User Exit Routines

Type of Exit	*Return Code*	*Action*
Input Header or Trailer Label	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 terminated on request of the user routine.
Output Header or Trailer Label	0	The system resumes normal processing. No label is 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 terminated on request of the user routine.
Totaling Exits	0	Processing continues, but no further 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 terminated.
All other exits (except IEBTCRIN's ERROR and OUTREC and IEBPTPCH's exit OUTREC)	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 terminated and this return code is passed to the calling processor.
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). IEBTCRIN removes the EDW from an edited MTDI record before processing continues.
	16	Utility program is terminated.
OUTREC (IEBTCRIN)	0	Record is not placed in normal output data set.
	4	Record is placed in normal output data set (SYSUT2).
	16	Utility program is terminated.
OUTREC (IEBPTPCH)	4	Record is not placed in output data set. The return code is not passed to the calling processor.
	12 or 16	Utility program is terminated and this return code is passed to the calling processor.
	Any other	Record is placed in the output data set.
	Number	The return code is not passed to the calling processor.

Further information on the use of the RETURN macro instruction is contained in *OS Data Management Services Guide*, GC26-3746, and *OS Supervisor & Data Management Macro Instructions*, GC28-6647.

For a list of return codes issued by IEBTCRIN at job termination, see the "IEBTCRIN Program" chapter of this publication.

Appendix B: Invoking Utility Programs from a Problem Program

Utility programs can be invoked by a problem program through the use of the ATTACH or LINK macro instruction. In addition, IEBTCRIN can be invoked through the use of the LOAD or CALL macro instruction.

The problem program must supply the following to the utility program:

- The information usually specified in the PARM parameter of the EXEC statement.
- The ddnames of the data sets to be used during processing by the utility program.

Note: When IEHMOVE, IEHPROGM, or IEHLIST is dynamically invoked in a job step containing a program other than one of these three, the DD statements defining mountable devices for the IEHMOVE, IEHPROGM, or IEHLIST program must be included in the job stream prior to DD statements defining data sets required by the other program.

LINK or ATTACH Macro Instruction

The LINK or ATTACH macro instruction can be used to invoke a utility program from a problem program.

The format of the LINK or ATTACH macro instruction is:

```
[name]  {LINK  }EP = progname
        {ATTACH}
                ,PARAM = (optionaddr[,ddnameaddr][,hdingaddr])
                ,VL = 1
```

where:

EP = *progname*
specifies the symbolic name of the utility program.

PARAM =
specifies, as a sublist, address parameters to be passed from the problem program to the utility program. These values can be coded:

optionaddr
specifies the address of an option list, which is usually specified in the PARM parameter of the EXEC statement. This address must be written for all utility programs.

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. If it is the last parameter, it may be omitted.

hdingaddr
specifies the address of a six-byte list, HDNGLIST, which contains an EBCDIC page count for the output device. If *hdingaddr* is omitted, the page number defaults to 1.

VL = 1
specifies that the sign bit of the last fullword of the address parameter list is to be set to 1.

Figure 66 shows these lists as they exist in the user's DC area. Note that the symbolic starting addresses for OPTLIST and DDNMELST fall on halfword boundaries. Note also the alternative ddnames INSTREAM, INPUTSET, and WHICHPTR.

The PARAM parameter of the LINK macro instruction in the calling program provides the utility program with the symbolic addresses of the parameter lists shown in Figure 66, as follows:

- The option list, OPTLIST, which includes the number of bytes in the list (hexadecimal 08) and the NOVERIFY option.
- The alternate ddname list, DDNMELST, which includes the number of bytes in the list (hexadecimal 48) and alternative names for the SYSIN, SYSUT1, and SYSUT2 data sets.
- The heading list, HDNGLIST, which includes the number of bytes in the list (hexadecimal 04) and indicates the starting page number (shown as 10) for printing operations controlled through the SYSPRINT data set.

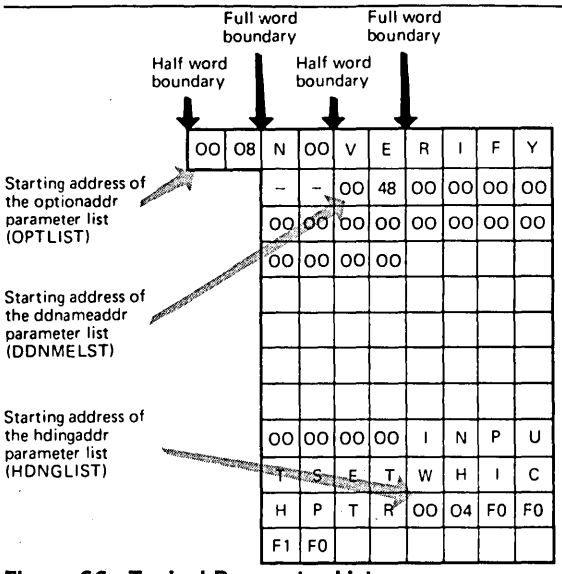


Figure 66. Typical Parameter Lists

The option list, OPTLIST, must begin on a halfword boundary that is not also a fullword boundary. The two high order bytes contain a count of the number of bytes in the remainder of the list. (For all programs except IEHMOVE, IEHPROGM, IEHINITT, IEBISAM, and IEBUPDTE, the count must be zero.) OPTLIST is free form with fields separated by commas. No blanks or zeros should appear in the list.

The ddname list, DDNMELST, must begin on a halfword boundary that is not also a fullword boundary. The two high order bytes contain a count of the number of bytes in the remainder of the list. Each name of fewer than eight bytes must be left aligned and padded with blanks. If an alternate ddname is omitted from the list, the standard name is assumed. If the name is omitted within the list, the eight-byte entry must contain binary zeros. Names can be omitted from the end by merely shortening the list. Table 58 shows the sequence of the eight-byte entries in the ddname list pointed to by *ddnameaddr*.

The first two bytes of HDNGLIST contain the length in bytes of the heading list. The remaining four bytes contain a page number that the utility program is to place on the first page of printed output.

LOAD Macro Instruction

IEBTCRIN can be invoked through use of the LOAD macro instruction.

The LOAD macro instruction causes the control program to bring the load module containing the specified entry point into main storage unless a copy is already there. Control is not passed to the load module.

Table 58. Sequence of DDNMELST Entries

Entry	*Standard Name*
1	00000000
2	00000000
3	00000000
4	00000000
5	SYSIN
6	SYSPRINT
7	00000000
8	SYSUT1
9	SYSUT2
10	SYSUT3
11	SYSUT4

The format of the LOAD macro instruction is:

```
[name] LOAD {EP = IEBTCRIN          }
            {EPLOC = address of name}
```

where:

EP = IEBTCRIN
is the entry point name of the program to be brought into main storage.

EPLOC = *address of name*
is the main storage address of the entry point name described above.

CALL Macro Instruction

The CALL macro instruction can be used to pass control to IEBTCRIN after IEBTCRIN has been loaded into main storage.

Control can be passed to IEBTCRIN via a CALL macro instruction or via a branch and link instruction. If the branch and link instruction is used, register 1 must be loaded with the address of a parameter list of full words as described under "LINK or ATTACH Macro Instruction." The last parameter list address must contain X'80' in byte 1 to indicate the last parameter in the list.

The format of the CALL macro instruction is:

```
[name]CALL IEBTCRIN(,optionaddr[,ddnameaddr][,hdingaddr]),VL
```

where:

IEBTCRIN
is the name of the entry point to be given control; the name is used in the macro instruction as the operand of a V-type address constant.

optionaddr
specifies the address of an option list, OPTLIST, usually specified in the PARM parameter of the EXEC statement. This address must be written for all utility programs.

ddnameaddr
specifies the address of a list of alternate ddnames, DDNMELST, 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. If it is the last parameter, it may be omitted.

hdingaddr
specifies the address of a six-byte list containing an EBCDIC page count for the output device.

VL
specifies that the high order bit of the last address parameter in the macro expansion is to be set to 1.

The option list, OPTLIST, must begin on a halfword boundary that is not also a fullword boundary. The two high order bytes contain a count of the number of bytes in the remainder of the list. (For all programs except IEHMOVE, IEHPROGM, IEHINITT, and IEBISAM, the count must be zero.) The option list is free form with fields separated by commas. No blanks or zeros should appear in the list.

The ddname list, DDNMELST, must begin on a halfword boundary that is not also a fullword boundary. The two high order bytes contain a count of the number of bytes in the remainder of the list. Each name of fewer than eight bytes must be left aligned and padded with blanks. If an alternate ddname is omitted from the list, the standard name is assumed. If the name is omitted within the list, the eight-byte entry must contain binary zeros. Names can be omitted from the end by merely shortening the list. The sequence of the eight-byte entries in the ddname list pointed to by *ddnameaddr* is shown earlier in Table 58.

The first two bytes of the heading list, HDNGLIST, contain the length in bytes of the heading list. The remaining four bytes contain a page number that the utility program is to place on the first page of printed output.

Appendix C: DD Statements for Defining Mountable Devices

When defining mountable devices to be used by system utility programs IEHPROGM, IEHMOVE, IEHLIST, or IEHDASDR, the user must consider the implications of the DD statements he uses to define those devices.

DD statement parameters must ensure that no one else has access to either the volume or the data set. Caution should be used when altering volumes that are permanently resident or reserved (for example, volumes containing system data sets, non-demountable volumes, and volumes reserved through the PRESRES option).

Under normal conditions, a mountable device should not be *shared* with another job step; that is, if a utility program is used to update a volume on a mountable device, the volume being updated must remain mounted until the operation is completed.

Following are ways to ensure that mountable devices are not shared:

- Specify DEFER in a DD statement defining a mountable device.
- Specify unit affinity on a second DD statement defining a mountable device.
- Specify a volume count in the VOLUME parameter of a DD statement that is greater than the number of mountable devices to be allocated.
- Specify PRIVATE in a DD statement defining a mountable device.

For a detailed discussion, see *OS JCL Reference*, GC28-6704.

DD Statement Examples

In the following examples of DD statements, an IBM 2314 Disk Storage Device is indicated as the mountable device. Alternative parameters are stacked.

DD Example 1

This DD statement makes a specific request for a private, nonsharable volume or volumes to be mounted on a single 2314 device.

The example follows:

```
//DD1   DD   UNIT=(2314,,DEFER),DISP=(,KEEP),
//             VOLUME=(PRIVATE,SER=(123456))
```

A utility program causes a mount message to be issued for a specific volume when the volume is required for processing by the program. The user should supply the operator with the clearly marked volume or volumes to be mounted during the job step.

This DD statement ensures that the volume integrity of a mountable volume is maintained. If only one volume is to be processed, it is mounted at the start of the job step and dismounted at the end of the step. If additional volumes are processed, they are mounted and dismounted when needed by the utility program. The last volume to be processed is dismounted at the end of the job step.

DD Example 2

This DD statement makes a request for a private, nonsharable volume.

The example follows:

```
//DD2   DD   UNIT=(2314,,DEFER),VOLUME=PRIVATE,DISP=(NEW,KEEP)
```

The results of this statement are identical to those shown in DD Example 1.

If a specific unit is requested and the volume serial number is not given in the DD statement, the user must be certain that either: (1) the desired volume is already mounted on that unit or (2) a volume is not mounted, causing the system to issue a mount message.

Note: This statement can be used only if the user is certain that a removable volume, rather than a fixed volume, will be allocated by the scheduler. If there is any chance that a fixed volume will be allocated, this statement must not be used.

DD Example 3

This DD statement makes a specific request for a private, sharable volume to be mounted on a 2314 device.

The example follows:

```
//DD1 DD UNIT=2314,VOLUME=(PRIVATE,SER=(121212)),DISP=OLD
```

This DD statement does not ensure that volume integrity is maintained. It should be used with extreme caution. A concurrently running job step might make a specific request for the volume, use the volume, and demount it.

DD Example 4

This DD statement makes a specific request for a public, nonsharable volume to be mounted on a 2314 device.

The example follows:

```
//DD3 DD UNIT=(2314,,DEFER),VOLUME=SER=789012,DISP=(,OLD)
```

If the volume is already mounted, it is used. The volume remains mounted at the end of the job step, and is not demounted until another job step requires the device on which the volume is mounted.

This DD statement ensures that volume integrity is maintained between jobs; two or more such statements in a single job can allocate the same device.

DD Example 5

This DD statement makes a specific request for a public, sharable volume to be mounted on a 2314 device.

The example follows:

```
//DD1  DD  UNIT=2314,VOLUME=SER=654321,DISP=OLD
```

If the volume is already mounted, it is used. The volume remains mounted at the end of the job step, and is not demounted until another job step requires the device on which the volume is mounted. (This DD statement can also be used to define permanently resident devices.)

This DD statement does not ensure that the volume integrity of a mountable volume is maintained. It should be used with extreme caution because there is the possibility that a job step running concurrently might use the device.

Appendix D: Generation Data Groups

A generation data group is a group of related cataloged data sets. The manner in which these data sets are cataloged is what makes them a generation data group. Within a generation data group, the generations can have like or unlike DCB attributes and data set organizations. If the attributes and organizations of all generations in a group are identical, the generations can be retrieved together as a single data set. Each data set within a generation data group is called a generation data set. Generation data sets are sometimes called *generations*.

There are advantages to grouping related data sets. Because the catalog management routines can refer to the information in a special index—called a *generation index*—in the catalog:

- All of the data sets in the group can be referred to by a common name.
- The operating system is able to keep the generations in chronological order.
- Outdated or obsolete generations can be automatically deleted by the operating system.

The management of a generation data group depends upon the fact that generation data sets have sequentially ordered names—absolute and relative names—that represent their age. The absolute generation name is the representation used by the catalog management routines in the catalog. Older data sets have smaller absolute numbers. The relative name is a signed integer used to refer to the latest (0), next to the latest (-1), etc. generation. The relative number can also be used to catalog a new generation (+1).

Absolute Generation and Version Numbers

An absolute generation and version number is used to identify a specific generation of a generation data group. The generation and version numbers are in the form GxxxxVyy, where xxxx is an unsigned four-digit decimal generation number and yy is an unsigned two-digit decimal version number. For example:

- A.B.C.G0000V00 is generation data set zero, version zero in the generation data group A.B.C.
- A.B.C.G0001V00 is generation data set one, version zero in generation data group A.B.C.
- A.B.C.G0009V01 is generation data set nine, version one in generation data group A.B.C.

The number of new generations and versions is limited by the number of digits in the absolute generation name, that is, 9999 for generations and 99 for versions.

The generation number is automatically maintained by the system. The number of generations kept depends on the size of the generation index. For example, if the size of the index allows ten entries, the ten latest generations may be maintained in the index.

The version number allows you to perform normal data set operations without disrupting the management of the generation data group. For example, if you want to update the second generation in a three-generation index, replace generation two, version zero, with generation two, version one. Only one version is kept per generation.

A generation can be cataloged using either absolute or relative numbers. When a generation is cataloged, a generation and version number is placed as a low level entry in the generation index. In order to catalog a version number other than V00, you must use an absolute generation and version number.

Figure 67 shows how the index looks when three generations are cataloged. Note that the generation index is a pushdown list, which allows you to use relative generation numbers when cataloging or retrieving a generation.

Note: A new version of a specific generation can be cataloged automatically by specifying the old generation number along with a new version number. For example, if generation A.B.C.G0005V00 is cataloged in the index and you now create and catalog A.B.C.G0005V01, the new entry is cataloged in the index location previously occupied by A.B.C.G0005V00. This process removes the old entry from the catalog but does not scratch the old version. To scratch the old version and make its space available for reallocation, a DD card, describing the data set to be deleted, with

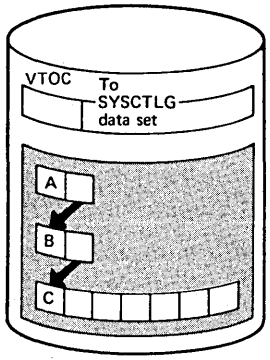


Figure 67. Generation Index—Three Entries

DISP = (OLD,DELETE) should be included at the time the data set is to be replaced by the new version.

Relative Generation Numbers

As an alternative to using absolute generation and version numbers when cataloging or referring to a generation, you can use a relative generation number. To specify a relative number, use the generation data group name followed by a negative integer, a positive integer, or a zero, enclosed in parentheses. For example, A.B.C(-1), A.B.C(+1), or A.B.C(0).

The value of the specified integer tells the operating system what generation number to assign to a new generation, or it tells the system the location (in the generation index) of an entry representing a previously cataloged generation.

When you use a relative generation number to catalog a generation, the operating system assigns an absolute generation number and a version number of V00 to represent that generation. The absolute generation number assigned depends on the number last assigned and the value of the relative generation number that you are now specifying. For example, if in a previous job generation A.B.C.G0005V00 was the last generation cataloged and you specify A.B.C(+1), the generation now cataloged is assigned the number G0006V00. Though any positive relative generation number can be used, a number greater than 1 may cause absolute generation numbers to be skipped.

When you use a relative generation number to refer to a generation that was cataloged in a previous job, the relative number has the following meaning:

- A.B.C(0) refers to the latest existing cataloged entry.
- A.B.C(-1) refers to the next to latest entry, etc.

When cataloging is requested:

- A relative number refers to the same generation throughout a job.
- A job step that terminates abnormally may be deferred for a later step restart. If the step cataloged a generation data set via JCL, you must change all relative generation numbers in the succeeding steps via JCL before resubmitting the job.

For example, if the succeeding steps contained the relative generation numbers:

- A.B.C(+1), which refers to the entry cataloged in the terminated job step.
- A.B.C(0), which refers to the next to latest entry.
- A.B.C(-1), which refers to the third latest entry, etc.

You must change them as follows before the step can be restarted: A.B.C(0), A.B.C(-1), A.B.C(-2), etc.

Note: New nonspecific generation data group requests are cataloged with a volume serial number of X'FF4040404040' if they are not opened, so that data set integrity is maintained and an incorrect generation is not retrieved.

Figure 68 shows how an index looks after three generations—A.B.C(+1), A.B.C(+1), and A.B.C(+2)—have been cataloged in three separate jobs. The first generation is assigned the generation number G0001V00; the second, G0002V00; the third, G0004V00.

Building a Generation Index

A generation data group is managed via the information found in a generation index. To build a generation index, use the BLDG function of the IEHPROGM utility program. The BLDG function builds the index, providing lower level entries for as many generations (up to 255) as needed in a generation data group. The BLDG function

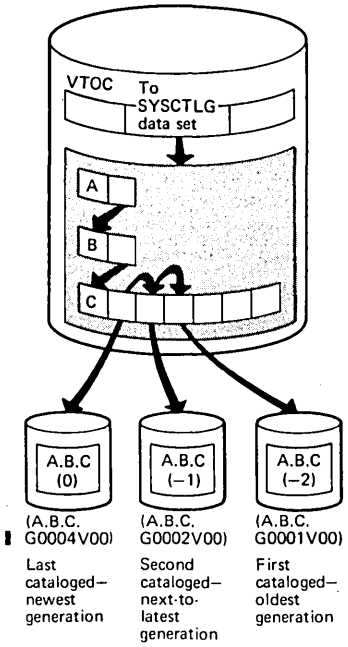


Figure 68. Relative Positioning—Three Entries in the Catalog

also indicates how older or obsolete generations are to be handled when the index is full. For example, when the index is full, you may wish to empty it, scratch existing generations, and begin cataloging a new series of generations.

Note: An alias cannot be assigned to the highest level of a generation index.

Figure 69 shows a generation index. When the index was built, provision was made for the subsequent cataloging of ten generations.

After the index is built, a generation can be cataloged by its generation data group name and either an absolute generation and version number or a relative generation number.

Examples showing how to build a generation index included in "Cataloging a Generation" in this appendix.

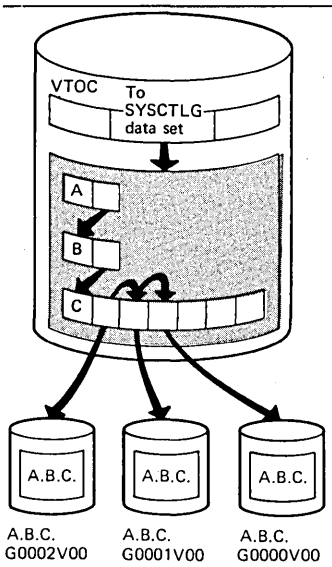


Figure 69. Generation Index

Creating a New Generation

To create a new generation data set:

- Allocate the generation.
- Catalog the generation.

Allocating a Generation

To take full advantage of the facilities of the system, the allocation can be patterned after a previously allocated generation in the same group. This is accomplished by the specification of DCB attributes for the new generation as described below.

If you are using absolute generation and version numbers, DCB attributes for a generation can be supplied directly in the DCB parameter of the DD statement defining the generation to be created and cataloged.

If you are using relative generation numbers to catalog generations, DCB attributes can be supplied either: (1) by creating a model DSCB on the volume on which the index resides (the volume containing the SYSCTLG data set) or (2) by referring to a cataloged data set for the use of its attributes. Attributes can be supplied before you catalog a generation, when you catalog it, or at both times, as follows:

1. Create a model DSCB on the volume on which your index resides. You can provide initial DCB attributes when you create your model; however, you need not provide any attributes at this time. Initial or overriding attributes can be supplied when you create and catalog a generation.[1] To create a model DSCB, include the following DD statement in the job step that builds the index or in any other job step that precedes the step in which you create and catalog your generation:

   ```
   //name DD   DSNAME = datagrpname,DISP = (,KEEP),SPACE = (TRK,(0)),
   //           UNIT = yyyy,VOLUME = SER = xxxxxx,
   //           DCB = (applicable subparameters)
   ```

   The DSNAME is the common name by which each generation is identified; xxxxxx is the serial number of the volume containing the catalog. If no DCB subparameters are desired initially, you need not code the DCB parameter.

2. You do not need to create a model DSCB if you can refer to a cataloged data set whose attributes are identical to those you desire or to an existing model DSCB for which you can supply overriding attributes. A cataloged data set referred to in this manner must reside on the same volume as your index. To refer to a cataloged data set for the use of its attributes, specify DCB = (dsname) on the DD statement that creates and catalogs your generation. To refer to an existing model, specify DCB = (modeldscbname, your attributes) on the DD statement that creates and catalogs your generation.

Cataloging a Generation

A generation can be cataloged through the use of normal job control language procedures or through the use of IEHPROGM.

Using JCL Procedures to Catalog a Generation

Assuming that a generation index has been built and that provisions have been made for supplying DCB attributes, a generation is created and cataloged in the same manner as any other type of data set.

When you use relative numbers in job control language procedures, you must include the CATLG subparameter in the DD statement defining the new generation. When you use absolute generation and version numbers, you need not catalog the new generation immediately.

Using IEHPROGM to Catalog a Generation

The CATLG function of IEHPROGM can be used to catalog a generation. Again, the prerequisite for cataloging a generation is the existence of a generation index in the SYSCTLG data set.

Note: You must always use an absolute generation and version number to catalog or uncatalog a generation using IEHPROGM. (IEHMOVE and IEHLIST also require that absolute generation and version numbers be used.)

Creating an ISAM Data Set as Part of a Generation Data Group

To create an indexed sequential data set as part of a generation data group, you must: (1) create the indexed sequential data set separately from the generation group and (2) use IEHPROGM to put the indexed sequential data set into the generation group.

Use the RENAME function to rename the data set. Then use the CATLG function to catalog the data set. For instance, if MASTER is the name of the generation data group, and GggggVvv is the absolute generation name, you would code the following:

```
RENAME DSNAME = ISAM,VOL = 2314 = SCRTCH,NEWNAME = MASTER.GggggVvv
CATLG DSNAME = MASTER.GggggVvv,VOL = 2314 = SCRTCH
```

---

[1] Only one model DSCB is necessary for any number of generations. If you plan to use only one model, do not supply DCB attributes when you create the model. When you subsequently create and catalog a generation, include necessary DCB attributes in the DD statement referring to the generation. In this manner, any number of generation data groups can refer to the same model.

Retrieving a Generation

A generation is retrieved through the use of job control language procedures. Any operation that can be applied to a non-generation data set can be applied to a generation. For example, a generation can be updated and reentered in the catalog, or it can be copied, printed, punched, or used in the creation of new generation or non-generation data sets.

You can retrieve a generation by using either relative generation numbers or absolute generation and version numbers.

Because two or more jobs can compete for the same resource, generation data groups should be updated with caution, as follows:

1. No two jobs running concurrently should refer to the same generation data group. As a partial safeguard against this situation, use absolute generation and version numbers when cataloging or retrieving a generation in a multiprogramming environment. If you use relative numbers, a job running concurrently may update the generation index, perhaps cataloging a new generation which you will then retrieve in place of the one you wanted.
2. Even when using absolute generation and version numbers, a job running concurrently might catalog a new version of a generation or perhaps delete the generation you wished to retrieve. For this reason, some degree of control should be maintained over the execution of job steps referring to generation data groups.

Generation Data Group Examples

The following examples show some of the ways in which generations can be created and cataloged or retrieved and used as source data in the creation of new generation or non-generation data sets.

Generation Example 1

In this example, an IEHPROGM job step, STEPA, creates a model DSCB and builds a generation index. STEPB, an IEBGENER job step, creates and catalogs a sequential generation from card input.

The example follows:

```
//BLDINDX  JOB
//STEPA    EXEC PGM=IEHPROGM
//SYSPRINT DD   SYSOUT=A
//BLDDSCB  DD   DSNAME=A.B.C,DISP=(,KEEP),SPACE=(TRK,0),
// DCB=(LRECL=80,RECFM=FB,BLKSIZE=800),
// VOLUME=SER=111111,UNIT=2314
//SYSIN    DD   *
         BLDG INDEX=A.B.C,ENTRIES=10,EMPTY,DELETE
/*
//STEPB    EXEC PGM=IEBGENER
//SYSPRINT DD   SYSOUT=A
//SYSIN    DD   DUMMY
//SYSUT2   DD   DSNAME=A.B.C(+1),UNIT=2314,DISP=(,CATLG),
// VOLUME=SER=231400,SPACE=(TRK,(20))
//SYSUT1   DD   DATA
(input card data)
/*
```

The control statements are discussed below:

- BLDDSCB DD creates a model DSCB on the system residence volume.
- SYSIN DD indicates that a utility control statement (BLDG) is included next in the input stream.
- BLDG specifies the generation group name A.B.C and makes provision for ten lower level entries in the index. When the 11th entry is to be entered, the index is emptied and all of the generations are deleted.
- SYSUT2 DD defines an output sequential generation. The generation is assigned the absolute generation and version number G0001V00 in the index.
- SYSUT1 DD defines the input card data set.

Any subsequent job that causes the deletion of the generations should include DD statements defining the devices on which the volumes containing those generations are to be mounted. Each generation for which no DD statement is included is uncataloged at that time, but not deleted.

After the generation data group is emptied, new generations continue to be assigned generation numbers according to the last generation number assigned before the empty operation. To restart the numbering operation (that is, to reset to G0000V00 or G0001V00), it is necessary to uncatalog all the old generation data sets and then rename and recatalog, beginning with G0000V00.

Generation Example 2

In this example, a second generation is created and cataloged in the index built in Example 1. DCB attributes are included to override those attributes that were specified when the model DSCB was created.

The example follows:

```
//          JOB
//          EXEC PGM=IEBGENER
//SYSPRINT DD   SYSOUT=A
//SYSIN    DD   DUMMY
//SYSUT2   DD   DSNAME=A.B.C(+1),UNIT=2314,DISP=(,CATLG),
// DCB=(LRECL=80,RECFM=FB,BLKSIZE=1600),
// VOLUME=SER=231401,SPACE=(TRK,(20))
//SYSUT1   DD   DATA
```

(input data set)

```
/*
```

The control statements are discussed below:

- SYSUT2 DD defines an output sequential generation. The generation is assigned the absolute generation and version number G0002V00 in the index. The specified DCB attributes override those initially specified in the model DSCB. The DCB attributes specified when the model DSCB was created remain unchanged; that is, those attributes are applicable when you catalog a succeeding generation unless you specify overriding attributes at that time.
- SYSUT1 DD defines the input card data set.

Generation Example 3

In this example, a generation index for generation data group A.B.C is built. Three existing noncataloged, non-generation data sets are renamed; the renamed data sets are cataloged as generations in the generation index.

The example follows:

```
//BLDINDEX JOB
//          EXEC PGM=IEHPROGM
//SYSPRINT DD   SYSOUT=A
//DD1      DD   UNIT=2314,VOLUME=SER=111111,DISP=OLD
//DD2      DD   UNIT=(2314,,DEFER),DISP=OLD,
// VOLUME=(PRIVATE,,SER=(231400))
//SYSIN    DD   *
  BLDG     INDEX=A.B.C,ENTRIES=10
  RENAME   DSNAME=DATASET1,VOL=2314=231400,                    ⟟C
               NEWNAME=A.B.C.G0001V00
  RENAME   DSNAME=DATASET2,VOL=2314=231400,                    ⟟C
               NEWNAME=A.B.C.G0002V00
  RENAME   DSNAME=DATASET3,VOL=2314=231400,                    ⟟C
               NEWNAME=A.B.C.G0003V00
  CATLG    DSNAME=A.B.C.G0001V00,VOL=2314=231400
  CATLG    DSNAME=A.B.C.G0002V00,VOL=2314=231400
  CATLG    DSNAME=A.B.C.G0003V00,VOL=2314=231400
/*
```

The control statements are discussed below:

- DD1 DD defines the system residence volume, on which the SYSCTLG (system catalog) data set resides.
- BLDG specifies the generation group name A.B.C and makes provision for ten entries in the index. The oldest generation is to be uncataloged when the index becomes full. No generations are to be scratched.
- The RENAME statements rename three non-generation data sets residing on a 2314 disk volume.
- CATLG catalogs the renamed data sets in the generation index.

Note: Because the DCB parameters were supplied when the non-generation data sets were created, no DCB parameters are now specified; therefore, no model DSCB is required.

Generation Example 4

In this example, a non-generation version of a generation data set is to be made. The generation is represented as the next to latest entry in the index. The name of the resultant data set is TESTSET. This example assumes that the generation to be copied is partitioned.

The example follows:

```
//COPY     JOB
//         EXEC PGM=IEBCOPY
//SYSPRINT DD   SYSOUT=A
//SYSUT1   DD   DSNAME=A.B.C(-1),DISP=OLD
//SYSUT2   DD   DSNAME=TESTSET,UNIT=2314,DISP=(,KEEP),
// VOLUME=SER=231400,SPACE=(TRK,(20,10,5)),
///  DCB=(RECFM=FB,LRECL=80,BLKSIZE=80)
//SYSIN    DD   DUMMY
/*
```

The control statements are discussed below:

- SYSUT1 DD defines the generation from which a copy is to be made.
- SYSUT2 DD defines a partitioned data set (TESTSET) on a 2314 output volume. The DCB attributes in this statement are identical to those assigned to the generation. (Reblocking is permitted, but the SYSUT2 block size specification must be a multiple of the original block size.)

Generation Example 5

In this example, a partitioned generation, consisting of three members, is to be used as source data in the creation of a new generation. IEBUPDTE is to be used to add a fourth member to the three source members and to number the new member. The resultant data set is to be cataloged as a new generation.

The example follows:

```
//         JOB
//         EXEC PGM=IEBUPDTE,PARM=MOD
//SYSPRINT DD   SYSOUT=A
//SYSUT1   DD   DSNAME=A.B.C(0),DISP=OLD
//SYSUT2   DD   DSNAME=A.B.C(+1)DISP=(,CATLG),UNIT=2314,
// VOLUME=SER=231400,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,IGNORE=EOF
./  NUMBER    NEW1=10,INCR=5
```

(data cards comprising MEM4)

```
./  ENDUP
/*
```

The control statements are discussed below:

- SYSUT1 DD defines the latest generation, which is used as source data.
- SYSUT2 DD defines the new generation, which is created from the source generation and from an additional member included as input card data.
- The **REPRO** Function statements reproduce the named source members in the output generation.
- The **ADD** Function statement specifies that the data cards 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 incremented by 5.
- ENDUP signals the end of input card data.

Note: This example assumes that a model DSCB exists on the catalog volume on which the index was built.

Appendix E: Processing User Labels

User labels can be processed by IEBGENER, IEBCOMPR, IEBPTPCH, IEHMOVE, IEBCTRIN, and IEBUPDTE. In some cases, user-label processing is automatically performed; in other cases, you must indicate the processing to be performed. In general, user label support allows the utility program user to:

- Process user labels as data set descriptors.
- Process user labels as data.
- Total the processed records prior to each WRITE command (IEBGENER and IEBUPDTE only).

For either of the first two options, the user must specify standard labels (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.

The user cannot update labels by means of the IEBUPDTE program. This function must be performed by a user's label processing routines. IEBUPDTE will, however, allow you to create labels on the output data set from data supplied in the input stream. See the discussion of the LABEL statement in the chapter "IEBUPDTE Program."

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. See the chapter "IEHMOVE Program."

Volume switch labels of a multivolume data set cannot be processed by IEHMOVE, IEBGENER, or IEBUPDTE. Volume switch labels are therefore 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 the user's label processing routines receives control for each user label of the specified type. The user's 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 EXIT keyword parameters indicate that a user routine should receive control each time the OPEN, EOV, or CLOSE routine encounters a user label of the type specified.

Figure 70 illustrates the action of the system at OPEN, EOV, or CLOSE time. When OPEN, EOV, 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. The user's routine processes the label and returns control, along with a return code, to the utility program. The utility program then returns control to OPEN, EOV, or CLOSE.

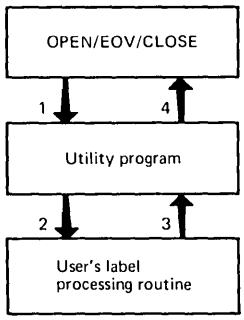


Figure 70. System Action at OPEN, EOV, or CLOSE Time

This cycle is repeated up to eight times, depending upon the number of user labels in the group and the return codes supplied by the user's routine.

Exiting to a User's Totaling Routine

When an exit is taken to a user's totaling routine, an output record is passed to the user's 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 the user's routine. If the user has 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.

Note: 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.

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. The user can have his labels printed or punched by IEBPTPCH, 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's 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 LABEL statement is in the label buffer, and the user's routine can modify it.

The code returned by the user's totaling routine determines system response as follows:

- 0, which specifies that processing is to continue, but no further exits are to be taken.
- 4, which specifies that normal processing is to continue.
- 8, which specifies that processing is to terminate, except for EOD processing on the output data set (user label processing).
- 16, which specifies that processing is to be terminated.

Index

Indexes to systems reference library manuals are consolidated in *IBM System/360 Operating System: Systems Reference Library Master Index*, GC28-6644. For additional information about any subject listed below, refer to other publications listed for the same subject in the Master Index.

Note: If more than one page number is given, the primary discussion is listed first. The entries in the index appear the same way they appear in the body of the book, which means that entries with bold type or italic type in the body of the book appear in bold type or italic type in the index.

[ ] 7
{ } 7

A

absolute generation and version number 289
action on return codes 281
action (IEBDG) 87
ADD 168
ADD statement 269
adding data set passwords 262
adding new member to a symbolic library 167
ALIAS statement
  for IEBUPDAT 162
  for IEBUPDTE 174
alias names
  listed by IEHLIST 218
  processed by IEBCOPY 53
ALLOCATE module, changing or replacing 215
allocating space
  with the IEBCOPY program 57
  with the IEHMOVE program 227,228,229
alphameric tape labeling 209
alternate DD names, specifying 283
alternate tracks, assigning
  with IBCDASDI 27,30
  with IEHATLAS 185
  with IEHDASDR 189,198
ANALYZE statement 194
ANS volume labels 209,211
ASCII labels 209
assigning
  alternate tracks
    with IBCDASDI 27,30
    with IEHATLAS 185
    with IEHDASDR 189,198
  sequence numbers 171,172
  serial numbers (IEHDASDR) 195,197,198,199,201
asterisk (*) in PDS directory entry 217
ATTACH macro instruction 283
attributes of DD statements defining mountable volumes . . 287

B

backup copy, producing a
  using IEBCOPY 53
  using IEBGENER 107
  using IEHDASDR 190
bad VTOC, assigning alternate track for 185
basic move and copy operations 230
BDAM data set, moving or copying 230
BLDA statement 268
BLDG statement 269
BLDX statement 267
bold type, use of 7
bootstrap records, construction of 189
braces { }, use of 7
brackets [ ], use of 7
buffer
  FCB, loading of 43
  UCS, loading of 43
building
  a generation data set 291
  a generation index 260,290
  an index 258
  an index alias 259
bypassing defective-track checking feature 27,29,31

C

carriage control, specifying 133,135
catalog
  building index in 258,260
  copying 233
  listing 217
  moving 233
  placing entries in 257
cataloged data sets, punching 129
cataloging
  a data set 257
  a generation data set 297
  a procedure 175
  with the IEHMOVE program 227
  with the IEHPROGM program 257
cataloging moved or copied data, automatically 227
CATLG statement 266
CHANGE 168
changing
  a volume serial number 190
  the logical record length of a data set 109
  the organization of a data set 165,107
chart, utility program function 19
checking for flagged defective tracks
  with the IBCDASDI program 27
  with the IEHDASDR program 189
CLOSE module, changing or replacing 215
COLUMN specification 170
combinations of NEW, MEMBER, and NAME keywords . . . 171
comments, on utility control statements 24
COMPARE statement 47
comparing
  partitioned directories 45
  partitioned data sets 45
  records 45
  sequential data sets 45
compatible volumes 228
compatibility with respect to size, volume 228
compress in place 55
compressing a data set 55
concatenating SYSIN data sets when using IEHDASDR . . . 193
concurrent operations when using IEHDASDR, specifying . . 193
CONNECT statement 268
connecting two control volumes 259
considerations
  for the MVT user 24
  for defining DD statements 287
continuing utility controll statements 24
control passwords
  adding 269
  deleting 271
  replacing 270
control statements
  format of 23
  restrictions 24
control statements, subordinate 238

control volumes
connecting . . . 259
copying . . . 233
disconnecting . . . 259
moving . . . 233
controlling
IBCDASDI . . . 27
IBCDMPRS . . . 33
IBCRCVRP . . . 37
ICAPRTBL . . . 43
IEBCOMPR . . . 46
IEBCOPY . . . 56
IEBDG . . . 83
IEBEDIT . . . 101
IEBGENER . . . 109
IEBISAM . . . 125
IEBPTPCH . . . 130
IEBTCRIN . . . 149
IEBUPDAT . . . 159
IEBUPDTE . . . 166
IEHATLAS . . . 185
IEHDASDR . . . 192
IEHINITT . . . 210
IEHIOSUP . . . 215
IEHLIST . . . 220
IEHMOVE . . . 234
IEHPROGM . . . 263
IFHSTATR . . . 277
conventions, notational . . . 7
converting a data set
from partitioned to sequential organization . . . 107,165
from sequential to partitioned organization . . . 165
converting data
from alphameric to hexadecimal . . . 134,137
from H-set BCD to EBCDIC . . . 108,134,137
from packed to unpacked decimal . . . 108,114,134,137
from unpacked to packed decimal . . . 108
COPY statement . . . 58
COPY CATALOG statement . . . 244
COPY DSGROUP statement . . . 240
COPY DSNAME statement . . . 239
copy operation . . . 658,53
COPY PDS statement . . . 242
COPY VOLUME statement . . . 245
copying
a BDAM data set . . . 230
a catalog . . . 233
a data set . . . 53,230
a direct data set with variable spanned records . . . 234
a dumped data set . . . 191
a group of data sets . . . 232
a partitioned data set . . . 53,227,231
a volume of data sets . . . 233
an indexed sequential data set . . . 123,124
members of a partitioned data set . . . 53,227
records of a sequential data set . . . 107
sequential data sets . . . 53
CREATE statement . . . 89
creating
a backup copy
using IEBCOPY . . . 53
using IEBGENER . . . 107
using IEHDASDR . . . 190
a generation index . . . 290
a library . . . 165
a model DSCB . . . 292
a partitioned data set from sequential input . . . 107
a sequential copy of an indexed sequential data set . . . 123
a sequential output data set . . . 145
a sequential output job stream . . . 101
an edited data set . . . 107
user labels on sequential output . . . 107

D

DADEF statement . . . 28
DASDI program (see IBCDASDI)
data
dumped . . . 191
movable . . . 228
reconstructed . . . 227,123
recovering usable . . . 37
unloaded . . . 227,123,124
unmovable . . . 227
data set control block (DSCB), setting protection status in . . . 261
data set passwords . . . 261,262,263
adding . . . 262,269
deleting . . . 262,271
listing . . . 269,272
replacing . . . 270
data set utility programs . . . 25
IEBCOMPR . . . 45
IEBCOPY . . . 53
IEBDG . . . 81
IEBEDIT . . . 101
IEBGENER . . . 107
IEBISAM . . . 123
IEBPTPCH . . . 129
IEBTCRIN . . . 145
IEBUPDAT . . . 159
IEBUPDTE . . . 165
data sets
cataloging . . . 257
compressing . . . 55
converting . . . 107,165
copying . . . 53,230
merging . . . 55
moving . . . 230
protecting . . . 261
reconstructing . . . 55
re-creating . . . 257
renaming . . . 257
scratching . . . 257
uncataloging . . . 257
unloading . . . 227,123
data sets, moving or copying a group of . . . 232,233
data sets, partitioned (see partitioned data sets)
Data statement . . . 173
DD names, alternate . . . 283
DD statement, attributes . . . 287
DD statements, operational results of . . . 287
ddnameaddr . . . 283
deblocking with IEBCOPY . . . 57
defective track
indicated by IEHATLAS . . . 185
indicated by data check . . . 185
indicated by missing address marker . . . 185
recovering data from . . . 185,33
testing for . . . 189,27
deferred mounting, specifying . . . 287
deferred step restart with relative generation numbers . . . 290
defining data sets
with the IEBCOMPR program . . . 46
with the IEBCOPY program . . . 56
with the IEBDG program . . . 81,82,83
with the IEBEDIT program . . . 101
with the IEBGENER program . . . 110
with the IEBISAM program . . . 126
with the IEBPTPCH program . . . 130
with the IEBTCRIN program . . . 149
with the IEBUPDTE program . . . 166
with the IEHDASDR program . . . 191
with the IEHINITT program . . . 210
with the IEHIOSUP program . . . 215
with the IEHLIST program . . . 221
with the IEHMOVE program . . . 235
with the IEHPROGM program . . . 263

defining mountable devices 287
DFN 43
DELET statement 161
DELETE 171
DELETEP statement 271
deleting
  a record 171,161
  an index 258
  an index alias 259
  data set passwords 271
demounting mountable volumes 287
Detail statement 171
device name 24
DFN statement 43
direct access volumes
  assigning alternate tracks to 190,27,185
  dumping 373,198,199
  initializing 27,189
  restoring 33
direct data set, moving or copying 230
  with variable spanned records 234
directory entry, format of 212
disconnecting volumes 259
DLTA statement 268
DLTX statement 267
DSCB
  model for generation data sets 292
  setting protection status in 269,262
DSD statement 85
dummy header label 209
DUMP statement
  for IBCDMPRS program 33
  for IEHDASDR program 198
DUMP/RESTORE program (see IBCDMPRS)
dumping a direct access volume 333,189,198
dumping multiple volumes to a single restore tape 200
DUP (see TCRGEN statement)

E

EDIT statement 101
edited format
  of a VTOC 219
  of a PDS directory entry 218
editing
  a job stream 101
  data 145,148,101
  sequential data set 107
  partitioned data set 107
editing facilities
  with the IEBGENER program 107
  with the IEBPTPCH program 130
  with the IEBTCRIN program 151
ellipsis, use of 7
END statement
  for IBCDASDI 31
  for IBCDMPRS 36
  for IBCRCVRP 41
  for ICAPRTBL 44
  for IEBDG 93
end of cartridge 149
end of file (EOF) record, assigning alternate track 187
ENDUP statement
  for IEBUPDAT 162
  for IEBUPDTE 175
ensuring volume integrity 287
entering job control statements into a procedure library . . 176
EOV module, changing or replacing 215
ERROPT 152
ERROR 153
ESV record
  format 277
  processing 277
examples
  IBCDASDI 31
  IBCDMPRS 36
  IBCRCVRP 41
  ICAPRTBL 44
  IEBCOMPR 48
  IEBCOPY 62
  IEBDG 93
  IEBEDIT 103
  IEBGENER 115
  IEBISAM 127
  IEBPTPCH 139
  IEBTCRIN 157
  IEBUPDAT 162
  IEBUPDTE 175
  IEHATLAS 187
  IEHDASDR 202
  IEHINITT 212
  IEHIOSUP 215
  IEHLIST 223
  IEHMOVE 248
  IEHPROGM 272
  IFHSTATR 278
EXCLUDE statement
  for IEBCOPY 61
  for IEHMOVE 246
excluding data from copy operations 54,55
exclusive copy operation 55,61
executing
  a data set utility program 24
  a system utility program 24
  an independent utility program 26
exit routine linkage 279
exit routines
  location of 279
  parameter lists for 279
  return codes issued by user 281
  returning from 280
EXITS statement
  for IEBCOMPR 47
  for IEBGENER 111
  for IEBPTPCH 136
  for IEBTCRIN 153
expanding a partitioned data set (see *also* merging partitioned data sets) 107
EXPDT subparameter 236
expiration date, specifying 236

F

FCB
  loading of 43,44
  statement 44
FD statement 85
FEOV module, changing or replacing 215
FIELD parameter 114,137
field processing and editing information, specifying . . 114,137
flagged defective tracks, checking for 189,27
FORMAT statement 196
format of utility control statements 23,24
forms control buffer, loading the 43,44
Function statement 167

G

general uses
  for data set utility programs 25
  for independent utility programs 25
  for system utility programs 24
GENERATE statement 111
generating test data 81

generation
allocating a . . . 292
cataloging a . . . 291
creating a new . . . 291
DCB attributes for . . . 289,292
definition of . . . 289
retrieving a . . . 293
supplying DCB attributes for . . . 289,292
uncataloging . . . 292
using IEHPROGM to catalog . . . 292
using JCL to catalog . . . 292
generation data groups . . . 289
building . . . 289
cataloging . . . 292
creating an ISAM data set as part of . . . 292
deferred step restart with . . . 290
examples . . . 293
general discussion of . . . 289
multiprogramming considerations with . . . 293
generation index, building . . . 290
generation numbers
absolute generation and version number . . . 289
deferred step restart with . . . 290
relative . . . 290
GETALT statement
for IBCDASDI program . . . 28
for IEHDASDR program . . . 198

H

header record, initializing . . . 209
Header statement . . . 160
H-set BCD to EBCDIC conversion . . . 108,114

I

IBCDASDI program . . . 27,25,26
control of . . . 27
examples . . . 31
executing . . . 26
input and output . . . 27
introduction . . . 25
used to
assign an alternate track . . . 27
initialize a direct access volume . . . 27
utility control statements . . . 27
DADEF . . . 28
END . . . 28
GETALT . . . 28
IPLTXT . . . 28
JOB . . . 27
LASTCARD . . . 28
MSG . . . 28
VLD . . . 28
VTOCD . . . 28
IBCDMPRS program . . . 33
control of . . . 33
examples . . . 36
executing . . . 26
input and output . . . 33
introduction . . . 25
used to
dump data . . . 33
restore data . . . 33
utility control statements . . . 33
DUMP . . . 33
END . . . 33
JOB . . . 33
MSG . . . 33
RESTORE . . . 35
VDRL . . . 35
IBCRCVRP program . . . 37
control of . . . 37
examples . . . 41
executing . . . 26
input and output . . . 37
introduction . . . 25
used to
recover usable data . . . 37
replace bad data . . . 37
utility control statements . . . 37
END . . . 41
JOB . . . 38
INSERT . . . 40
LIST . . . 40
MSG . . . 38
RECOVER . . . 38
REPLACE . . . 39
ICAPRTBL program . . . 43
control of . . . 43
example . . . 44
executing . . . 26
input and output . . . 43
introduction . . . 25
used to
load forms control buffer . . . 43
load Universal Character Set buffer . . . 43
utility control statements . . . 43
DFN . . . 43
END . . . 44
FCB . . . 44
JOB . . . 43
UCS . . . 43
IEBCOMPR program . . . 45
control of . . . 46
job control statements . . . 46
restrictions . . . 46
utility control statements . . . 47
examples . . . 48
input and output . . . 46
introduction . . . 25
region size . . . 46
return codes . . . 46
used to
compare partitioned data sets . . . 45
compare sequential data sets . . . 45
verify backup copies . . . 45
verify portions of records . . . 45
utility control statements . . . 47
COMPARE . . . 47
EXITS . . . 47
LABELS . . . 47
IEBCOPY program . . . 53
control of . . . 56
job control statements . . . 56
restrictions . . . 57
space allocation for spill data sets . . . 57
utility control statements . . . 58
examples . . . 62
input and output . . . 55
introduction . . . 25
region size . . . 56
return codes . . . 56
used to
compress a data set in place . . . 55
copy data sets . . . 53
create a backup copy . . . 53
exclude members from a copy operation . . . 55
merge data sets . . . 55
re-create a data set when allocated space is exhausted . . . 55
rename selected members . . . 55
replace identically named members . . . 54
replace selected members . . . 55
select members to be copied . . . 54

utility control statements . . . 58
COPY . . . 58
EXCLUDE . . . 61
INDD . . . 59
SELECT . . . 60
IEBDG program . . . 81
control of . . . 83
job control statements . . . 83
PARM information . . . 84
restrictions . . . 84
utility control statements . . . 85
examples . . . 93
input and output . . . 82
introduction . . . 25
region size . . . 84
return codes . . . 83
used to
generate test data . . . 81
modify selected fields . . . 82
utility control statements . . . 85
CREATE . . . 89
DSD . . . 85
END . . . 92
FD . . . 85
REPEAT . . . 92
IEBEDIT program . . . 101
control of . . . 101
job control statements . . . 101
restrictions . . . 101
utility control statement . . . 101
examples . . . 103
input and output . . . 101
introduction . . . 25
region size . . . 101
return codes . . . 101
used to
copy an entire job . . . 101
copy selected job steps . . . 101
utility control statement . . . 101
EDIT . . . 101
IEBGENER program . . . 107
control of . . . 109
job control statements . . . 110
restrictions . . . 110
utility control statements . . . 111
examples . . . 115
input and output . . . 109
introduction . . . 25
region size . . . 110
return codes . . . 109
used to
change logical record length . . . 109
create a backup copy . . . 107
create user labels on sequential output . . . 107
expand a partitioned data set . . . 107
produce a partitioned data set from sequential input . 107
produce an edited data set . . . 107
reblock . . . 109
utility control statements . . . 111
EXITS . . . 111
GENERATE . . . 111
LABELS . . . 112
MEMBER . . . 113
RECORD . . . 113
IEBISAM program . . . 123
control of . . . 125
job control statements . . . 125
PARM information . . . 125
examples . . . 127
input and output . . . 125
introduction . . . 25
region size . . . 126
return codes . . . 125
used to
copy an indexed sequential data set . . . 123
create a sequential copy of an indexed sequential data set . . . 123
create an indexed sequential data set from an unloaded data set . . . 124
print an indexed sequential data set . . . 124
IEBPTPCH program . . . 129
control of . . . 130
job control statements . . . 130
restrictions . . . 131
utility control statements . . . 131
examples . . . 139
input and output . . . 130
introduction . . . 25
region size . . . 130
return codes . . . 130
used to print or punch
a partitioned directory . . . 129
an edited data set . . . 130
data sets . . . 129
selected members . . . 129
selected records . . . 129
utility control statements . . . 131
EXITS . . . 136
LABELS . . . 138
MEMBER . . . 136
PRINT . . . 132
PUNCH . . . 134
RECORD . . . 136
TITLE . . . 135
IEBTCRIN program . . . 145
control of . . . 149
job control statements . . . 149
restrictions . . . 150
utility control statements . . . 150
examples . . . 157
input and output . . . 149
introduction . . . 25
region size . . . 149
used to
edit data . . . 145,148
produce sequential output data . . . 145
read input . . . 145
utility control statements . . . 150
EXITS . . . 153
TCRGEN . . . 151
IEBUPDAT program . . . 159
control of . . . 159
job control statements . . . 159
PARM information . . . 159
utility control statements . . . 160
examples . . . 162
input and output . . . 159
introduction . . . 25
region size . . . 159
used to incorporate source language modifications . . . 159
utility control statements . . . 160
ALIAS . . . 162
DELET . . . 161
ENDUP . . . 162
Header . . . 160
Logical Record . . . 162
NUMBR . . . 161
IEBUPDTE program . . . 165
control of . . . 166
job control statements . . . 166
PARM information . . . 167
restrictions . . . 166
utility control statements . . . 167
examples . . . 175
input and output . . . 165
introduction . . . 25
region size . . . 166

return codes . . . 165
used to incorporate source language modifications . . . 165
utility control statements . . . 167
ALIAS . . . 174
Data . . . 173
Detail . . . 171
ENDUP . . . 175
Function . . . 167
LABEL . . . 173
IEHATLAS program . . . 185
control of . . . 185
job control statements . . . 185
restrictions . . . 185
utility control statement . . . 186
examples . . . 187
input and output . . . 185
introduction . . . 24
region size . . . 185
used to indicate a defective track . . . 185
utility control statement
TRACK or VTOC . . . 186
IEHDASDR program . . . 189
control of . . . 192
job control statements . . . 192
PARM information . . . 193
restrictions . . . 193
utility control statements . . . 194
examples . . . 202
input and output . . . 191
introduction . . . 24
region size . . . 191
return codes . . . 191
used to
assign alternate tracks . . . 190
change volume serial numbers . . . 190
copy dumped data . . . 191
create a copy . . . 191
initialize with recording-surface analysis . . . 189
initialize without recording-surface analysis . . . 189
utility control statements . . . 194
ANALYZE . . . 194
DUMP . . . 198
FORMAT . . . 196
GETALT . . . 198
IPLTXT . . . 201
LABEL . . . 198
RESTORE . . . 200
IEHINITT program . . . 209
control of . . . 210
job control statements . . . 210
PARM information . . . 210
restrictions . . . 210
utility control statement . . . 211
examples . . . 212
input and output . . . 210
introduction . . . 24
region size . . . 210
return codes . . . 210
used to place volume label sets on magnetic tape . . . 209
utility control statement
INITT . . . 211
IEHIOSUP program . . . 215
control of . . . 215
job control statements . . . 215
restrictions . . . 215
examples . . . 215
input and output . . . 215
introduction . . . 24
region size . . . 215
return codes . . . 215
used to update TTR entries . . . 215
IEHLIST program . . . 217
control of . . . 220
job control statements . . . 220
PARM information . . . 221
restrictions . . . 221
utility control statements . . . 221
examples . . . 223
input and output . . . 220
introduction . . . 24
region size . . . 220
return codes . . . 220
used to list entries in
catalog . . . 217
directory . . . 217
volume table of contents . . . 218
utility control statements . . . 221
LISTCTLG . . . 221
LISTPDS . . . 222
LISTVTOC . . . 222
IEHMOVE program . . . 227
control of . . . 234
job control statements . . . 235
for track overflow feature . . . 237
PARM information . . . 236
restrictions . . . 236
utility control statements . . . 237
examples . . . 248
input and output . . . 234
introduction . . . 24
region size . . . 235
return codes . . . 234
used to move or copy
a catalog . . . 233
a data set . . . 230
a group of cataloged data sets . . . 232
a volume of data sets . . . 233
direct data sets with variable spanned records . . . 234
utility control statements . . . 237
COPY CATALOG . . . 244
COPY DSGROUP . . . 240
COPY DSNAME . . . 239
COPY PDS . . . 242
COPY VOLUME . . . 245
EXCLUDE . . . 246
INCLUDE . . . 246
MOVE CATALOG . . . 243
MOVE DSGROUP . . . 239
MOVE DSNAME . . . 238
MOVE PDS . . . 241
MOVE VOLUME . . . 245
REPLACE . . . 247
SELECT . . . 247
IEHPROGM program . . . 257
control of . . . 263
job control statements . . . 264
PARM information . . . 265
restrictions . . . 264
utility control statements . . . 265
examples . . . 272
input and output . . . 263
introduction . . . 24
region size . . . 264
return codes . . . 263
used to
add data set passwords . . . 262
build and maintain a generation index . . . 260
build or delete an index . . . 258
build or delete an index alias . . . 259
catalog or uncatalog a data set . . . 257
connect and release volumes . . . 259
delete data set passwords . . . 262
maintain data set passwords . . . 261

rename a data set or member . . . 257
replace data set passwords . . . 262
scratch a data set or member . . . 257
utility control statements . . . 265
ADD . . . 269
BLDA . . . 268
BLDG . . . 269
BLDX . . . 267
CATLG . . . 266
CONNECT . . . 268
DELETEP . . . 271
DLTA . . . 268
DLTX . . . 267
LIST . . . 272
RELEASE . . . 269
RENAME . . . 266
REPLACE . . . 270
SCRATCH . . . 265
UNCATLG . . . 267
IFHSTATR program . . . 277
control of . . . 277
job control statements . . . 278
example . . . 278
input and output . . . 277
introduction . . . 24
region size . . . 278
use of . . . 277
INCLUDE statement . . . 246
including data in move and copy operations . . . 227
INDD statement . . . 59
independent utility programs . . . 25
IBCDASDI . . . 25
IBCDMPRS . . . 33
IBCRCVRP . . . 37
ICAPRTBL . . . 43
index
building . . . 258,267
create new . . . 267
deleting . . . 258,267
generate entries in . . . 266
generation . . . 289,290,269
remove entries from . . . 267,269
index alias
building . . . 259,268
deleting . . . 259,268
index structure . . . 258,259,260
building . . . 258,259,260
deleting from . . . 258,259,268
indexed sequential data sets
copying . . . 123
creating backup . . . 123
loading . . . 123,124
printing . . . 124
unloading . . . 123
initializing direct access volumes
with IBCDASDI . . . 27
with IEHDASDR . . . 189
with surface analysis . . . 189
without surface analysis . . . 189
INITT statement . . . 211
input stream, organizing . . . 101
input to and output from
IBCDASDI . . . 27
IBCDMPRS . . . 33
IBCRCVRP . . . 37
ICAPRTBL . . . 43
IEBCOMPR . . . 46
IEBCOPY . . . 55
IEBDG . . . 82
IEBEDIT . . . 101
IEBGENER . . . 109
IEBISAM . . . 125
IEBPTPCH . . . 130
IEBTCRIN . . . 149
IEBUPDAT . . . 159
IEBUPDTE . . . 165
IEHATLAS . . . 185
IEHDASDR . . . 191
IEHINITT . . . 210
IEHIOSUP . . . 215
IEHLIST . . . 220
IEHMOVE . . . 234
IEHPROGM . . . 263
IFHSTATR . . . 277
INSERT statement . . . 40
inserting blocks of records . . . 173,165
introduction
to data set utilities . . . 25
to independent utilities . . . 25
to system utilities . . . 24
invoking utility programs . . . 283
IPLTXT statement
for IBCDASDI . . . 30
for IEHDASDR . . . 201
IPL program . . . 195,197,201,28,30
IPL records, writing . . . 195,197,201,28,30
italic type, use of . . . 7

J

job control statement requirements . . . 23
job control statements for
IEBCOMPR . . . 46
IEBCOPY . . . 56
IEBDG . . . 83
IEBEDIT . . . 101
IEBGENER . . . 110
IEBISAM . . . 125
IEBPTPCH . . . 130
IEBTCRIN . . . 149
IEBUPDAT . . . 159
IEBUPDTE . . . 166
IEHATLAS . . . 185
IEHDASDR . . . 192
IEHINITT . . . 210
IEHIOSUP . . . 215
IEHLIST . . . 220
IEHMOVE . . . 235
IEHPROGM . . . 264
IFHSTATR . . . 278
JOB statement
for IBCDASDI . . . 27
for IBCDMPRS . . . 33
for IBCRCVRP . . . 38
for ICAPRTBL . . . 43
job statements in an output data set . . . 101
JOB steps, copying . . . 111
job stream, organizing . . . 101

K

keywords, combinations of NEW, MEMBER, and NAME . . . 171

L

LABEL statement
for IEBUPDTE . . . 173
for IEHDASDR . . . 198
label processing
using IEBCOMPR . . . 47
using IEBGENER . . . 112
using IEBPTPCH . . . 138
using IEBTCRIN . . . 154,155
using IEBUPDTE . . . 169
using IEHMOVE . . . 230
LABELS statement
for IEBCOMPR . . . 47
for IEBGENER . . . 112
for IEBPTPCH . . . 136

labeling a magnetic tape volume . . . 209
LASTCARD statement . . . 31
levels of index
creating . . . 258
deleting . . . 258
libraries, updating symbolic . . . 165
LINK macro instruction . . . 273,279
linking to an exit routine . . . 273
LIST statement
for IBCRCVRP . . . 40
for IEHPROGM . . . 272
listing
a catalog . . . 217
a partitioned data set . . . 129
a partitioned directory . . . 129,217
a password entry . . . 272
a volume table of contents . . . 218
catalog entries . . . 217
data set passwords . . . 272
error statistics by volume (ESV) records . . . 277
system control data . . . 217
LISTCTLG statement . . . 221
LISTPDS statement . . . 222
LISTVTOC statement . . . 222
literal information, supplying . . . 7
load operation, specified in PARM parameter . . . 126
loading
an indexed sequential data set . . . 123,124
an unloaded data set . . . 123
forms control buffer . . . 43
Universal Character Set buffer . . . 43
logical record length, changing . . . 109
Logical Record statement . . . 162

M

magnetic tape volumes
labeling . . . 209
moving or copying a group of data sets to . . . 232
moving or copying a volume of data to . . . 233
MEMBER, NEW, and NAME keywords, combinations of . . . 171
MEMBER statement
for IEBGENER . . . 113
for IEBPTPCH . . . 136
members, partitioned, data set
comparing . . . 45
copying and merging . . . 53,227
printing and punching . . . 129
renaming . . . 257,227
replacing . . . 227
scratching . . . 257
members of a symbolic library
adding . . . 167
replacing . . . 227
merge data sets . . . 55
methods of executing
data set utility programs . . . 25
independent utility programs . . . 26
system utility programs . . . 24
imum region sizes (see region sizes for utility programs)
model DSCB, creating . . . 292
modify selected fields . . . 82
mountable devices, defining . . . 287
MOVE CATALOG statement . . . 243
MOVE DSGROUP statement . . . 239
MOVE DSNAME statement . . . 238
MOVE PDS statement . . . 241
MOVE VOLUME statement . . . 245
moving
a BDAM data set . . . 230
a catalog . . . 233
a data set . . . 230
a direct data set with variable spanned records . . . 234
a group of data sets . . . 232
a volume of data sets . . . 233
the SYSCTLG data set . . . 233
moving and copying
data . . . 227
user labels . . . 230
moving and copying operations
excluding data from . . . 55,61,246
including data in . . . 246
results of . . . 227,228
selecting members for . . . 54,60,247
moving or copying a password-protected volume . . . 229
moving the SVC library . . . 230
move vs. copy . . . 227
MSG statement
for IBCDASDI . . . 28
for IBCDMPRS . . . 33
for IBCRCVRP . . . 38
MTDI input . . . 145
MTST input . . . 145
multivolume data sets, moving or copying . . . 227
multiprogramming considerations
with generation data groups . . . 292
for MFT system . . . 287
for MVT system . . . 287,24

N

new master data set . . . 165
NEW, MEMBER, and NAME keywords, combinations of . . . 171
nonsharable attribute, assigning . . . 287
nonsharable devices . . . 287
notation for defining control statements . . . 7
NUMBR statement . . . 161
numbering records in a partitioned data set . . . 165,168
numeric tape labeling . . . 209

O

old master data set . . . 165
OPEN module, changing or replacing . . . 215
operand field . . . 23,24
operating procedures for independent utilities . . . 26
optionaddr . . . 283,285
order of moved or copied members with the IEHMOVE program . . . 231,232
organizing an input stream . . . 101
output from utility programs (see input to and output from)

P

packed to unpacked decimal conversion . . . 108,114,134,137
PARAM subparameter . . . 283
parameters passed to exit routines . . . 279
for label processing . . . 279
for non-label processing . . . 280
PARM information
with the IEBDG program . . . 84
with the IEBISAM program . . . 125
with the IEBUPDAT program . . . 159
with the IEBUPDTE program . . . 167
with the IEHDASDR program . . . 193
with the IEHINITT program . . . 210
with the IEHLIST program . . . 221
with the IEHMOVE program . . . 236
with the IEHPROGM program . . . 265
partial dumps of direct access volumes . . . 198
partitioned data sets
comparing . . . 45
compressing in place . . . 55
converting to sequential . . . 107,165
copying . . . 53,230
copying selected members of . . . 53,227
editing . . . 107
expanding . . . 107
excluding from move and copy operations . . . 61,246
listing . . . 217

merging members of . . . 61
moving . . . 230
numbering records in . . . 165,168
produced from sequential input . . . 165
renaming . . . 257
replacing records in . . . 54,165
unloading . . . 227
updating in place . . . 165
partitioned data set directory entry, edited format . . 217,218
PASSWORD data set
adding entries to . . . 269
deleting entries from . . . 271
listing entries in . . . 272
replacing entries in . . . 270
password-protected data sets, IEHDASDR . . . 189
password-protected volumes, moving or copying . . . 229
passwords, data set
adding . . . 269
deleting . . . 271
listing . . . 272
replacing . . . 270
patterns of test data . . . 81
picture, user-specified . . . 82
prerequisite publications . . . 6
print specifications
standard . . . 129
user . . . 129
PRINT statement . . . 131
printing
a partitioned directory . . . 129,217
an edited data set . . . 130
data sets . . . 129
indexed sequential data sets . . . 124
logical records from an indexed sequential data set . . . 124
partitioned data sets . . . 129
selected records . . . 129
sequential data set . . . 107,129
private attribute, assigning . . . 287
procedure library, entering procedures in . . . 176
procedures
cataloging . . . 176
program classes
data set . . . 25
system . . . 24
independent . . . 25
program selection . . . 19
protecting data sets (see IEHPROGM utility program)
public attribute, specifying . . . 287
punch specifications
standard . . . 129
user . . . 129
PUNCH statement . . . 134
punching
records . . . 129,136
partitioned data sets . . . 129
sequential data sets . . . 129
punctuation, in syntax . . . 7
purging unexpired data sets
ANALYZE operation . . . 194
DUMP operation . . . 198
FORMAT operation . . . 196
RESTORE operation . . . 200

Q

quick DASDI . . . 27,189,196

R

read MTDI input . . . 145
read MTST input . . . 145
reader procedure, selecting . . . 25
rearranging data fields within a record . . . 107
reblocking
with IEBCOPY . . . 57
with IEBGENER . . . 109
with IEHMOVE . . . 230
RECORD statement
for IEBGENER . . . 113
for IEBPTPCH . . . 136
record groups, assigning . . . 107
records
adding . . . 160,165,173,159
assigning sequence numbers to . . . 159,161,165,170
comparing . . . 45
copying . . . 107
deleting . . . 171
error . . . 145
error statistics by volume . . . 277
ESV . . . 277
printing . . . 129
punching . . . 129
renumbering . . . 165
replacing . . . 167,157,165,173
RECOVER statement . . . 38
recovering data from defective tracks . . . 27
recovering usable data . . . 37
re-creating a data set . . . 55
region specifications
IEBCOMPR . . . 46
IEBCOPY . . . 56
IEBDG . . . 84
IEBEDIT . . . 101
IEBGENER . . . 110
IEBISAM . . . 126
IEBPTPCH . . . 130
IEBTCRIN . . . 149
IEBUPDAT . . . 159
IEBUPDTE . . . 166
IEHATLAS . . . 185
IEHDASDR . . . 192
IEHINITT . . . 210
IEHIOSUP . . . 215
IEHLIST . . . 220
IEHMOVE . . . 235
IEHPROGM . . . 264
IFHSTATR . . . 278
registers, contents of when linking . . . 279
relative generation numbers . . . 290
RELEASE statement . . . 269
releasing two volumes . . . 259
removable volumes, allocating . . . 287
removing
entries from an index structure . . . 257
member and alias names from a partitioned directory . 257
RENAME statement . . . 266
renaming
a data set . . . 257
a multivolume data set . . . 273
selected members . . . 55,257,266
renumbering . . . 165
REPEAT statement . . . 92
REPL . . . 168
REPLACE statement
for IBCRCVRP . . . 39
for IEHMOVE . . . 247
for IEHPROGM . . . 270
replacement data records . . . 159,165,173
replacing
bad data . . . 37
data set passwords . . . 270
identically named members . . . 54
members in move and copy operations . . . 227,53,54
members of a symbolic library . . . 165
records in a partitioned data set . . . 54,165
selected members . . . 55

REPRO 168
reproducing members of a symbolic library 159,165
required publications 6
requirements, job control statement 23
RESTORE statement
  for IBCDMPRS 35
  for IEHDASDR 200
restoring data to a direct access volume 33,200
RETURN macro instruction 280
return codes
  for IEBCOMPR 46
  for IEBCOPY 56
  for IEBDG 83
  for IEBEDIT 101
  for IEBGENER 109
  for IEBISAM 125
  for IEBPTPCH 130
  for IEBUPDTE 165
  for IEHDASDR 191
  for IEHINITT 210
  for IEHIOSUP 215
  for IEHLIST 220
  for IEHMOVE 234
  for IEHPROGM 263
return codes, action on 280
returning from an exit routine 280

S

SCRATCH module, changing or replacing 215
SCRATCH statement 265
scratching
  a data set 257
  a member 257
  a volume table of contents 257,265
  temporary data sets 257,265
secondary passwords 262
  adding 269
  deleting 271
  replacing 270
SELECT statement
  for IEBCOPY 60
  for IEHMOVE 247
selecting a program 19
selecting members to be moved or copied 54
selective
  copy 60
  rename 55
  replace 60
sequence numbers, assigning 165
sequential data sets
  comparing 45
  compressing 55
  converting to partitioned 159,165
  copying 53
  creating 123,145,101
  editing 145,148,101
  printing 107,129,130
  punching 129,130
  unloaded 123
  unloading 227
sequential output job stream, creating 101
sharing mountable devices 287
simultaneous IEHDASDR operations 193
SOR 145
space allocation with IEBCOPY 57
specific request for mountable volumes 287
specific volumes, making requests for 287
specifying an expiration date 236
spill data sets, used with IEBCOPY 57
standard print operation 129
standard punch operation 129
straight copy 54
summary of major changes 17
supplying literal information 7
surface analysis of direct access volumes 189,194,27
SVC library, moving 215
symbolic libraries, updating 165
SYSCTLG data set
  creating 266
  moving or copying 233
system control data, listing 217
system status information 170
system utility programs 24
  IEHATLAS 185
  IEHDASDR 189
  IEHINITT 209
  IEHIOSUP 215
  IEHLIST 217
  IEHMOVE 227
  IEHPROGM 257
  IFHSTATR 277

T

tape volumes, labeling 209
tapemark in a volume label set 209
TCLOSE module, changing or replacing 215
TCRGEN statement 151
temporary data sets, scratching 265
temporary spill data sets 57
test data
  generating 81
  patterns of 81
TITLE statement 135
TRACK statement 186
track overflow feature
  with IEHATLAS 186
  with IEHMOVE 237
tracks, getting alternate 27,185,189
transfer control tables, updating 215
TTR entries, updating 215
TYPE 151
type 21 record processing 277

U

UCS
  loading of 43
  statement 43
uncataloging a data set 257
UNCATLG statement 267
underscore, use of 7
unexpired data sets encountered
  during ANALYZE operation 195
  during DUMP operation 200
  during FORMAT operation 197
  during RESTORE operation 201
Universal Character Set buffer, loading the 43
unloaded data 123
unloaded data sets
  creating 123
  loading 123,124
  reconstructing 123
  format of (IEBISAM) 124
unloading
  indexed sequential data set 123
  partitioned data set 227
  sequential data set 227
unmovable data sets, moving or copying 227
unpacked to packed decimal conversion 114,115
updating
  symbolic libraries 165
  transfer control tables 215
  TTR entries in the SVC library 215
updating in place, a partitioned data set 165
user exits (see exits)

user labels
as data . . . 298
as data set descriptors . . . 297
copying . . . 230
EXITS statement . . . 47,111,136,153
LABEL statement
LABELS statement . . . 47,112,138
linkage with label processing exit routines . . . 297
moving . . . 230
parameter lists of exit routines . . . 279
RECORD statement . . . 113
relationship between EXITS and LABELS . . . 298
return codes from exit routines . . . 298
utility program handling of . . . 297
volume switch labels . . . 297
with IEBCOMPR . . . 47
with IEBGENER . . . 112
with IEBPTPCH . . . 138
with IEBUPDTE . . . 173
with IEHMOVE . . . 230
user print specifications . . . 129
user punch specifications . . . 129
user-supplied picture . . . 82
using NEW, MEMBER, and NAME keywords . . . 171
utility control statements, format of . . . 23
utility control statements (IBCDASDI) . . . 27
DADEF . . . 27
END . . . 28
GETALT . . . 28
IPLTXT . . . 28
JOB . . . 27
LASTCARD . . . 28
MSG . . . 28
VLD . . . 28
VTOCD . . . 28
utility control statements (IBCDMPRS) . . . 33
DUMP . . . 33
END . . . 36
JOB . . . 33
MSG . . . 33
RESTORE . . . 35
VDRL . . . 35
utility control statements (IBCRCVRP) . . . 37
END . . . 41
INSERT . . . 40
JOB . . . 38
LIST . . . 40
MSG . . . 38
RECOVER . . . 38
REPLACE . . . 39
utility control statements (ICAPRTBL) . . . 43
DFN . . . 43
END . . . 44
FCB . . . 44
JOB . . . 43
UCS . . . 43
utility control statements (IEBCOMPR) . . . 46
COMPARE . . . 46
EXITS . . . 46
LABELS . . . 46
utility control statements (IEBCOPY) . . . 58
COPY . . . 58
EXCLUDE . . . 61
INDD . . . 59
SELECT . . . 60
utility control statements (IEBDG) . . . 85
CREATE . . . 89
DSD . . . 85
END . . . 92
FD . . . 85
REPEAT . . . 92
utility control statements (IEBEDIT) . . . 101
EDIT . . . 101
utility control statements (IEBGENER) . . . 111
EXITS . . . 111
GENERATE . . . 111
LABELS . . . 112
MEMBER . . . 113
RECORD . . . 113
utility control statements (IEBPTPCH) . . . 131
EXITS . . . 136
LABELS . . . 138
MEMBER . . . 136
PRINT . . . 132
PUNCH . . . 134
RECORD . . . 136
TITLE . . . 135
utility control statements (IEBTCRIN) . . . 150
EXITS . . . 153
TCRGEN . . . 151
utility control statements (IEBUPDAT) . . . 160
ALIAS . . . 162
DELET . . . 161
ENDUP . . . 162
Header . . . 160
Logical Record . . . 162
NUMBR . . . 161
utility control statements (IEBUPDTE) . . . 167
ALIAS . . . 174
Data . . . 173
Detail . . . 171
ENDUP . . . 175
Function . . . 167
LABEL . . . 173
utility control statement (IEHATLAS) . . . 186
TRACK or VTOC . . . 186
utility control statements (IEHDASDR) . . . 194
ANALYZE . . . 194
DUMP . . . 198
FORMAT . . . 196
GETALT . . . 198
IPLTXT . . . 201
LABEL . . . 198
RESTORE . . . 200
utility control statement (IEHINITT) . . . 211
INITT . . . 211
utility control statements (IEHLIST) . . . 221
LISTCTLG . . . 221
LISTPDS . . . 222
LISTVTOC . . . 222
utility control statements (IEHMOVE) . . . 237
COPY CATALOG . . . 244
COPY DSGROUP . . . 240
COPY DSNAME . . . 239
COPY PDS . . . 242
COPY VOLUME . . . 245
EXCLUDE . . . 246
INCLUDE . . . 246
MOVE CATALOG . . . 243
MOVE DSGROUP . . . 239
MOVE DSNAME . . . 238
MOVE PDS . . . 241
MOVE VOLUME . . . 245
REPLACE . . . 247
SELECT . . . 247
utility control statements (IEHPROGM) . . . 265
ADD . . . 269
BLDA . . . 268
BLDG . . . 269
BLDX . . . 267
CATLG . . . 266
CONNECT . . . 268
DELETEP . . . 271
DLTA . . . 268
DLTX . . . 267
LIST . . . 272
RELEASE . . . 269

RENAME 266
REPLACE 270
SCRATCH 265
UNCATLG 267
utility programs
functions of 19
invocation of from a problem program 283

V

VDRL statement 35
verify
backup copies 45
portions of records 45
VLD statement 29
volume compatibility with respect to size 228
volume integrity, ensuring 287
volume label set
contents of 209
placing on magnetic tape 209
volume serial number, changing 190
volume switch labels, processing 297
volume table of contents
listing 218
scratching 265
volumes
copying 232
mounting and demounting 287
moving 232
VTOC, see volume table of contents
VTOC entries/track, by device type
VTOC statement 186
VTOCD statement 30

W

3211
loading forms control buffer for 43,44
loading Universal Character Set buffer for 43

Your Comments, Please ...

IBM System/360 Operating System
OS Utilities
System Reference Library

Order Number GC28-6586-15

Your comments and constructive criticism regarding this publication will help us to improve it so that you may be better served. Each comment will be reviewed carefully by those responsible for the publication. Comments and suggestions become the property of IBM.

Requests for copies of publications or for assistance in using IBM systems and programs should not be offered as comments but should be directed to your local IBM representative.

Thank you for your cooperation.

Comments:

No postage necessary if mailed in the U.S.A.

First Class
Permit No. 568
Boulder, Colorado

Business Reply Mail
No Postage Necessary If Mailed In The U. S. A.

Postage will be paid by:

IBM Corporation
Post Office Box 1900
Boulder, Colorado 80302

Attention:
Programming Publications

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U. S. A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)