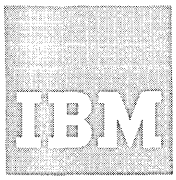
File No. S360-24
GC28-6380-3

OS



Systems Reference Library

IBM System/360 Operating System

COBOL (F) Programmer's Guide

Program Number 360S-CB-524

360S-LM-525

This publication describes how to compile, linkage edit, and execute a COBOL (F) program. The text also describes the output from each of these steps. In addition, it explains options of the compiler and many available features of the operating system.

Fourth Edition

This is a major revision of, and makes obsolete, Form C28-6380-2. It contains a new chapter on the Operating System Checkpoint/Restart facility, a revised discussion of the MFT and MVT environments, and expanded discussions of the source program library facility and error processing procedures. Other changes to the text, and small changes to illustrations, are indicated by a vertical line to the left of the change; changed or added illustrations are denoted by the symbol • to the left of the caption.

This edition corresponds to Release 17 of the IBM System/360 Operating System.

Changes are periodically made to the specifications herein; any such changes will be reported in subsequent revisions or Technical Newsletters.

Requests for copies of IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

Address comments concerning the contents of this publication to IBM Corporation, Programming Publications, 1271 Avenue of the Americas, New York, N. Y. 10020.

The purpose of this publication is to enable programmers to compile, linkage edit, and execute COBOL (F) programs under control of IBM System/360 Operating System. The COBOL (F) language is described in IBM System/360 Operating System: COBOL Language, Form C28-6516, which is a corequisite to this publication.

Programmers who are unfamiliar with the concepts of System/360 Operating System should read the "Introduction," "Using the DD Statement," "A Checklist for Job Control Procedures," and "Cataloged Procedures." These chapters provide information about the preparation of COBOL programs for processing by the operating system.

Programmers who are familiar with the operating system and wish to know how to run COBOL programs should read 'Options for the Compiler and Linkage Editor' under "Format of the Job Control Statements," "Compiler Data Set Requirements," "Linkage Editor Data Set Requirements," "Execution Time Data Set Requirements," and "Output" in addition to the four chapters listed above.

Chapters "Program Checkout" and "Programming Techniques" are of special interest, since they contain information about debugging and efficient programming. Other chapters discuss optional features of the language and the operating system. Some chapters include introductory information about features of the operating system which are described in detail in other publications.

The machine configuration required for system operations is described in the chapter "Machine Considerations."

Wider and more detailed discussions of the operating system are given in the following publications:

IBM System/360 Operating System: Job Control Language, Form C28-6539

IBM System/360 Operating System: Job Control Language Charts, Form C28-6632

IBM System/360 Operating System: System's Programmer's Guide, Form C28-6550

IBM System/360 Operating System: Linkage Editor, Form C28-6538

IBM System/360 Operating System: Supervisor and Data Management Services, Form C28-6646

IBM System/360 Operating System: Supervisor and Data Management Macro-Instructions, Form C28-6647

IBM System/360 Operating System: Sort/Merge, Form C28-6543

IBM System/360 Operating System: Utilities, Form C28-6586

IBM System/360 Operating System: System Generation, Form C28-6554

IBM System/360 Operating System: Messages, and Codes, Form C28-6631

IBM System/360 Operating System: Programmer's Guide to Debugging, Form C28-6670

IBM System/360 Operating System: Concepts and Facilities, Form C28-6535

INTRODUCTION 9
Data Set Organization 9
Executing A COBOL Program 10
Compilation 10
Linkage Editing 10
Execution 10
System/360 Operating System Environments 10
Primary Control Program (PCP) 10
Multiprogramming with a Fixed Number of Tasks (MFT) 10
Multiprogramming with a Variable Number of Tasks (MVT) 11

JOB CONTROL PROCEDURES 13
Format of the Job Control Statements . . 14
Job Management 14
Coding Job Control Statements 14
Conventions for Character Delimiters . 15
Rules for Continuing Job Control Statements 15
JOB Statement 16
Priority Scheduling Job Parameters . . 17
EXEC statement 18
Options for the Compiler 21
Options for the Linkage Editor 22
Priority Scheduling System Parameters 23
DD Statement 24
Notation for Describing Job Control Statements 26
JOBLIB DD Statement 40
Delimiter Statement 40

COMPILER DATA SET REQUIREMENTS 41
SYSUT1, SYSUT2, SYSUT3, SYSUT4 41
SYSIN 41
SYSPRINT 41
SYSPUNCH 42
SYSLIN 42
SYSLIB 42

LINKAGE EDITOR DATA SET REQUIREMENTS . . 45
SYSLIN 45
SYSPRINT 45
SYSLMOD 46
SYSUT1 46
SYSLIB 46
User-Specified Data Sets 46

EXECUTION TIME DATA SET REQUIREMENTS . . 47
The DISPLAY Statement 47
The ACCEPT Statement 48
The EXHIBIT or TRACE Statement 48
Abnormal Termination Dump 48
User-Defined Files 48
File Names and Data Set Names 48
Specifying Information About a File . 49
The File Processing Techniques 49
Processing with QSAM 49

USING THE DD STATEMENT 69
Creating a Data Set 69
Retrieving Previously Created Data Sets 73
DD Statements that Specify Unit Record Devices 74
Cataloging a Data Set 75
Generation Data Groups 75
Naming Data Sets 76

A CHECKLIST FOR JOB CONTROL PROCEDURES . 77
Compilation 77
Linkage Editor 78
Execution Time 79

CATALOGED PROCEDURES 81
Calling Cataloged Procedures 81
Types of Cataloged Procedures 82
Programmer-Written Cataloged Procedures 82
IBM-Supplied Cataloged Procedures . . 82
Modifying Existing Cataloged Procedures 86
Overriding and Adding to Cataloged Procedures 86
Overriding and Adding to EXEC Statements 86
Overriding and Adding to DD Statements 87
Using The DDNAME Parameter 89
Examples of Using the DDNAME Parameter 89

CHECKPOINT/RESTART 91
Taking a Checkpoint 91
Checkpoint Methods 91
DD Statement Formats 91
Planning Checkpoints 93
Restarting a Program 93
The RD Parameter 93
Automatic Restart 93
Deferred Restart 94
The Checkpoint Macro-Instruction (CHKPT) 95

OUTPUT 97
Compiler Output 97
The Object Module 102
Linkage Editor Output 102
COBOL Load Module Execution Output . . .105
Requests for Output 106
Operator Messages 106
System Output 106

PROGRAM CHECKOUT 107
The Debug Language 107
Following the Flow of Control 107
Displaying Data Values During Execution 108
Testing a Program Selectively 109
Testing Changes and Additions to Programs 110
Dumps 110
Errors that can Cause a Dump 110
Using the Abnormal Termination Dump .112
Incomplete Abnormal Termination 117

Scratching Data Sets 118

PROGRAMMING TECHNIQUES 119
Data Division 119
Procedure Division 126

USING THE REPORT WRITER FEATURE 129
Record Length and Format 129
Using Control Items in Control Footings 129
Summing Technique 129
Levels 129
TALLY 129
Output Line Overlay 130
LINE-COUNTER 130
Page Breaks 130
With Code Clause 130
Control Footings and Page Format . . .130

USING THE SORT FEATURE 131
Sort DD Statements 131
Sort Input DD Statements 131
Sort Output DD Statements 131
Sort Work DD Statements 131
DD Statement Requirements for Sort Data Sets 132
System DD Statements 134
Sharing Devices between Tape Data Sets 135
Using More Than One Sort Statement in a Job 135
SORT Program Example 135
Cataloging Sort DD Statements 136
Sort Diagnostic Messages 136
Linkage with the Sort/Merge Program .136

LIBRARIES 137
Kinds of Libraries 137
Libraries Provided by the System . . .137
Other Libraries Recognized by the System 138
Automatic Call Library 138
COBOL Copy Library 139
Job Library 142
Libraries Created by the User 142
Creating and Changing Libraries 143

CALLING AND CALLED PROGRAMS 145
Specifying Linkage 145
Linkage in a Calling COBOL Program . .145
Linkage in a Called COBOL Program . .145
Linkage in a Calling or Called Assembler-Language Program 146
File-Name and Procedure-Name Arguments 148
Linkage Editing Programs 148
Specifying Primary Input 149
Specifying Additional Input 150
Linkage Editor Processing 151

Overlay Structures 152

ADDITIONAL FILE PROCESSING INFORMATION .155
The Data Control Block 155
Overriding DCB Fields 155
Identifying DCB Information 155
Error Processing for COBOL Files 156
Volume Labeling 158
Standard Volume Labels 159
Data Set Labels 159
Magnetic Tape Volume and Data Set Labels 159
Direct-Access Volume and Data Set Labels 160

MACHINE CONSIDERATIONS 161
Minimum Machine Requirements for the COBOL (F) Level Compiler 161
Multiprogramming with a Variable Number of Tasks (MVT) 161
Execution Time Considerations 162
Sort Feature Considerations 163

APPENDIX A: SAMPLE PROGRAM OUTPUT . . .165

APPENDIX B: COBOL LIBRARY SUBROUTINES .177
COBOL Library Conversion Subroutines . .177
COBOL Library Arithmetic Subroutines . .177
COBOL Library Input/Output Subroutines .177
Sort Feature Subroutine (IHDFSORT) . . .179
COBOL Library Subroutines for Other Verbs 179

APPENDIX C: FIELDS OF THE DATA CONTROL BLOCK 181

APPENDIX D: COMPILER OPTIMIZATION . . .189
Block Size for Compiler Data Sets . . .189
How Buffer Space Is Allocated to Buffers 189

APPENDIX E: INVOCATION OF THE COBOL (F) COMPILER AND COBOL (F) COMPILED PROGRAMS 191
Invoking the COBOL (F) Compiler . . .191
Invoking COBOL (F) Compiled Programs .192

APPENDIX F: SOURCE PROGRAM SIZE CONSIDERATIONS 193
Compiler Capacity 193
Linkage Editor Capacity 194

APPENDIX G: INPUT/OUTPUT ERROR CONDITIONS 195

APPENDIX H: DIAGNOSTIC MESSAGES 199
Compiler Diagnostic Messages 199
Object Time Messages 213

INDEX 215

FIGURES

Figure 1. Job Control Procedure . . . 13
Figure 2. JOB Statement 26
Figure 3. EXEC Statement 27
Figure 4. Compiler and Linkage Editor PARM Options 27
Figure 5. The DD Statement 28
Figure 6. Device Class Names Required for IBM-Supplied Cataloged Procedures 32
Figure 7. Delimiter Statement 40
Figure 8. Determining the File Processing Technique 50
Figure 9. Permissible COBOL Clauses for QSAM 52
Figure 10. DD Statement Parameters Applicable to QSAM Output Files 52
Figure 11. DD Statement Parameters Applicable to QSAM Input and I-O Files 52
Figure 12. Permissible COBOL Clauses for BSAM 57
Figure 13. DD Statement Parameters Applicable to BSAM Output Files 57
Figure 14. DD Statement Parameters Applicable to BSAM Input Files 57
Figure 15. Permissible COBOL Statements for BDAM 59
Figure 16. DD Statement Parameters Applicable to BDAM Input and I-O Files . 59
Figure 17. DD Statement Parameters Applicable to QISAM Output Files 60
Figure 18. DD Statement Parameters Applicable to QISAM and BISAM Input and I-O Files 61
Figure 19. Example of DD Statements for New Indexed Sequential Files 63
Figure 20. Parameters Frequently Used in Creating Data Sets 70
Figure 21. Parameters Frequently Used in Retrieving Previously Created Data Sets 73
Figure 22. Parameters Used to Specify Unit Record Devices 74
Figure 23. General Job Control Procedure for Compilation 77
Figure 24. General Job Control Procedure for a Linkage Editor Job Step 79
Figure 25. General Job Control Procedure for an Execution-Time Job Step 80
Figure 26. Example of Adding Procedures to the Procedure Library . . 83
Figure 27. Statements in the COBFC Procedure 85
Figure 28. Statements in the COBFLG Procedure 85
Figure 29. Statements in the COBFCLG Procedure 85
Figure 30. Examples of Compiler Output 97
Figure 31. Linkage Editor Output Showing Module Map and Cross Reference List 103
Figure 32. Execution Job Step Output .105
Figure 33. COBOL Program with Abnormal Termination Dump 114
Figure 34. Data Format Characteristics 125
Figure 35. Relationship of PICTURE to Storage Allocation 127
Figure 36. Treatment of Varying Values in a Data Item of PICTURE S9 . .127
Figure 37. Format of a Library . . .138
Figure 38. Entering Source Statements Into the Copy Library 139
Figure 39. Updating Source Statements in a Copy Library 140
Figure 40. COBOL Statements to Deduct FICA Tax 141
Figure 41. Changed COBOL Statements to Update FICA Example 142
Figure 42. Sample Linkage Coding Used in a Calling Assembler-Language Program 147
Figure 43. Save Area Layout and Contents 148
Figure 44. Sample Linkage Coding Used in a Called Assembler Language Program .149
Figure 45. Specifying Primary and Additional Input to the Linkage Editor .150
Figure 46. Sample Deck for Linkage Editor Overlay Structure 153
Figure 47. Links Between the Select Statement, the DD Statement, the Data Set Label, and the Input/Output Statements 156
Figure 48. Standard Tape Labels 160
Figure 49. Error Processing Summary for ISAM Files 198

TABLES

Table 1. Job Control Statements . . . 14
Table 2. Direct-Access Volume States . 36
Table 3. Data Set References 37
Table 4. Data Sets Used for Compilation 42
Table 5. Data Sets Used For Linkage Editing 45
Table 6. DEN Values 51
Table 7. Direct-Access Device Overhead Formulas 55
Table 8. Direct-Access Storage Device Capacities 55
Table 9. Direct-Access Device Track Capacity 56
Table 10. Glossary Definition and Usage 101
Table 11. Symbols Used in the Listing and Glossary to Define Compiler-Generated Information 102
Table 12. System Message Identification Codes 106
Table 13. Summary of DD Statement Parameters Required by the Sort Feature 133
Table 14. Summary of DCB Subparameters Required by the Sort Feature 133
Table 15. Linkage Registers 146
Table 16. Input/Output Error Processing Facilities 157
Table 17. Functions of COBOL Library Conversion Subroutine 178
Table 18. Functions of COBOL Library Arithmetic Subroutines 179
Table 19. Data Control Block for QSAM .182
Table 20. Data Control Block for BSAM .183
Table 21. Data Control Block for QISAM 184
Table 22. Data Control Block for QISAM 185
Table 23. Data Control Block for BISAM 186
Table 24. Data Control Block for BDAM .187

A COBOL (F) program is processed by the System/360 Operating System. The operating system consists of a number of processing programs and a control program. The processing programs include the COBOL (F) compiler, service programs, and user-written programs.

The control program supervises the execution of the processing programs; controls the location, storage, and retrieval of data; and schedules jobs for continuous processing.

A request to the operating system for facilities and scheduling of program execution is called a job. For example, a job could be execution of the COBOL (F) compiler to compile a program. A job consists of one or more job steps, each of which specifies execution of a program. A programmer makes these requests to the operating system by use of job control statements that may be punched into cards.

Each job is headed by a JOB statement which identifies the job. Each job step is headed by an EXEC statement which describes the job step and calls for execution. Included in each job step are DD statements, which describe data and request allocation of input/output devices.

The data processed by execution of any processing program must be in the form of a data set. A data set is a named, organized collection of one or more records that are logically related. Information in a data set may or may not be restricted to a specific type, purpose, or storage medium. A data set may be, for example, a source program, a library of subroutines, or a file of data records that is to be processed by a COBOL program.

A data set resides in one or more volumes. A volume is a unit of external storage that is accessible to an input/output device. For example, a volume may be a reel of tape or a disk pack.

To facilitate retrieval of a data set, the serial number of the volume upon which it resides can be entered, along with the data set name, in the system catalog of data sets. The catalog itself is a data set residing on one or more direct-access volumes. It is organized into indexes that relate each data set name to its location--the volume in which it resides and its position within the volume. Only the data set name need be specified to identify a cataloged data set to the system.

The catalog is originally created by a utility program. Once the catalog exists, any data set residing on either a direct-access or a magnetic tape volume can be cataloged automatically by use of a catalog subparameter in a DD statement that refers to the data set.

Several input/output devices grouped together and given a single name when the system is generated constitute a device class. Each device class can be referred to by a collective name. For example, one device class called SYSDA could consist of all the direct-access devices in the installation; another called SYSSQ could consist of all the direct-access devices and tape devices.

DATA SET ORGANIZATION

A data set used by a COBOL program can have one of five types of organization: sequential, indexed sequential, direct, relative, and partitioned. The first type (sequential) may be on any input/output device. All other types must be on direct-access devices.

1. A sequential data set is one in which records are organized solely on the basis of their successive physical positions, such as they are on tape.

2. An indexed sequential data set is one in which records are arranged in logical sequence (according to a key which is part of every record) on the tracks of a direct-access device. A separate index or set of indexes maintained by the system indicates the location of each record. This permits random, as well as sequential, access to any record.

3. A direct data set in COBOL is one in which records are referred to by use of relative track addressing. An actual key specifies the track relative to the first track allocated to the data set. A symbolic key identifies the record on the track.

4. A relative data set in COBOL is one in which records are referred to by use of relative record addressing. A symbolic key identifies the record loca-

tion relative to the first record in the data set.

5. A partitioned data set (PDS) is one that is composed of named, independent groups of sequential data, each of which is called a member. Each member has a simple name stored in a directory which is part of the data set and which contains the location of each member's starting point. Partitioned data sets are used to store programs, and are often referred to as libraries.

EXECUTING A COBOL PROGRAM

Three basic operations are performed to execute a COBOL program: compilation, linkage editing, and actual execution.

COMPILATION

Compilation is the process of translating a COBOL (F) source program into a series of instructions comprehensible to the computer, i.e., machine language. In operating system terminology, the input to the compiler, the source program, is called the source module. The output from the compiler, the compiled source program, is called the object module.

LINKAGE EDITING

The linkage editor is a service program that prepares object modules for execution. It can also be used to combine two or more separately compiled object modules into a format suitable for execution as a single program. The executable output of the linkage editor is called a load module, which must always be stored as a member of a partitioned data set.

In addition to processing object modules, the linkage editor can combine previously edited load modules, with or without one or more object modules, to form one load module.

During the process of linkage editing, external references between different modules are resolved.

EXECUTION

Actual execution is under supervision of the control program, which obtains a load module from a library, loads it into main storage, and initiates execution of the machine language instructions contained in the load module.

SYSTEM/360 OPERATING SYSTEM ENVIRONMENTS

System/360 Operating System offers three control programs. These are the Primary Control Program, Multiprogramming with a Fixed Number of Tasks, and Multiprogramming with a Variable Number of Tasks.

PRIMARY CONTROL PROGRAM (PCP)

The primary control program (PCP) of the Operating System provides all sequential scheduling features of the Job Control Language as specified in the publications IBM System/360 Operating System: Job Control Language, Form C28-6539, and IBM System/360 Operating System: Operator's Guide, Form C28-6540. It affords data management capability and contains a supervisor that provides for:

- Efficient overlapping of central processing unit operations and input/output channel activity
- Error checking and standard input/output error recovery procedures
- Supervision and processing of interruptions
- Supervision of requests for various services provided by the system

This control program provides for a single input job stream and the sequential processing of job steps through single task operations.

MULTIPROGRAMMING WITH A FIXED NUMBER OF TASKS (MFT)

The MFT control program divides storage into a number of discrete areas called partitions. Job steps are directed to these partitions using a priority scheduling system; that is, jobs are not read in as encountered in the job stream but according to a priority code. In addition to the facilities provided by the primary control program, the MFT control program provides for:

- priority scheduling of jobs using the class code
- concurrent scheduling and execution of up to 15 separately protected jobs

- reading one or more input streams

MULTIPROGRAMMING WITH A VARIABLE NUMBER OF TASKS (MVT)

The MVT control program divides storage into areas called regions. Like MFT, the MVT control program uses a priority scheduling system provides for concurrent execution of up to 15 jobs, and reads more than one input stream. In addition, the MVT control program:

- assigns storage regions on a variable basis according to a region code

For further information about the various optional features of the System/360 Operating System, see the publications IBM System/360 Operating System: Storage Estimates, Form C28-6551, and IBM System/360 Operating System: Concepts and Facilities, Form C28-6535.

The sequence of job control statements required to specify a job is called a job control procedure.

For example, the job control procedure shown in Figure 1 could be placed in the input stream to compile a COBOL source module.

JOB1 is the name of this job. The JOB statement indicates the beginning of a job.

STEP1 is the name of the single job step in the job. The EXEC statement specifies that the COBOL (F) compiler (IEQCBL00) is to execute the job. The statement also specifies that a card deck of the object module is to be produced (PARM=DECK).

The SYSUT1, SYSUT2, SYSUT3, and SYSUT4 DD statements define utility data sets used by the compiler to process the source module. The names of the data sets defined by SYSUT1, SYSUT2, SYSUT3, and SYSUT4 are &UT2, &UT3, and &UT4, respectively. &UT1, SYSUT1 must be on direct access (UNIT=SYSDA). The system is to allocate 40 tracks of space to SYSUT1. The other three utility data sets are assigned either to any available tape, in which case the SPACE parameter is ignored, or to a direct-access unit (UNIT=SYSSQ).

The SYSPRINT DD statement defines the data set that is to be printed. SYSOUT=A is the standard designation for data sets whose destination is the system output device, usually indicating that the data set is to be listed on a printer.

The SYSPUNCH DD statement defines the data set that is to be punched. SYSOUT=B is a standard designation for a card punch.

The SYSIN DD statement defines the data set (in this case the source module) that is to be input to the job step. The asterisk (*) indicates that the input data set follows in the input stream.

The delimiter (/*) statement separates data from subsequent control statements in the input stream.

Output from this job step includes any diagnostic messages associated with the compilation. They are printed in the data set specified by SYSPRINT.

Note: SYSDA, SYSSQ, B, and A are IBM-specified device class names. If they are to be used, they must be incorporated at system generation time. If SYSOUT=B is to be used, the unit name SYSCP must be specified at system generation time.

To avoid the repetition of rewriting statements and the possibility of error, procedures that are used frequently may be placed on a system library called the procedure library. A procedure contained in the procedure library is called a cataloged procedure. A cataloged procedure can be called for execution by placing in the input stream a simple procedure that may require only the JOB and EXEC statements.

If slightly modified, the procedure in the previous example can be cataloged, i.e., placed in the procedure library. For example, if it were cataloged and given the name CATPROC, it could be called for execution by placing the following procedure in the input stream:

```
//JOB1      JOB
//STEP1     EXEC  PGM=IEQCBL00,PARM=DECK
//SYSUT1    DD    DSNAME=&UT1,UNIT=SYSDA,SPACE=(TRK,(40))
//SYSUT2    DD    DSNAME=&UT2,UNIT=SYSSQ,SPACE=(TRK,(40))
//SYSUT3    DD    DSNAME=&UT3,UNIT=SYSSQ,SPACE=(TRK,(40))
//SYSUT4    DD    DSNAME=&UT4,UNIT=SYSSQ,SPACE=(TRK,(40))
//SYSPRINT  DD    SYSOUT=A
//SYSPUNCH  DD    SYSOUT=B
//SYSIN     DD    *
    (source deck)
/*
```

Figure 1. Job Control Procedure

```
//JOB2          JOB
//STEPA         EXEC    PROC=CATPROC
//STEP1.SYSIN DD      *
    (source deck)
/*
```

JOB2 is the name of the job. STEPA is the name of the single job step. The EXEC statement calls the cataloged procedure containing STEP1 to execute the job step (PROC=CATPROC).

The chapters "Using the DD Statement" and "A Checklist of Job Control Procedures" explain, with numerous examples, the preparation of job control procedures. The chapters "Compiler Data Set Requirements", "Linkage Editor Data Set Requirements", and "Execution Time Data Set Requirements" describe required and optional data sets for compilation, linkage editing, and execution time job steps. The chapter "Cataloged Procedures" provides information about using and modifying cataloged procedures, including three cataloged procedures supplied by IBM.

The following section, "Format of the Job Control Statements," shows the format and use of the parameters and subparameters that can be specified for each job control statement. Some parameters of the statements are described only briefly. For further information, see the publication IBM System/360 Operating System: Job Control Language, Form C28-6539. The syntactic format descriptions on the fold-out page in this chapter can be used as a reference for the exact format and for the use of each parameter.

Table 1. Job Control Statements

Statement	Function
JOB	Indicates the beginning of a new job and describes that job
EXEC	Indicates a job step and describes that job step; indicates the load module or cataloged procedure to be executed
DD	Describes data sets, and controls device and volume assignment
delimiter	Separates data sets in the input stream from control statements; it must follow each data set that appears in the input stream, e.g., after a COBOL source module punched deck

FORMAT OF THE JOB CONTROL STATEMENTS

The COBOL programmer uses the job control statements shown in Table 1 to compile, linkage edit, and execute programs.

JOB MANAGEMENT

Job control statements are processed by a group of operating system routines known collectively as job management. Job management routines interpret control statements, control the flow of jobs, and issue messages to both the operator and the programmer. Job management has two major components: a job scheduler and a master scheduler.

The specific facilities available through the job scheduler and the master scheduler depend on the scheduling level the installation selects during system generation. Schedulers are available at two levels: the sequential scheduler and the more powerful priority scheduler.

Sequential schedulers process job steps one at a time in the order of their appearance in the input stream. Operating systems with a primary control program (PCP) use sequential schedulers.

Priority schedulers process jobs according to their relative priority and available system resources. Systems that provide multiprogramming (the MFT or MVT environments) use priority schedulers.

CODING JOB CONTROL STATEMENTS

Job control statements are identified by the initial characters // or /* in card columns 1 and 2, and may contain three fields -- name, operation, and operand -- plus a possible comments field, as shown:

Applicable Format	Control Statements
// Name Operation Operand [Comment]	JOB,EXEC,DD
// Operation Operand [Comment]	EXEC,DD
/* [Comment]	delimiter

Name Field

The name contains from one through eight alphanumeric characters, the first of which

must be alphabetic. The name begins in card column 3. It is followed by one or more blanks. The name is used as follows:

1. To identify the control statement to the operating system.

2. To enable other control statements in the job to refer to information contained in the named statement.

3. To relate DD statements to files named in a COBOL source program.

Operation Field

The operation field is preceded and followed by one or more blanks. It may contain one of the following operation codes:

```
JOB
EXEC
DD
```

If the statement is a delimiter statement, there is no operation field and comments may start after one blank.

Operand Field

The operand field is preceded and followed by one or more blanks and may continue through column 71 and onto one or more continuation cards. It contains the parameters that give required and optional information to the operating system. The parameters are separated by commas. A blank in the operand field causes the system to treat the remaining data on the card as a comment. There are two types of parameters: positional and keyword.

Positional Parameters: Positional parameters are the first parameters in the operand field, and they must appear in the specified order. If a positional parameter is omitted and other positional parameters follow, the omission must be indicated by a comma. If other positional parameters do not follow, no comma is needed.

Keyword Parameters: A keyword parameter may be placed anywhere in the operand field following the positional parameters. A keyword parameter consists of a keyword, followed by an equal sign, followed by a single value or a list of subparameters. If there is a subparameter list, it must be enclosed in parentheses or single quotation marks; the subparameters in the list must be separated by commas. Keyword parameters may appear in any order.

Subparameters: Subparameters are either positional or keyword. Positional and keyword subparameters for job control statements are shown in Figures 2, 3, and 5 on the fold-out page. Positional subparameters appear first in the parameter and must be in the specified order. If a positional subparameter is omitted and other positional subparameters follow, a comma must indicate the omission.

Note: The fold-out page is to be used in conjunction with the text for easy reference.

Comments Field

A comment must be separated from the last parameter (or the /* in a delimiter statement) by one or more blanks and may appear in the remaining columns up to and including column 71. A comment may be continued onto one or more continuation cards. Comments can contain blanks.

CONVENTIONS FOR CHARACTER DELIMITERS

Commas, parentheses, and blanks are interpreted as character delimiters. If they are not intended by the programmer to be used as delimiters, the fields in which they appear must be enclosed in single quotation marks, indicating that the enclosed information is to be treated as a single field. When an apostrophe (or a single quotation mark, since the same character is used for either) is to be contained within such a field, it must be shown as two consecutive single quotation marks (5-8 punch), not as a double quotation mark (7-8 punch). For example,

```
'WM. O' 'CONNOR'
```

This convention applies to three fields: programmer's name in the JOB statement, information in the PARM parameter of the EXEC statement, and accounting information in the JOB and EXEC statements.

RULES FOR CONTINUING JOB CONTROL STATEMENTS

The operand field or a comment field can be continued from one card to another. The rules for continuation are:

1. Interrupt the operand after a complete parameter (after the separating comma if another parameter is to follow). A comment may be interrupted at any convenient place before column 72.

2. Insert a nonblank character in column 72.

3. Insert slashes in columns 1 and 2 of the next card.

4. Continue the interrupted operand or comments in column 16 of that card.

These same rules apply to subsequent continuations if more than one continuation card is required.

JOB STATEMENT

The JOB statement (Figure 2) is the first statement in the sequence of control statements that describe a job. The JOB statement contains the following information:

1. Name of the job.

2. Accounting information relative to the job.

3. Programmer's name.

4. Indication of whether or not the job control statements are to be printed on the system output listing.

5. Conditions for terminating the execution of the job.

6. For priority scheduling systems: job priority assignment, job scheduler message class, and, for the MVT environment, main storage region size.

7. Parameters specifying checkpoint and restart procedures for the entire job.

jobname
: must always be specified; it identifies the job to the operating system. No two jobs being handled concurrently by a priority scheduler should have the same jobname.

account-number and accounting-information
: contain the installation account number and any parameters to be passed to the installation accounting routines. Such routines are written at an installation and inserted in the operating system when it is generated.

programmer-name
: identifies the person responsible for the job.

MSGLEVEL
: indicates the type of control statement messages the programmer wishes to receive from the control program.

MSGLEVEL=1
: indicates that all control statements, as well as control statement errors and diagnostic messages, are to be listed.

MSGLEVEL=0
: indicates that only control statement errors and diagnostic messages are to be listed. This is the default condition.

COND=((code,operator)[,(code,operator)]...)
: specifies conditions under which a job is to be terminated. Up to eight different tests, each consisting of a code and an operator, may be specified to the right of the equal sign. The operator indicates the algebraic relationship between the code placed in the JOB statement and the codes that may be issued by each completed job step. The code or codes specified are compared with the return code at the completion of each job step; if any condition is met, the remaining steps of the job are bypassed. (See the COND parameter on the EXEC statement for a discussion of the operator values and the codes issued by the compiler and linkage editor at the end of a job step.)

RD=request
: is used with the PCP or MVT environments. This is the restart definition parameter. The programmer uses this parameter to tell the operating system:

  a. whether or not to take checkpoints during execution of a program, and

  b. whether or not to restart a program which has been interrupted.

  A checkpoint is taken by periodically recording the contents of storage and registers during a program's execution. The ability to take a checkpoint is provided by the RERUN clause in the COBOL language. Checkpoints are recorded onto a checkpoint data set. In the event of an interruption, the programmer may restart his program at a checkpoint rather than at the beginning of his job step.

See the RD parameter on the EXEC statement for a discussion of the forms that RD=request may take.

RESTART
is used with the PCP or MVT environments. This parameter specifies that the program may restart processing at a specified point. Restart may occur at the beginning of a job step or at a checkpoint within a job step. In order to restart at a point within a job step, a program must previously have had a checkpoint record written.

RESTART=*
indicates that the restart is to occur at the first job step of the job.

RESTART=stepname
indicates that restart is to occur at the beginning of the job step designated in stepname.

RESTART=(stepname[.procstepname][,checkid])
indicates that restart is to occur at the point specified within the job step designated by stepname. The subparameter procstepname is used to restart at a point within a cataloged procedure. The subparameter checkid is used to restart at the checkpoint identified by checkid. Checkid is expressed as a 1-16 character name usually assigned by the operating system at the time a checkpoint is taken.

Notes:

- If RESTART is omitted, the program is executed from the first job step.
- Generation data sets created and cataloged in steps preceding the restarted step must be referred to by subtracting one from their generation numbers, i.e., generation data set number 0 is referred to as -1.
- The PGM and COND parameters of the EXEC statement and the SUBALLOC and VOLUME=REF parameters of the DD statement cannot be used in or following the restart job step if they contain values of the form stepname or stepname.procstep referring to a step preceding the restart step.

PRIORITY SCHEDULING JOB PARAMETERS

CLASS=jobclass
assigns a job class to a job. Up to three job classes are associated with each partition. The term jobclass is replaced with an alphabetic character A through O. The use of this parameter and the meaning of the characters A through O are to be determined by each installation. If this parameter is omitted or CLASS A is specified, the default job class of A is assigned to the job.

PRTY=job priority
assigns other than the standard job priority established by an installation; job priority is expressed as a decimal number from 0 through 14.

Whenever possible, priority 13 should be avoided. This is used by the system to expedite processing of exceptional jobs. It is also intended for other special uses by future features of systems with priority schedulers. If this parameter is omitted, the standard job priority is assigned to the job.

MSGCLASS=classname
specifies an output class, other than the standard class A, to which all messages issued by the job scheduler are routed. An output writer assigned to the output class transfers this data to a specific device. Classname is expressed as either an alphabetic (A through Z) or numeric (0 through 9) character. If this parameter is omitted or is coded MSGCLASS=A, job scheduler messages are routed to the standard output class A.

TYPRUN=HOLD
temporarily prevents a job from being selected for processing. When this is specified, the job is held until a RELEASE command is issued by the operator. TYPRUN=HOLD is particularly useful when one job must be run after another job has terminated.

REGION=nnnnnK
is used with the MVT environment only. This parameter specifies the size of the main storage region to be allocated to a job that requires an unusual amount of main storage. The symbol nnnnn represents the number, in decimal, of 1024-byte areas to be allocated to the job, e.g., REGION=52K. This number must take into account the storage requirements of resident control functions and cannot exceed 16,384. It should be specified as an even number. (If specified as an odd number, the system treats it as the next highest even number.) If this parameter is omitted, a default region size is allocated.

For additional information on the REGION parameter see the chapter "Machine Considerations."

Note: If different region sizes for each step in the job are desired, the REGION parameter in the EXEC statement associated with each step can be coded instead (see "EXEC Statement").

ROLL=(x,y)
is used with the MVT environment only. This parameter allocates additional main storage to a job step whose own region does not contain any more available space. In order to allocate this additional space to a job step, another job step may have to be rolled out, i.e., temporarily transferred to secondary storage. When x is replaced with YES, each of the programmer's job steps can be rolled out; when x is replaced with NO, the job steps cannot be rolled out. When y is replaced with YES, this indicates that each job step can cause rollout; when y is replaced with NO, the job steps cannot cause rollout. (If additional main storage is required for the job's steps, YES must be specified for y.) If this parameter is omitted, ROLL=(YES,NO) is assumed. ROLL parameters can also be coded in EXEC statements, but are superseded by a ROLL parameter coded in the JOB statement.

EXEC STATEMENT

The EXEC statement (Figure 3) defines a job step and calls for execution. It contains the following information:

1. The name of a load module or the name of a cataloged procedure that contains the name of a load module that is to be executed. The load module can be the COBOL (F) compiler, the linkage editor, or any COBOL (F) program in load module form.
2. Accounting information for this job step.
3. Conditions for bypassing the execution of this job step.
4. For priority scheduling systems: computing time for a job step or cataloged procedure step, and main storage region size.
5. Compiler or linkage editor options chosen for the job step.
6. Parameters specifying checkpoint and restart procedures for the job step.

Note: If the information specified is normally delimited by parentheses, but contains blanks, parentheses, or equal signs, it must be delimited by single quotation marks instead of parentheses.

stepname
is the name of the job step. It is required when information from this control statement is to be referred to in a later job step.

PGM=
specifies the program to be executed in the job step.

Specifying the Compiler:

PGM=IEQCBL00
indicates that the COBOL (F) compiler is the processing program to be executed in the job step.

Specifying the Linkage Editor:

PGM=IEWL
indicates that the linkage editor is the processing program to be executed in the job step.

Specifying a Load Module:

PGM=progname
indicates that some other program is to be executed, normally a COBOL program that is in load module format.

PGM=*.stepname.ddname
specifies a load module defined as a data set in a DD statement of a previous job step. The * indicates the current job; stepname is the name of the previous job step that refers to the program; and ddname is the ddname of the DD statement within the previous job step that defines the load module. The job step referred to must be in the same job.

This form is used, for example, when the previous job step is the linkage editor and the output from the linkage editor is the load module that is to be executed in this job step.

{PROC=procname}
{procname }
indicates that a cataloged procedure is to be called. The procname is the unqualified name of the cataloged procedure (see "Using the DD Statement" for a discussion of qualified names).

PGM=*.stepname.procstep.ddname
indicates that the name of the program to be executed is to be taken from a DD statement of a cataloged procedure that has been previously called within the job. The * indicates the current job; stepname is the name of the job step that called the cataloged procedure; procstep is the name of a step within the procedure; ddname is the name of a DD statement within the procedure step. (The stepname cannot refer to a job step in another job.)

ACCT=(accounting information)
specifies accounting information to the installation accounting routines for this job step.

If both the JOB and EXEC statements contain accounting information, the installation accounting routines decide how the accounting information shall be used for the job step.

ACCT.procstep=(accounting information)
is used to pass accounting information to a step within a cataloged procedure.

A maximum of 142 characters may be written between the parentheses or single quotation marks that enclose the list of options.

{COND | COND.procstep}
states conditions, depending upon return codes issued by preceding job steps, under which the current step or steps within a cataloged procedure are to be bypassed. If any of the conditions are met, the job step is bypassed. For a COBOL programmer, the return codes are from a compilation or linkage editor job step, since a COBOL program itself cannot generate the return codes.

Bypassing the Current Job Step
COND=((code,operator[,stepname])...)
The stepname identifies the previously executed job step that issued the return code to be tested. If stepname is not specified, code is compared to the return codes issued by all preceding steps in the job.

Bypassing Steps within a Cataloged Procedure
COND
COND.procstep=((code,operator[stepname [.procstep]])...)
If the job step calls for execution of a cataloged procedure, the COND parameter (without the procstep qualification) replaces all COND parameters in each step of the procedure. The COND.procstep parameter applies to a job step contained within the cataloged procedure; procstep identifies this procedure step. This parameter replaces the entire COND parameter in the procedure step. It may be repeated in the EXEC statement once for each step in the cataloged procedure.

Note the difference between procstep in COND.procstep and stepname. The procstep identifies the job step that may or may not be bypassed. The stepname identifies the previously executed job step that issued the return code to be tested. If the return code of a step in a cataloged procedure is to be tested, the procedure stepname must be qualified by the name of the step that called for execution of the procedure (i.e., stepname.procstep). If stepname is not specified, the code is compared with the return codes issued by all preceding steps in the job.

code
a number, specified by the programmer, to be compared with the job step return code.

operator
specifies the test to be made of the relation between the programmer-specified code and the job step return code. It is similar to the relation test written in a COBOL IF statement; that is, it is interpreted as "if programmer-specified code is (operator) 'stepname' return code, then the current job step is to be bypassed." There are six operators:

Operator	Meaning
GT	greater than
GE	greater than or equal to
EQ	equal to
NE	not equal to
LT	less than
LE	less than or equal to

As many as eight conditions to be tested may be specified in either form of the COND parameter.

The return codes for both the compiler and linkage editor are:

00 Normal conclusion.
04 Warning messages have been listed, but program is executable.
08 Error messages have been listed; execution may fail.
12 Severe errors have occurred; execution is impossible.
16 Terminal errors have occurred; execution of the processor has been terminated.

The compiler issues a return code of 16 when:

- BASIS member-name is specified and no member of that name is found, or
- COPY member-name is specified and no SYSLIB statement is included, or
- Not enough core storage is available for the tables required for compilation.

The return codes have a correlation with the severity level of the error messages. With linkage editor, for example, the rightmost digit of the message number states the severity level; this number is multiplied by 4 to get the appropriate return code. With the COBOL compiler, 04 to 16 are equal to the severity codes W, C, E and D, respectively.

RD[.procstepname]=request
is used with the PCP or MVT environments. This is the restart definition parameter. The programmer uses this parameter to tell the operating system:

a. whether or not to take checkpoints during execution of a job step, and

b. whether or not to restart a job step which has been interrupted.

A checkpoint is taken by periodically recording the contents of storage and registers during a program's execution. The ability to take a checkpoint is provided by the RERUN clause in the COBOL language. Checkpoints are recorded onto a checkpoint data set. In the event of an interruption, the programmer may restart his program at a checkpoint rather than at the beginning of his job step.

Request may take one of the following forms:

R (Restart)
to indicate that an automatic restart is to occur as soon as an interrupt terminates processing. Restart will occur at the latest checkpoint taken. In order to take checkpoints, the programmer must specify at least one RERUN clause in his program.

RNC (Restart, No Checkpoint)
to indicate an automatic restart may occur only at the beginning of a job step. No checkpoints are taken; no RERUN clause in the COBOL program is necessary.

NR (No Restart)
to indicate that no restart is to occur in the event of an interrupt. Checkpoints may still be taken during the course of the program's execution in the event the programmer may want to restart his program at a later time. The programmer must specify at least one RERUN clause in order to take checkpoints.

NC (No Checkpoint)
to indicate that no restart is to occur and no checkpoint is to be taken. No RERUN clause is necessary.

Notes:

- If the RD parameter is omitted, the following action occurs: if no checkpoints were taken before an interruption, the program cannot restart; if checkpoints were taken, then automatic restart will occur in the event of an interruption.
- If no RERUN clause is specified in the user's program, no checkpoints are written, regardless of the disposition of the RD parameter.
- Systems operating under the MVT environment must also specify the MSGLEVEL=1 parameter in the JOB statement if the RD parameter indicates a restart option.
- If RD is coded on the JOB statement, it will override all RD parameters on any EXEC statement.
- The programmer may use the form RD. procstepname to restart at a procedure step within a cataloged procedure. The programmer may code as many parameters of this form as there are steps in the cataloged procedure.

Note: A COBOL load module cannot generate a return code.

PARM=(option[,option]...)
passes options (Figure 4 on the fold-out page) to the compiler or linkage editor when either is called by the PGM parameter in the EXEC statement or to the first step in a cataloged procedure.

PARM.procstep=(option[,option]...)
passes options to a compiler or linkage editor step within the named cataloged procedure step. Any PARM parameter already appearing in the procedure step is deleted, and the PARM parameter that is passed to the procedure step is inserted.

A maximum of 40 characters may be written between the parentheses or single quotation marks that enclose the list of options. The compiler selects the valid options of the PARM field for processing by looking for three significant characters of each key option word. When the keyword is identified, it is checked for the presence or absence of NO, as appropriate. The programmer can make the most efficient use of the option field by using the significant characters instead of the entire option. The following table lists the significant characters for each option (for an explanation of each option see "Options for the Compiler" below):

Option	Significant Characters
COPY	COP
BASIS	BAS
LINECNT	CNT
SEQ	SEQ
FLAGE(W)	LAG,LAGW
SIZE	SIZ
BUF	BUF
SOURCE	SOU
MAP	MAP
DECK	DEC
LOAD	LOA
SPACE	ACE
DMAP	DMA
PMAP	PMA
SUPMAP	SUP
CLIST	CLI

OPTIONS FOR THE COMPILER

The default options indicated by an underscore in the following discussion can be changed within each installation at system generation time. The format of the PARM parameter is illustrated in Figure 4.

Note: When a subparameter contains an equal sign, the entire information field of the PARM parameter must be enclosed by single quotation marks instead of parentheses (e.g., PARM='SIZE=160000,MAP').

SIZE=yyyyyyy
indicates the amount of main storage, in bytes, available for compilation (see "Machine Considerations").

BUF=yyyyyy
indicates the amount of main storage to be allocated to buffers. If both SIZE and BUF are specified, the amount allocated to buffers is included in the amount of main storage available for compilation (see Appendix D for information about how buffer size is determined).

{SOURCE }
{NOSOURCE}
indicates whether or not the source module is to be listed.

{CLIST }
{NOCLIST}
indicates whether or not a condensed listing is to be produced. If specified, the procedure portion of the listing will contain only the source card number, location of the first generated instruction, and EBCDIC verb name for each verb.

{MAP }
{NOMAP}
indicates whether or not a glossary, global tables, object code listing, and assembler language expansion of the source module are to be listed.

{DMAP }
{NODMAP}
indicates whether or not a glossary is to be listed.

{PMAP }
{NOPMAP}
indicates whether or not global tables and assembler language expansion of the source modules are to be listed.

Note: PMAP and DMAP taken together are the equivalent of MAP. If the user wants both PMAP and DMAP, he can specify MAP in the PARM parameter. If he wants only PMAP, he can specify either PMAP or NODMAP. If a conflict exists in the PARM parameter, the last encountered option will be selected. For example, NODMAP followed by MAP results in the selection of both PMAP and DMAP. MAP followed by NODMAP results in the selection of PMAP only.

{LOAD / NOLOAD}
indicates whether or not the object module is to be placed in a direct-access or a tape volume so that the module can be used as input to the linkage editor. If option is used, a SYSLIN DD statement must be specified.

{DECK / NODECK}
indicates whether or not the object module is to be punched. If the DECK option is used, a SYSPUNCH DD statement must be specified.

{SEQ / NOSEQ}
indicates whether or not the compiler is to check the sequence of the source module statements. If the statements are not in sequence, a message is printed.

LINECNT=nn
indicates the number of lines to be printed on each page of the compilation output listing. The number specified by nn must be a 2-digit integer from 01 to 99. If the LINECNT option is omitted, 60 lines are printed on each page of the output listing.

{FLAGW / FLAGE}
indicates the type of messages that are to be listed for the compilation. FLAGW indicates that all warning and diagnostic messages are to be listed. FLAGE indicates that all diagnostic messages are to be listed, but the warning messages are not to be listed.

{SUPMAP / NOSUPMAP}
indicates whether or not the object code listing, and object module and linkage editor decks are to be suppressed if an E (or D) level message is generated by the compiler.

{BASIS / NOBASIS}
indicates whether or not a BASIS card may be present in the source module that is to be compiled. BASIS specifies that a BASIS card may be in the source module. NOBASIS specifies that no BASIS statement is in the source module. This option is used to optimize buffer allocation at compilation time.

{COPY / NOCOPY}
indicates whether or not a COPY or INCLUDE statement may be in the source module. COPY specifies that one or both may appear. NOCOPY specifies that no INCLUDE or COPY statements are in the source module. This option is used to optimize buffer allocation at compilation time.

{SPACE1 / SPACE2 / SPACE3}
indicates what type of spacing is to be used on the listing that is generated when SOURCE is specified. SPACE1 specifies single spacing, SPACE2 specifies double spacing, and SPACE3 specifies triple spacing.

For examples of what the SOURCE, MAP, DECK, and SEQ options produce, refer to "Output."

OPTIONS FOR THE LINKAGE EDITOR

MAP
indicates that a map of the load module is to be listed. If MAP is specified, XREF cannot be specified, but both can be omitted.

XREF
indicates that a cross reference list and a module map is to be listed. If XREF is specified, MAP cannot be specified.

LIST
indicates that any linkage editor control statements associated with the job step are to be listed.

OVLY
indicates that the load module is to be in the format of an overlay structure.

The format of the PARM parameter is illustrated in Figure 4. For examples of what the MAP, XREF, and LIST options produce, refer to the chapter "Output." Linkage editor control statements and overlay structures are explained in the chapter "Calling and Called Programs." There are other PARM options for linkage editor processing that describe additional processing options and special attributes of the load module (see the publication IBM System/360 Operating System: Linkage Editor, Form C28-6538).

TIME=(minutes,seconds)
assigns a limit to the computing time used by a single job step, a cataloged procedure, or a cataloged procedure step (see "Notes" below). Such an assignment is useful in a multiprogramming environment where more than one job has access to the computing system. The terms minutes and seconds represent the maximum number of minutes and seconds allotted for execution of the job step.

Notes:

- If the job step requires use of the system for 24 hours (1440 minutes) or longer, the programmer should specify TIME=1440. Using this number suppresses timing. The number of seconds cannot exceed 59.
- If the time limit is given in minutes only, the parentheses need not be coded; e.g., TIME=5.
- If the time limit is given in seconds, the comma must be coded to indicate the absence of minutes; e.g., TIME=(,45).
- When the job step uses a cataloged procedure, a time limit for a single procedure step can be set by qualifying the key word TIME with the procedure step name; i.e., TIME.procstep=(minutes,seconds). This specification overrides the TIME parameter in the named procedure step if one is present. As many parameters of this form can be coded as there are steps in the cataloged procedure.
- To set a time limit for an entire procedure, the TIME key word is left unqualified. This specification overrides all TIME parameters in the procedure if any are present.

If this parameter is omitted, the standard job step time limit is assigned.

REGION=nnnnnK (MVT only)
specifies the size of the main storage region to be allocated to the associated job step, for jobs that require an unusually large amount of main storage. The symbol nnnnn represents the number, in decimal, of 1024-byte areas to be allocated to the job step; e.g., REGION=52K. This number must take into account the storage requirements of resident control functions and cannot exceed 16,384K bytes. It should be specified as an even number. (If specified as an odd number, the system treats it as the next highest even number.)

Notes:

- If a REGION parameter has been specified in the JOB statement, REGION parameters in the job's EXEC statements are ignored.
- When a job step uses a cataloged procedure, a region size for a single procedure step can be requested by qualifying the REGION parameter with the procedure stepname, i.e., REGION.procstep=nnnnnK. This specification overrides the REGION parameter in the named procedure step if one is present. As many parameters of this form can be coded as there are steps in the cataloged procedure.
- To request a single region size for an entire cataloged procedure, the REGION parameter is left unqualified. This specification overrides all REGION parameters in the procedure, if any are present.

If this parameter is omitted, a default region size is allocated.

ROLL=(x,y)
is used for priority scheduling systems only. This parameter allocates additional main storage to a job step whose own region does not contain any more available space. In order to allocate this additional space to a job step, another job step may have to be rolled out, i.e., temporarily transferred to secondary storage. When x is replaced with YES, the job step can be rolled out; when x is replaced with NO, the job step cannot be rolled out. When y is replaced with YES, the job step can cause rollout; when y is replaced with NO, the job step cannot cause rollout. (If additional main storage is required for the job step, YES must be specified for y.) If this parameter is omitted, ROLL=(YES,NO) is assumed.

Notes:

- If the ROLL parameter is specified in the JOB statement, the ROLL parameter in the EXEC statements is ignored.
- When a job step uses a cataloged procedure, it can be indicated whether or not a single procedure

step has the ability to be rolled out and to cause rollout of another job step. To indicate this the procedure stepname, i.e., ROLL.procstepname, is included as part of the ROLL parameter. This specification overrides the ROLL parameter in the named procedure step, if one is present. As many parameters of this form can be coded as there are steps in the cataloged procedure.

- To indicate whether or not all of the steps of a cataloged procedure have the ability to be rolled out and to cause rollout of other job steps, the ROLL parameter can be coded without a procedure stepname. This specification overrides all ROLL parameters in the procedure, if any are present.

DD STATEMENT

The data definition (DD) statement (Figure 5) identifies each data set that is to be used in a job step, and it furnishes information about the data set. The DD statement specifies input/output facilities required for use of the data set; it also establishes a logical relationship between the data set and input/output references in the program named in the EXEC statement for the job step.

Parameters used most frequently for COBOL programs are discussed in detail. The other parameters (e.g., SEP and AFF) are mentioned briefly. For further information see the publication, IBM System/360 Operating System: Job Control Language, Form C28-6539.

NOTATION FOR DESCRIBING JOB CONTROL STATEMENTS

The notation used in this publication to define the syntax of job control statements is as follows:

1. The set of symbols below define control statements, but they are never written in an actual statement.

hyphen	-	joins lower-case letters, words, and symbols to form a single variable
or symbol	|	indicates alternatives
braces	{ }	indicate a group of related items, only one of which is required
brackets	[ ]	indicate optional items. Brackets are also used with alternatives to indicate that a default is assumed if no alternative is listed
ellipsis	...	indicates that the preceding item or group of items can be repeated
superscript	1	indicates a footnote

2. Stacked items, enclosed in either brackets or braces, represent alternative items. No more than one of the stacked items can be written by the programmer.

3. Upper-case letters and words, numbers, and the set of symbols listed below are written in an actual control statement exactly as shown in the statement definition. (Any exceptions to this rule are noted in the definition of a control statement.)

single quotation mark	'
asterisk	*
comma	,
equal sign	=
parentheses	( )
period	.
slash	/

4. An underscore indicates a default option. If an underscored alternative is selected, it need not be written in the actual statement.

Note: Many of these defaults can be changed at system generation time.

5. Lower-case letters, words, and symbols appearing in a control statement definition represent variables for which specific information is substituted in the actual statement.

6. Blanks are used in the figures on the fold-out page to improve the readability of control statement definitions. In actual statements, blanks would be interpreted as delimiters.

Name	Operation	Operand
		Positional Parameters
//jobname	JOB	[([account-number] [,accounting-information])[1] [2] [3]]
		[,programmer-name][4] [5]
		Keyword Parameters
		[MSGLEVEL=1] / [MSGLEVEL=0]
		[CLASS=jobclass]
		[COND=((code,operator) [,(code,operator)]...[6])[7]]
		[RD=request]
		[RESTART={*, stepname[.procstepname], (stepname[.procstepname][,checkid])}]
		[PRTY=job priority]
		[MSGCLASS=classname]
		[REGION=nnnnnK]
		[ROLL=x,y)]
		[TYPRUN=HOLD]

[1]If the information specified (account-number and/or accounting-information) contains blanks, parentheses, or equal signs, the information must be delimited by single quotation marks instead of parentheses.
[2]If only account-number is specified, the delimiting parentheses may be omitted.
[3]The maximum number of characters allowed between the delimiting quotation marks is 144.
[4]If programmer-name contains commas, parentheses, apostrophes, or blanks, it must be enclosed within single quotation marks.
[5]The maximum number of characters allowed for programmer-name is 20.
[6]The maximum number of repetitions allowed is 7.
[7]If only one test is specified, the outer pair of parentheses may be omitted.

• Figure 2. JOB Statement

```
                Positional Subparameters

[ VOLUME=([PRIVATE],[RETAIN],[volume sequence number],[volume count]
                Keyword Subparameters
  [,SER=(volume-serial-number[volume-serial-number][9]...)]
  [,REF=  { dsname                          }                    ] ) ]
          { *.ddname                        }
          { *.stepname.ddname               }
          { *.stepname.procstep.ddname      }

 [ LABEL=([data-set-sequence-number],  { NL  }                    ]
                                       { SL  }
                                       { NSL }
  [,EXPDT=yyddd] )
  [,RETPD=xxxx ]

 [ DISP=( [NEW]      [,DELETE ]          )                        ]
          [OLD]      [,KEEP   ]
          [SHR]      [,PASS   ]
          [MOD]      [,CATLG  ]
                     [,UNCATLG]

  SYSOUT=classname
  SYSOUT=(x[,program-name][,form-no.])
```

[1]The name field must be blank when concatenating data sets.
[2]All parameters are optional to allow a programmer flexibility in the use of the DD statement; however, a DD statement with a blank operand field is meaningless.
[3]If the positional parameter is specified, keyword parameters cannot be specified.
[4]If subparameter-list consists of only one subparameter and no leading comma (indicating the omission of a positional subparameter) is required, the delimiting parentheses may be omitted.
[5]If subparameter-list is omitted, the entire parameter must be omitted.
[6]See User Defined Files for the applicable subparameters.
[7]See the publication IBM System/360 Operating System: Job Control Language, Form C28-6539.
[8]If only name is specified, the delimiting parentheses may be omitted.
[9]If only one volume-serial-number is specified, the delimiting parentheses may be omitted.
[10]The SEP and AFF parameters should not be confused with the SEP and AFF subparameters of the UNIT parameter.

'igure 5. The DD Statement (Part 2 of 2)

Name	Operation	Operand
//[stepname][1]	EXEC	Positional Parameters {PGM=progname PGM=*.stepname.ddname PROC=procname procname PGM=*.stepname.procstep.ddname} Keyword Parameters [{ACCT[2] ACCT.procstep} = (accounting-information)[3 4 5]] [{COND[2] COND.procstep} = ((code,operator[,stepname[.procstep]])...)[6 7]] [RD[.procstepname]=request] [{PARM[2] PARM.procstep} = (option[,option]..)[3 8 9]] [{TIME TIME.procstep} = (minutes,seconds)] [REGION REGION.procstep = nnnnnK] [{ROLL ROLL.procstep} = (x,y)]

[1]stepname is required when information from this control statement is referred to in a later job step.
[2]If this format is selected, it may be repeated in the EXEC statement once for each step in the cataloged procedure.
[3]If the information specified contains any special characters except hyphens, it must be delimited by single quotation marks instead of parentheses.
[4]If accounting-information contains any special characters except hyphens, it must be delimited by single quotation marks instead of parentheses.
[5]The maximum number of characters allowed between the delimiting quotation marks or parentheses is 142.
[6]The maximum number of repetitions allowed is 7.
[7]If only one test is specified, the outer pair of parentheses may be omitted.
[8]If the only special character contained in the value is a comma, the value may be enclosed in parentheses or in quotation marks; otherwise, special characters are enclosed in quotation marks.
[9]The maximum number of characters allowed between the delimiting quotation marks or parentheses is 40.

• Figure 3. EXEC Statement

Compiler:

PARM=([SIZE=yyyyyyy] [,BUF=yyyyyy] [SOURCE/,NOSOURCE] [,MAP/,NOMAP] [,DMAP/,NODMAP] [,PMAP/,NOPMAP] [,SUPMAP/,NOSUPMAP]
[,LOAD/,NOLOAD] [,DECK/,NODECK] [,SEQ/,NOSEQ] [,LINECNT=nn] [,CLIST/,NOCLIST]
[,FLAGW/,FLAGE] [,BASIS/,NOBASIS] [,COPY/,NOCOPY] [SPACE1/SPACE2/,SPACE3])[1 2 3]

Linkage Editor:

PARM =([MAP/XREF] [,LIST] [,OVLY])

[1]If the information specified contains any special characters, it must be delimited by single quotation marks instead of parentheses.
[2]If the only special character contained in the value is a comma, the value may be enclosed in parentheses or quotation marks.
[3]The maximum number of characters allowed between the delimiting quotation marks or parentheses is 40.

• Figure 4. Compiler and Linkage Editor PARM Options

```
Name                              Operation
// [ddname          ] 1           DD
   [procstep.ddname ]

Operand2

        Positional Parameters

  [*    ] 3
  [DATA ]
  [DUMMY]

        Keyword Parameters 4 5

 [DDNAME=ddname]

  [           (dsname                         ) ]
  [           (dsname(element)                ) ]
  [           (*.ddname                       ) ]
  [ DSNAME=   (*.stepname.ddname              ) ]
  [           (*.stepname.procstep.ddname     ) ]
  [           (&name                          ) ]
  [           (&name(element)                 ) ]

 [       [dsname                    ]                         ] 6
 [ DCB=( [*.ddname                  ] [,subparameter-list])   ]
 [       [*.stepname.ddname         ]                         ]
 [       [*.stepname.procstep.ddname]                         ]

  [SEP=(subparameter list)] 7     10
  [AFF=ddname             ]

                  Positional Subparameters          Keyword Subparameters

[UNIT=(name[,[n|P][,DEFER]] [,SEP=(list of up to 8 ddnames)]) 8]10
[                                                              ]
[UNIT=(AFF=ddname)                                             ]

                                 Positional Subparameters

[SPACE=({TRK                   }  ,(primary-quantity[,secondary-quantity],              ]
[       {CYL                   }                                                        ]
[       {average-record-length }                                                        ]
[                                                                                       ]
[                                                  [,MXLG  ]                            ]
[             [directory- or index-quantity])[,RLSE]  [,ALX   ]  [, round]      )         ]
[                                                  [,CONTIG]                            ]
[                                                                                       ]
[SPACE=(ABSTR,(quantity,beginning-address[,directory or index quantity]))               ]
[                                                                                       ]
[             [CYL                                                                    ] ]
[SPLIT=(n,    [average-record-length     ,(primary-quantity][,secondary-quantity])    ])]
[                                                                                       ]
[SUBALLOC=(   {TRK                  }     ,(primary-quantity[,secondary-quantity]       ]
[             {CYL                  }                                                   ]
[             {average-record-length}                                                   ]
[                                                                                       ]
[                                     {ddname                   }                       ]
[             [,directory-quantity])  {,stepname.ddname         }    )                  ]
[                                     {stepname.procstep.ddname }                       ]
```

• Figure 5. The DD Statement (Part 1 of 2)

```
              Positional Subparameters

  VOLUME=([PRIVATE],[RETAIN],[volume sequence number],[volume count]

              Keyword Subparameters

  ,SER=(volume-serial-number[volume-serial-number]9...)

             dsname                                       )
  ,REF=      *.ddname
             *.stepname.ddname
             *.stepname.procstep.ddname

                                                  NL
  LABEL=([data-set-sequence-number],              SL
                                                  NSL

  ,EXPDT=yyddd    )
  ,RETPD=xxxx

            NEW          ,DELETE
            OLD          ,KEEP
  DISP=(    SHR          ,PASS            )
            MOD          ,CATLG
                         ,UNCATLG

  SYSOUT=classname
  SYSOUT=(x[,program-name][,form-no.])
```

[1]The name field must be blank when concatenating data sets.

[2]All parameters are optional to allow a programmer flexibility in the use of the DD statement; however, a DD statement with a blank operand field is meaningless.

[3]If the positional parameter is specified, keyword parameters cannot be specified.

[4]If subparameter-list consists of only one subparameter and no leading comma (indicating the omission of a positional subparameter) is required, the delimiting parentheses may be omitted.

[5]If subparameter-list is omitted, the entire parameter must be omitted.

[6]See User Defined Files for the applicable subparameters.

[7]See the publication IBM System/360 Operating System: Job Control Language, Form C28-6539.

[8]If only name is specified, the delimiting parentheses may be omitted.

[9]If only one volume-serial-number is specified, the delimiting parentheses may be omitted.

[10]The SEP and AFF parameters should not be confused with the SEP and AFF subparameters of the UNIT parameter.

Figure 5. The DD Statement (Part 2 of 2)

Name Field

ddname
is used:

- to identify data sets defined by this DD statement to the compiler or linkage editor (see "Compiler Data Set Requirements" and "Linkage Editor Data Set Requirements"), or
- to relate the data sets defined in this DD statement to a file described in a COBOL source program (see "User-Defined Files"), and
- to identify this DD statement to other control statements in the input stream.

procstep.ddname
is used to alter or add DD statements in cataloged procedures. The step in the cataloged procedure is identified by procstep. The ddname identifies either:

- A DD statement in the cataloged procedure that is to be modified by the DD statement in the input stream, or
- A DD statement that is to be added to the DD statement in the procedure step.

Operand Field

*
indicates that data immediately follows this DD statement in the input stream. This parameter is used to specify a source deck or data in the input stream. If the EXEC statement specifies execution of a program, only one data set may be placed in the input stream. The end of the data set must be indicated by a delimiter statement. The data cannot contain // or /* in the first two characters of any record. The DD * statement must be the last DD statement of the job step. In MVT, for a step with a single input stream data set, DD * and a /* statement are not required. The system will supply both if missing. The default DDNAME will be SYSIN.

DATA
also indicates data in the input stream. The restrictions and use of the DATA parameter are the same as for the asterisk, except that // may appear in the first and second positions in the record, for example, when the data consists of control statements of a procedure that is to be cataloged.

DUMMY
allows the user's processing program to operate without performing input/output operations on the data set. The DUMMY parameter is valid only for data sets that are referred to by the basic sequential or queued sequential file processing techniques. If the DUMMY parameter is specified, a read request results in an end of data set exit giving unpredictable results. A write request is recognized, but no data is transmitted. No device allocation, external storage allocation, or cataloging takes place for dummy data sets.

DDNAME Parameter
defines a pseudo data set that will assume the characteristics of a real data set if a subsequent DD statement of the step is labeled with the specified ddname. When the DDNAME parameter is specified, it must be the first parameter in the operand. All other parameters are ignored and should be omitted when the DDNAME parameter appears (see "Cataloged Procedures").

DDNAME Subparameter:
ddname
names a DD statement that, if present, supplies the attributes of the data set. If it is not present, the statement is ignored.

DSNAME Parameter
allows the programmer to specify the name of the data set to be created or to refer to a previously created data set. Various types of names can be specified as follows (see "Using the DD Statement" for a discussion of the various names):

- Fully qualified names: For data sets to be retrieved from or stored in the system catalog.
- Generation data group names: For an entire generation data group, or any single generation thereof.
- Simple names: For data sets that are not cataloged.
- Reference names: For data sets whose names are given in the DSNAME parameter of another DD statement in the same job.

- Temporary names: For temporary data sets that are to be named for the duration of one job only.

If the DSNAME parameter is omitted, the operating system assigns a unique name to the data set. (This parameter should be supplied for all except temporary data sets.)

DSNAME Subparameters:

dsname
: specifies the fully qualified name of a data set. This is the name under which the data set can be cataloged or otherwise identified on the volume.

dsname(element)
: specifies a particular generation of a generation data group, a member of a partitioned data set, or an area of an indexed sequential data set. To indicate a generation of a generation data group, the element is a zero or a signed integer. To indicate a member of a partitioned data set, the element is a name. To indicate an area of an indexed sequential data set, the element is PRIME, OVFLOW, or INDEX. Refer to the section "Using the DD Statement" for information about generation data groups and examples of partitioned data sets.

*.ddname
: indicates that the DSNAME parameter (only) is to be copied from a preceding DD statement in the current job step.

*.stepname.ddname
: indicates that the DSNAME parameter (only) is to be copied from the DD statement, ddname, that occurred in a previous step, stepname, in the current job. If this form of the subparameter appears in a DD statement of a cataloged procedure, stepname refers to a previous step of the procedure, or, if no such step is found, to a previous step of the current job.

*.stepname.procstep.ddname
: indicates that the DSNAME parameter (only) is to be copied from a DD statement in a cataloged procedure. The EXEC statement that called for execution of the procedure, as well as the step and DD statement of the procedure, must be identified.

&name
: allows the programmer to supply a temporary name for a data set that is to be deleted by the end of the job. The operating system substitutes a unique symbol for this subparameter. The programmer can use the temporary name in other steps to refer to the data set. The same symbol is substituted for each recurrence of this name within the job. Upon completion of the job, the name is dissociated from the data set. The same temporary name can be used in other jobs without ambiguity.

&name(element)
: allows the programmer to supply a name for a member of a temporary partitioned data set that will be deleted at the end of the step.

DCB Parameter
: allows the programmer to specify at execution time, rather than at the time of compilation, information for completing the data control block associated with the data set. For further information about the data control block and DCB subparameters see "Execution Time Data Set Requirements" and "Additional File Processing Information."

  The first subparameter of this parameter may be used to copy DCB attributes from the data set label of a cataloged data set or from a preceding DD statement.

SEP and AFF Parameters
: allow the programmer to optimize the use of channels among groups of data sets. SEP indicates channel separation and AFF indicates channel affinity.

  If neither parameter is supplied, any available channel, consistent with the UNIT parameter requirement, is assigned. The affinity parameter groups two or more data sets so that they can be separated from another data set requesting channel separation. For indexed sequential data sets these parameters are written as for any data set. They can be used in succeeding DD statements to refer to the first DD statement defining an indexed sequential data set. However, the second and third DD statements cannot request separation from or affinity to one another because they are unnamed. Thus, to establish channel separation and affinity for all of the areas, the name subparameter of the UNIT parameter must be used to request specific devices on specific channels.

UNIT Parameter
: specifies the quantity and types of input/output devices to be allocated for use by the data set.

If the UNIT parameter is not specified in the current DD statement, there are several ways in which the unit information may be inferred by the system:

- If the current data set has already been created and it is either being passed to the current step, or if it has been cataloged, any unit name specified in this DD statement is ignored.

- If the REF subparameter of the VOLUME parameter is specified, the current data set is given affinity with the data set referred to; that data set's defining DD statement provides the unit information.

- If the current data set is to operate in the split cylinder mode with a previously defined data set, it will reside on the unit specified in the DD statement for the previous data set.

- If the current data set is to use space suballocated from that assigned to a previously defined data set, it will reside on the same unit as the data set from which the space is obtained.

- If the current data set is assigned to the standard output class (SYSOUT is specified), it is written on the unit specified by the operator for class A.

If the current data set is in the input stream (defined by a DD * or DD DATA statement), the DD statement defining the data set should not contain a UNIT parameter.

If this parameter specifies a direct-access device for a data set being created, it is also necessary to reserve the space the data set will occupy, using another parameter of the DD statement. Depending on the way in which the space will be used, the SPACE, SPLIT, or SUBALLOC parameter can be specified. These parameters are discussed under individual headings.

If the UNIT parameter specifies a tape device, no SPACE, SPLIT, or SUBALLOC parameters are required.

The UNIT parameter must be specified if VOLUME=SER is specified in the DD statement.

UNIT Subparameters:

name
: specifies the name of an input/output device, a single cell within a data cell drive, a device class name, or any meaningful combination of input/output devices specified by an installation. (Direct-access devices and magnetic tape devices can be combined. No other device type combination is allowed.) Names and device classes are defined at system generation time. The device class names that are required for IBM cataloged procedures and are normally used by most installations are shown in Figure 6. These names can be specified by the installation at system generation time.

The block size specified in the source program (in the BLOCK CONTAINS clause or in the record description) must not exceed the maximum block size permitted for the device. For example, the maximum block size for the IBM 2311 is 3,625 characters, and the maximum block size for the IBM 2400 series is 32,760 characters.

Note: When device-independence is specified by means of the ASSIGN TO...UTILITY statement in the Environment Division, the device chosen by the system will be dependent on the DD statement. Therefore, if the user's installation has both an IBM 2311 and an IBM 2302 that may be used as utility devices, the user should write BLOCK CONTAINS 3625 CHARACTERS (or any number smaller than 3625) to ensure that the block can be contained on one track.

Class Name	Class Functions	Device Type
SYSSQ	writing reading	direct-access magnetic tape
SYSDA	writing reading	direct-access

Figure 6. Device Class Names Required for IBM-Supplied Cataloged Procedures

n
: specifies the number of devices to be allocated to the data set. If this parameter is omitted, 1 is assumed.

P
: specifies parallel mount.

DEFER
: indicates deferred mounting. Deferred mounting cannot be specified for a new output data set on a direct-access device or for an indexed sequential data set.

SEP=(list of ddnames)
specifies unit separation.

AFF=ddname
specifies unit affinity.

SPACE Parameter
specifies space to be allocated in a direct-access volume. Although SPACE has no meaning for tape volumes, if a data set is assigned to a device class that contains both direct-access devices and tape devices, SPACE should be specified.

Two forms of the SPACE parameter may be used, with or without absolute track address. ABSTR requests that allocation begin at a specific address.

SPACE Subparameters:
{ABSTR / TRK / CYL / average-record-length}
specifies the unit of measurement in which storage is to be assigned. The units may be tracks (ABSTR,TRK), cylinders (CYL), or records (average-record-length expressed in decimal numbers). In addition, ABSTR indicates that the allocated space is to begin at a specific track address. If the specified tracks are already allocated to another data set, they will not be reallocated to this data set.

Note: For indexed sequential data sets, only the CYL or ABSTR subparameter is permitted. Neither the TRK subparameter nor the average record length can be specified. When an indexed sequential data set is defined by more than one DD statement, all must specify either CYL or ABSTR; if some statements contain CYL and others ABSTR, the job will be abnormally terminated.

(Primary-quantity[,secondary-quantity] [,directory or index quantity])
specifies the amount of space to be allocated for the data set. The primary quantity indicates the number of records, tracks, or cylinders to be allocated when the job step begins. For indexed sequential data sets, this subparameter specifies the number of cylinders for the prime, overflow, or index area (see "Execution Time Data Set Requirements"). The secondary quantity indicates how much additional space is to be allocated each time previously allocated space is exhausted. This subparameter must not be specified when defining an indexed sequential data set. The directory quantity is used when initially creating a partitioned data set (PDS), and it specifies the number of 256-byte records to be reserved for the directory of the PDS. It can also specify the number of cylinders to be allocated for an index area imbedded within the prime area when a new indexed sequential data set is being defined (see the section "Special Applications of the Control Statements" in the publication IBM System/360 Operating System: Job Control Language, Form C28-6539).

Note: The directory contains the name and the relative position, within the data set, for each member of a partitioned data set. The name requires 8 bytes, the location 4 bytes. Up to 62 additional bytes can be used for additional information. For a directory of a partitioned data set that contains load modules, the minimum directory requirement for each member is 34 bytes.

RLSE
indicates that all unused external storage assigned to this data set is to be released when processing of the data set is completed.

{MXIG / ALX / CONTIG}
qualifies the request for the space to be allocated to the data set. MXIG requests the largest single block of storage that is greater than or equal to the space requested in the primary quantity. ALX requests the allocation of additional tracks in the volume. The operating system will allocate tracks in up to five blocks of storage, each block equal to or greater than the primary quantity. CONTIG requests that the space indicated in the primary quantity be contiguous.

If this subparameter is not specified, or if any option cannot be fulfilled, the operating system attempts to assign contiguous space. If there is not enough contiguous space, up to five noncontiguous areas are allocated.

ROUND
indicates that allocation of space for the specified number of records is to begin and end on a cylinder boundary. It can be used only when average record length is specified as the first subparameter.

quantity
specifies the number of tracks to be allocated. For an indexed sequential data set, this quantity must be equivalent to an integral number of cylinders; it specifies the space for the prime, overflow, or index area (see "Execution Time Data Set Requirements").

beginning address
specifies the relative number of the track desired, where the first track of a volume is defined as 0. (Track 0 cannot be requested.) The number is automatically converted to an address based on the particular device assigned. For an indexed sequential data set this number must indicate the beginning of a cylinder.

directory quantity
defines the number of 256-byte records to be allocated for the directory of a new partitioned data set. It also specifies the number of tracks to be allocated for an index area embedded within the prime area when a new indexed sequential data set is being defined. In the latter case, the number of tracks must be equivalent to an integral number of cylinders (see the section "Special Applications of the Control Statements" in the publication IBM System/360 Operating System: Job Control Language, Form C28-6539).

SPLIT Parameter
is specified when other data sets in the job step require space in the same direct-access volume, and the user wishes to minimize access-arm movement by sharing cylinders with the other data sets. The device is then said to be operating in a split cylinder mode. In this mode, two or more data sets are stored so that portions of each occupy tracks within every allocated cylinder.

Note: SPLIT should not be used when one of the data sets is an indexed sequential data set.

SPLIT Subparameters:

n
indicates the number of tracks per cylinder to be used for this data set if CYL is specified. If the average record length is specified, n is the percentage of the tracks per cylinder to be used for this data set.

{CYL / average-record-length}
indicates the units in which the space requirements are expressed in the next subparameter. The units may be cylinders (CYL) or physical records (in which case the average record length in bytes is specified as a decimal number not exceeding 65,535). If the average record length is given, and the data set is defined to have a key, the key length must be given in the DCB parameter of this DD statement.

primary-quantity
defines the number of cylinders or space for records to be allocated to the entire group of data sets.

secondary-quantity
defines the number of cylinders or space for records to be allocated each time the space allocated to any of the data sets in the group has been exhausted and more data is to be written. This quantity will not be split.

A group of data sets that share cylinders in the same device is defined by a sequence of DD statements. The first statement in the sequence must specify all parameters except secondary quantity, which is optional. Each of the statements that follow must specify only n, the amount of space required.

SUBALLOC Parameter
permits space to be obtained from another data set for which contiguous space was previously allocated. This enables data sets to be stored in a single volume. Space obtained through suballocation is removed from the original data set, and may not be further suballocated. The SUBALLOC parameter should not be used to obtain space for an indexed sequential data set.

Except for the subparameters described below the subparameters in the SUBALLOC parameter have the same meaning as those described in the SPACE parameter.

SUBALLOC Subparameters:

ddname
indicates that space is to be suballocated from the data set defined by the DD statement, ddname, that appears in the current step.

stepname.ddname
indicates that space is to be suballocated from the data set defined by the DD statement, ddname, occurring in a previous step, stepname. If this form of the subparameter appears in a DD statement in a cataloged procedure, stepname refers to a previous step of the procedure, or if no such step is found, to a previous step of the current job.

stepname.procstep.ddname
indicates that space is to be suballocated from a data set defined in a cataloged procedure. The first term identifies the step that called for execution of the procedure, the second identifies the procedure step, and the third identifies the DD statement which requested space originally.

VOLUME Parameter
specifies information about the volume(s) on which an input data set resides, or on which an output data set will reside. A volume can be a tape reel, a disk pack, part of a disk storage drive, or a data cell. Volumes can be used most efficiently if the programmer is familiar with the states a volume can assume. Volume states involve two criteria: the type of data set the programmer is defining and the manner in which the programmer requests a volume.

Data sets can be classified as one of two types, temporary or nontemporary. A temporary data set exists only for the duration of the job that creates it. A nontemporary data set can exist after the job is completed. The programmer indicates that a data set is temporary by coding:

- DSNAME=&name
- No DSNAME parameter
- DISP=(NEW,DELETE), either explicitly or implied, e.g., DISP=(,DELETE)
- DSNAME=reference, referring to a DD statement that defines a temporary data set.

All other data sets are considered nontemporary. If the programmer attempts to keep or catalog a passed data set that was declared temporary, the system changes the disposition to PASS unless DEFER was specified in the UNIT parameter. Such a data set is deleted at the end of the job.

The manner in which the programmer requests a volume can be considered specific or nonspecific. A specific reference is implied whenever a volume with a specific serial number is requested. Any one of the following conditions denotes a specific volume reference:

- The data set is cataloged or passed from an earlier job step.
- VOLUME=SER is coded in the DD statement.
- VOLUME=REF is coded in the DD statement, referring to an earlier specific volume reference.

All other types of volume references are nonspecific. (Nonspecific references can be made only for new data sets, in which case the system assigns a suitable volume.)

The state of a volume determines when the volume will be demounted and what kinds of data sets can be assigned to it.

Direct-Access Volumes: Direct-access volumes differ from tape volumes in that they can be shared by two or more data sets processed concurrently by more than one job. Because of this difference, direct-access volumes can assume different volume states than tape volumes. The volume state is determined by one characteristic from each of the following groups:

Mount Characteristics	Allocation Characteristics
Permanently Resident	Public
Reserved	Private
Removable	Storage

Permanently resident volumes are always mounted. The permanently resident characteristic applies automatically to:

- All physically nondemountable volumes, such as 2301 Drum Storage.
- The volume from which the system is loaded (the IPL volume).
- The volume containing the system data sets SYS1.LINKLIB, SYS1.PROCLIB, and SYS1.SYSJOBQE.
- Other volumes can be designated as permanently resident in a special member of SYS1.PROCLIB named PRESRES.

Permanently resident volumes are always public. The reserved characteristic applies to volumes that remain mounted until the operator issues an UNLOAD command. They are reserved by a MOUNT command referring to the unit on which they are mounted or by a PRESRES entry. The removable

characteristic applies to all volumes that are neither permanently resident nor reserved. Removable volumes do not have an allocation characteristic when they are not mounted. A reserved volume becomes removable after an UNLOAD command is issued for the unit on which it resides.

The allocation characteristics, public, private, and storage, deal with a volume's availability to be assigned by the system to temporary data sets, and, if the volume is removable, when it is to be demounted.

A public volume is used primarily for temporary data sets and, if it is permanently resident, for frequently used data sets. It must be requested by a specific volume reference if a data set is to be kept or cataloged on it. If a public volume is removable, it is demounted only when its unit is required by another volume. The programmer can change a removable/public volume to private by specifying VOLUME=PRIVATE.

A private volume must be requested by a specific volume reference. A new data set can be assigned to a private volume by specifying VOLUME=PRIVATE. If the volume is reserved, it remains mounted until the operator issues an UNLOAD command for the unit on which it resides. If it is removable, it will be demounted after it is used unless the programmer specifically requested that it be retained (VOLUME=RETAIN) or passed (DISP=PASS). Once a removable volume has been made private, it will ultimately be demounted. To use it as a public volume, it must be remounted.

A storage volume is used as an extension of main storage, to keep or catalog nontemporary data sets having non-specific volume requests. The programmer can assign the PRIVATE option to storage volumes.

Table 2 shows how direct-access volumes are assigned their mount and allocation characteristics.

Table 2. Direct-Access Volume States

Mount Characteristic	Allocation Characteristic: Public	Private	Storage
Permanently Resident	PRESRES or DEFAULT	PRESRES	PRESRES
Reserved	PRESRES or MOUNT command	PRESRES or MOUNT command	PRESRES or MOUNT command
Removable	Default	VOLUME=PRIVATE	N/A

Magnetic Tape Volumes: The volume state of a reel of magnetic tape is also determined by a combination of mount and allocation characteristics:

Mount Characteristics	Allocation Characteristics
Reserved	Private
Removable	Scratch

The reserved/scratch combination is not a valid volume state. Reserved tape volumes assume their state when the operator issues a MOUNT command for the unit on which they reside. They remain mounted until the operator issues a corresponding UNLOAD command. Reserved tapes must be requested by a specific volume reference.

A removable tape volume is assigned the private characteristic when one of the following occurs:

- It is requested with a specific volume reference.
- It is requested for allocation to a nontemporary data set.
- The VOLUME parameter is coded with the PRIVATE option.

A removable/private volume is demounted after its last use in the job step, unless the programmer requests that it be retained.

All other tape volumes are assigned the removable/scratch state. They remain mounted until their unit is required by another volume.

Volume Parameter Facilities: The facilities of the VOLUME parameter allow the programmer to:

- Request private volumes (PRIVATE)
- Request that private volumes remain mounted until the end of the job (RETAIN)
- Select particular volumes when the data set resides on more than one volume (seq#)
- Request more than one nonspecific volume (volcount)
- Identify specific volumes (SER and REF)

These facilities are all optional. The programmer can omit the VOLUME parameter when defining a new data set, in which case the system assigns a suitable public or scratch volume.

VOLUME Subparameters:

PRIVATE
indicates that the volume on which space is being allocated to the data set is to be made private. If the PRIVATE, SER, and REF subparameters are omitted for a new output data set, the system assigns the data set to any suitable public or scratch volume that is available.

RETAIN
indicates that this volume is to remain mounted after the job step is completed. Volumes are retained so that data may be transmitted to or from the data set, or so that other data sets may reside in the volume. If the data set requires more than one volume, only the last volume is retained; the other volumes will have been demounted previously. Another job step indicates when to demount the volume by omitting RETAIN. If each job step issues a RETAIN for the volume, the retained status lapses when execution of the job is completed.

volume-sequence-number
is a 1- to 4-digit number that specifies the sequence number of the first volume of the data set that is read or written. The volume sequence number is meaningful only if the data set is cataloged and earlier volumes are omitted.

volume-count
specifies the number of volumes required by the data set. Unless the SER or REF subparameter is used, this subparameter is required for every multi-volume output data set.

SER
specifies one or more serial numbers for the volumes required by the data sets. A volume serial number consists of one to six alphanumeric characters. If it contains fewer than six characters, the serial number is left adjusted and padded with blanks. If SER is not specified, and DISP is not specified as NEW, the data set is assumed to be cataloged, and serial numbers are retrieved from the catalog. A volume serial number is not required for new output data sets. Two volumes should not have the same serial number. When the SER parameter is included, the volume is treated as PRIVATE commencing with allocation for the current job step.

If this subparameter is specified, the UNIT parameter must also be specified.

REF
indicates that the data set is to occupy the same volume(s) as the data set identified by dsname *.ddname, *.stepname.ddname, or *.stepname.procstep.ddname. Table 3 shows the data set references.

When the data set resides in a tape volume and REF is specified, the data set is placed in the same volume, immediately behind the data set referred to by this subparameter. When the REF subparameter is used, the UNIT and LABEL parameters, if supplied, are ignored.

If SER or REF is not specified, the control program will allocate any nonprivate volume that is available.

Table 3. Data Set References

Option	Refers To
REF=dsname	a data set named dsname
REF=*.ddname	a data set indicated by DD statement ddname in the current job step
REF=*.stepname.ddname	a data set indicated by DD statement ddname in the job step stepname
REF=*.stepname.procstep.ddname	a data set indicated by DD statement ddname in the cataloged procedure step procstep called in the job step stepname. (See the chapter, "Cataloged Procedures.")

LABEL Parameter
specifies information about the label or labels associated with the data set. If a data set is passed from a previous job step, label information is retained from the DD statement that specified DISP=(,PASS). A LABEL parameter, if specified in the DD statement receiving the passed data set, is ignored. If the LABEL parameter is omitted and the data set is not being passed, standard labeling is assumed. The operating system verifies mounting when the label parameter specifies SL. Nonstandard labels can be specified only when installation-written routines to write and process nonstandard labels have been incorporated into the operating system. Refer to the publication IBM System/360 Operating System: System Programmer's Guide, Form C28-6550, for a description of how to write such routines. No label infor-

mation is required for direct-access devices.

LABEL Subparameters:

data-set-sequence-number
is a 4-digit number that identifies the relative location of the data set with respect to the first data set in a tape volume. (For example, if there are three data sets in a magnetic tape volume, the third data set is identified by data set sequence number 0003.) If the data set sequence number is not specified, the operating system assumes 0001. (This option should not be confused with the volume sequence number, which represents a particular volume for a data set.)

{NL / SL / NSL}
specifies the kind of label used for the data set. NL indicates no labels. SL indicates standard labels. NSL indicates nonstandard labels.

{EXPDT=yyddd / RETPD=xxxx}
specifies how long the data set shall exist. The expiration date, EXPDT=yyddd, indicates the year (yy) and the day (ddd) the data set can be deleted. The period of retention, RETPD=xxxx, indicates the period of time, in days, that the data set is to be retained. If neither is specified, the retention period is assumed to be zero.

DISP Parameter
describes the status of a data set and indicates what is to be done with it after its last use, or at the end of the job. The job scheduler executes the requested disposition functions at the completion of the associated job step. If the step is not executed because of an error found by the system before trying to initiate the step (e.g., an error in a job control language statement), the remaining statements are read and interpreted; however, none of the succeeding steps are executed, and the requested dispositions are not performed. This parameter can be omitted for data sets created and deleted during a single job step. Additional information about the relationship between the DISP parameter and the volume table of contents is contained in the chapter, "Additional File Processing Information."

DISP Subparameters:

NEW
indicates that the data set is being generated in this step. If the status is omitted, NEW is assumed.

OLD
indicates that the data set specified in the DSNAME parameter already exists.

SHR
has meaning only in a multiprogramming environment for existing data sets that reside on direct-access volumes. This subparameter indicates that the data set is part of a job in which operations do not prevent simultaneous use of the data set by another job. For a data set that is to be shared, the DD statement DISP parameter should be specified as DISP=SHR for every reference to the data set in a job. Unless this is done, the data set cannot be used by a concurrently operating job, and the job will have to wait until the particular file is free. DISP=SHR can also be specified under PCP (the primary control program) in which the system will assume an OLD disposition.

MOD
causes logical positioning after the last record in the data set. It indicates that the data set already exists and that it is to be added to, rather than read. If MOD is specified and (1) the volume serial number is not given, and (2) the data set is not cataloged or passed from an earlier job step, MOD is ignored and NEW is assumed. If the volume serial number is given, the data set is assumed to reside on the specified volume.

DELETE
causes the space occupied by the data set to be released for other purposes at the end of the current step. If the data set is cataloged, and the catalog is used to locate it, reference to the data set is removed from the catalog. If it is on a direct-access device, all references are removed from the volume table of contents, and the device space is made available for use by other data sets. If the data set is on tape, the volume in which the data set resides is then available for use by other data sets.

KEEP
ensures that the data set remains intact until a DELETE parameter is exercised in either the current job or some subsequent job. If the data set

is on a direct-access device, it remains tabulated in the volume table of contents after completion of the job. When the volume containing the data set is to be demounted, the operator is advised of the disposition.

PASS
indicates that the data set is to be referred to in a later step of the current job, at which time its disposition may be determined. When a subsequent reference to this data set is encountered, its PASS status lapses unless another PASS is issued. The final disposition of the data set should be specified in the last DD statement referring to the data set within the current job.

While a data set is in PASS status, the volume or volumes in which it resides are, in effect, retained; that is, the system will attempt to avoid demounting them. If demounting is necessary, the system will ensure proper remounting, through operator messages. The unit name specified on the DD statement in the receiving step must be consistent with the unit name in the passing step.

CATLG
causes the creation, at the end of the job step, of an index entry in the system catalog pointing to the data set. The data set can be referred to by name in subsequent jobs, without the need for volume serial number or device type information from the programmer. Cataloging also implies KEEP.

UNCATLG
causes the index entry that points to this data set to be removed from the index structure at the end of this step. The data set is not deleted. If it is on a direct-access volume, reference to it remains in the volume table of contents.

Note: The absence of DELETE, KEEP, PASS, CATLG, and UNCATLG indicates that no special action is to be taken to alter the permanent or temporary status of this data set. If the data set was created in this job, it will be deleted at the end of the current step. If the data set existed before this job, it will be kept.

SYSOUT Parameter
schedules a printing or punching operation for the data set described by the DD statement.

SYSOUT Subparameters:

classname
specifies the system output class on which the data set is to be written. A classname is an installation-specified 1-character name designating the output class to which the data set is to be written. Each classname is related to a particular output unit. Valid values for this parameter are A through Z and 0 through 9. A is the standard output class. Both data sets and system messages can be routed through the same output stream when using a priority scheduler. In this case, the output class selected for the data sets must be the same output class as that selected for the MSGCLASS parameter in the JOB statement. SYSOUT=B should be specified for any data set that is to be punched, under PCP, MFT, or MVT. The priority scheduler will route the output to class B. In an MVT environment, this will allow jobs to execute concurrently with the punching of output data.

Note: Classes 0 through 9 should not be used except in cases when the other classes are not sufficient. These classes are intended for future features of systems using priority schedulers.

(x[,program-name][,form-no.])
is used for priority scheduling systems only. When priority schedulers are used, the data set is written on an intermediate direct-access device during program execution, and later routed through an output stream to a system output device. The letter x can be an alphabetic or numeric character specifying the system output class. Output writers route data from the output classes to system output devices. The DD statement for this data set can also include a unit specification describing the intermediate direct-access device and an estimate of the space required. If there is a special installation program to handle output operations, its program-name should be specified. Program-name is the member name of the program, which must reside in the system library. If the output data set is to be printed or punched on a specific type of output form, a 4-digit form-no. should be specified. This form number is used to instruct the operator of the form to be used in a message issued at the time the data set is to be printed.

Note: If the program-name and form-no. are omitted, the delimiting parentheses can be omitted.

JOBLIB DD STATEMENT

The JOBLIB statement is a unique DD statement. It is used when one or more load modules to be executed in the job are members of a private library. When used, the JOBLIB statement must be placed after the JOB statement and before the first EXEC statement in the input stream. The ddname JOBLIB is specified in the name field of the statement.

If a job step other than the first is the one to execute the load module from the private library, the JOBLIB statement must specify the DISP parameter with PASS as the second subparameter. Only one JOBLIB statement can be specified for a job, but other libraries can be concatenated with the library named in the JOBLIB statement.

The JOBLIB statement concatenates the named library with the system library. It does not have to be specified for load modules created in the job, or for permanent members of the system library. (See "The Checklist for Job Control Procedures" for examples.)

DELIMITER STATEMENT

The delimiter statement (Figure 7) is used to separate data from subsequent control statements in the input stream, and is placed after each data set in the input stream. It is required only when job control statements and data appear together.

The delimiter statement is not required to mark the end of a data set defined with a DD * statement when using a system with MFT or MVT.

The delimiter statement contains a slash in column 1, an asterisk in column 2, and a blank in column 3. The remainder of the card may contain comments.

Name	Operation	Operand
/*		

Figure 7. Delimiter Statement

Nine data sets may be defined for a compilation job step; six of these (SYSUT1, SYSUT2, SYSUT3, SYSUT4, SYSIN, and SYSPRINT) are required. The other three data sets (SYSLIN, SYSPUNCH, and SYSLIB) are optional.

For compiler data sets other than utility data sets, a logical record size can be specified by using the LRECL and BLKSIZE subparameters of the DCB parameter. The values specified must be permissible for the device on which the data set resides. LRECL equals the logical record size, and BLKSIZE equals (n*LRECL) where n is equal to the blocking factor. If this information is not specified on the DD statement, it is assumed that the logical record sizes for the unblocked data sets have the following default values:

SYSIN, SYSLIN, SYSPUNCH, SYSLIB:
80 bytes each
SYSPRINT: 121 bytes

(See Appendix D for information about allocation of buffer space for these data sets.) The ddname that must be used in the DD statement describing the data set appears as the heading for each description that follows. Table 4 lists the function, device requirements, and allowable device classes for each data set. (See Appendix D for further information on blocked compiler data sets other than utility data sets.)

SYSUT1, SYSUT2, SYSUT3, SYSUT4

The DD statements using these ddnames define utility data sets that are used by the compiler when processing the source module. The data set defined by the SYSUT1 DD statement must be on a direct-access device. These data sets are temporary and have no connection with any other job step. For example, the DD statement

```
//SYSUT1 DD UNIT=SYSDA,SPACE=(TRK,(40,10))
```

specifies that the data set is to be written on any available direct-access device, with a primary allocation of 40 tracks. Additional tracks, if required, are to be allocated in groups of 10. The data set is to be deleted at the end of the job step (by default).

Note: When compiling large programs, if the primary space allocated for the data set is insufficient, an area of core storage may be used to complete processing. Then, when additional core storage is needed during compilation, enough contiguous space may not be available to load a compiler phase. Such a condition will result in an abnormal termination of the job. The programmer should attempt to allocate enough primary space to eliminate the need for a secondary allocation.

SYSIN

The data set defined by the SYSIN DD statement contains the input for the compiler, i.e., the source module statements that are to be processed. The input/output device assigned to this data set can be either the device transmitting the input stream (the device designated as SYSIN at system generation time) or a device designated by the programmer. When using a cataloged procedure, the DD statement describing this data set usually appears in the input stream. For example,

```
//SYSIN   DD   *
```

specifies that the input data set follows in the input stream. If the asterisk or DATA convention is used, the SYSIN DD statement must be the last DD statement in the job step.

SYSPRINT

This data set is used by the compiler to produce a listing. Output may be directed either to a printer, to a direct-access device, or to a magnetic-tape device. The listing will include the results of either the default or the specified options of the PARM parameter (i.e., diagnostic messages, the object code listing). For example, in the DD statement

```
//SYSPRINT  DD  SYSOUT=A
```

SYSOUT is the disposition for printer data sets, and A is the standard output class for printer data sets.

Note: If SYSOUT=A is not specified for SYSPRINT, MOD must be specified in the DISP parameter.

Table 4. Data Sets Used for Compilation

ddname	Type	Function	Device Requirements	Allowable Device Classes
SYSIN (required)	INPUT/OUTPUT	Reading the source program	card reader intermediate storage	SYSSQ or the input stream device (specified by DD * or DD DATA)
SYSPRINT (required)		Writing the storage map, listings, and messages	printer intermediate storage	SYSSQ, SYSDA, standard output class A
SYSPUNCH (optional)		Punching the object module deck	card punch direct-access magnetic tape	SYSCP, SYSSQ, SYSDA, output class B
SYSLIN (optional)		Creating an object module data set as output from the compiler and input to the linkage editor	direct-access magnetic tape	SYSSQ, SYSDA
SYSUT1 (required)	UTILITY	Work data set needed by the compiler during compilation	direct-access	SYSDA
SYSUT2 (required)		Work data set needed by the compiler during compilation	direct-access magnetic tape	SYSSQ, SYSDA
SYSUT3 (required)		Work data set needed by the compiler during compilation	direct-access magnetic tape	SYSSQ, SYSDA
SYSUT4 (required)		Work data set needed by the compiler during compilation	direct-access magnetic tape	SYSSQ, SYSDA
SYSLIB (optional)	LIBRARY	Optional user source program library	direct-access	SYSDA

SYSPUNCH

The device defined by the SYSPUNCH DD statement is used to punch an object module deck. This data set can be directed to a card punch, direct-access device, or magnetic tape. For example, in the DD statement

```
//SYSPUNCH  DD  SYSOUT=B,UNIT=SYSCP
```

SYSCP is the device class name for data sets that are to be punched.

Note: If SYSPUNCH is defined as a partitioned data set (PDS), then it must previously have been defined, and DISP=OLD must be specified.

SYSLIN

The device defined by the SYSLIN DD statement is used by the compiler to store an object module. It may be on a direct-access or magnetic-tape device. For example:

```
//SYSLIN DD  DSNAME=&GOFILE,                X
//             DISP=(MOD,PASS),             X
//             UNIT=SYSDA,                  X
//             SPACE=(TRK,(30,10))
```

The temporary name of the data set is GOFILE; the parameter DISP=(MOD,PASS) indicates that the data is to be created or added to in this job step and is to be passed to another job step, which may be the linkage editor step. The device to be assigned for storage is a direct-access device on which 30 tracks are initially allocated to the data set. If more space is n eeded, tracks are allocated 10 at a time.

Note: If SYSLIN is defined as a partitioned data set (PDS), then it must previously have been defined, and DISP=OLD must be specified.

SYSLIB

The SYSLIB DD statement defines the library (PDS) that contains the data requested by an INCLUDE or COPY statement (in the source module) or by a BASIS card in the input stream. Note that more than

one partitioned data set may be used for the library function by concatenating them with SYSLIB (see the chapter entitled "Libraries" for an example). Libraries must always be on direct-access devices. Only one SYSLIB statement may be used in a compilation job step. For example, in the DD statement

```
//SYSLIB  DD  DSNAME=USERLIB,DISP=OLD
```

the name of the library is USERLIB, and DISP=OLD indicates that the library has been created in a previous job. No other information need be given if the specified library has been cataloged.

Four data sets are required for linkage editor processing. Others may be necessary if secondary input is specified. In the following discussions the ddname that must be used in the DD statement describing the data set appears as the heading for each description of the particular data set. For any user-defined data set, the ddname is defined by the programmer. Table 5 lists the function, device requirements, and allowable device classes for each data set.

SYSLIN

The SYSLIN DD statement defines the data set that is primary input to linkage editor processing. Normally this data set consists of the output from a previous compilation job step. The primary input may also be linkage editor control statements, such as the INCLUDE, LIBRARY, or OVERLAY statements, but not job control statements (see "Calling and Called Programs"). The input device assigned to this data set is either the device transmitting the input stream, if the input is an object module deck, or a device designated by the programmer. However, the data set may simply be passed from the previous compilation job step. For example, in the DD statement

```
//SYSLIN DD  DSNAME=*.STEPNAME.SYSLIN,     X
//             DISP=(OLD,DELETE)
```

the data set is defined in the SYSLIN DD statement contained in the compiler job step, STEPNAME. DISP=(OLD,DELETE) indicates that the data set was created in a previous job step and is to be deleted at the end of this job step.

SYSPRINT

The data set defined by the SYSPRINT DD statement is used by the linkage editor to produce a listing. Output may be directed to a printer or to magnetic tape. The listing may include any options specified by the PARM parameter of the EXEC statement (a module map or cross reference list, diagnostic or informative messages, etc.). For example:

```
//SYSPRINT  DD  SYSOUT=A
```

Table 5. Data Sets Used For Linkage Editing

ddname	Type	Function	Device Requirements	Allowable Device Classes
SYSLIN (required)	INPUT/ OUTPUT	Primary input data, normally the output of the compiler	direct-access magnetic tape card reader	SYSSQ, SYSDA, or the input stream device (specified by DD * or DD DATA)
SYSPRINT (required)		Diagnostic messages informative messages module map cross reference list	printer intermediate storage device	SYSSQ, standard output class A
SYSLMOD (required)		Output data set for the load module	direct-access	SYSDA
SYSUT1 (required)	UTILITY	work data set	direct-access	SYSDA
SYSLIB required for COBOL library subroutines	LIBRARY	Automatic call library (SYS1.COBLIB is the name of the COBOL subroutine library)	direct-access	SYSDA
user-specified (optional)		Additional object modules and load modules	direct-access magnetic tape	SYSDA, SYSSQ

SYSLMOD

The SYSLMOD DD statement defines the output data set, in this case the load module. The load module must be placed in a library as a named member. The library can be the Link Library (SYS1.LINKLIB) or a private user-defined library. Such libraries must always reside on a direct-access device, and space for the library is allocated when the library is created. For example, in the DD statement

```
//SYSLMOD DD DSNAME=SYS1.LINKLIB(MEMBER),X
//           DISP=MOD
```

the load module, MEMBER, is stored as a member of the link library. DISP=MOD indicates that the library is already created and is to be added to.

```
//SYSLMOD  DD  DSNAME=LIB1(BALANCE),      X
//             DISP=(NEW,CATLG),          X
//             VOLUME=SER=11111,          X
//             SPACE=(TRK,(40,10,1)),     X
//             UNIT=SYSDA
```

The load module, BALANCE, is to be a member of a library, LIB1, which is to be created in this job step, with BALANCE as its first member. The direct-access volume to which it is directed is identified by the serial number, 11111. A primary quantity of 40 tracks is allocated to the library with an additional allocation for one 256-byte record to be used for the directory. If more space is needed for the library, tracks are added, 10 at a time. (However, no additional space can be allocated for the directory.)

Note: If the load module is placed in a private library, the JOBLIB DD statement must be specified in subsequent jobs that execute load modules from the library.

SYSUT1

The SYSUT1 DD statement defines a utility data set used by the linkage editor when processing object modules and load modules. The data set must be on a direct-access device. It is a temporary data set and has no connection with any other job step. For example:

```
//SYSUT1 DD UNIT=SYSDA,SPACE=(TRK,(40,10))
```

The data set is initially allocated 40 tracks on any available direct-access device. If more space is needed, tracks are added, 10 at a time. A temporary name is assigned to the data set for the job step.

SYSLIB

The SYSLIB DD statement assigns the named partitioned data set to the automatic call library from which modules may be automatically obtained by the linkage editor to resolve external references.

```
//SYSLIB  DD  DSNAME=SYS1.COBLIB,DISP=SHR
```

This statement assigns the COBOL subroutine library to the automatic call library. It must be specified if there is a possibility that the compiler may have generated calls to any COBOL library subroutines. See Appendix B for a list of library subroutines, their functions and entry points.

USER-SPECIFIED DATA SETS

Additional data sets may be defined for linkage editor processing. These data sets may be used as additional input sources of object modules or load modules. They may also be concatenated with the primary input data set or the automatic call library (see "Libraries").

Any number of data sets may be used for execution time processing. These data sets, or files, are identified in the source program, and each must be described by a DD statement. The ddname is used to link the DD statement to the COBOL statement in the source program that specifies the ddname. DD statement requirements for the DISPLAY, ACCEPT, EXHIBIT, and TRACE statements and for user-defined files are discussed in the following text. A DD statement that specifies an abnormal termination dump is also discussed. Use of the Sort Feature requires additional DD statements. For information about these statements, refer to the chapter "Using the Sort Feature."

THE DISPLAY STATEMENT

The DISPLAY statement requires an associated DD statement unless the data is to be displayed on the console. The DD statements needed for each form of the statement are as follows:

Example 1:

```
          {data-name}
DISPLAY   {literal  }  ...UPON SYSPUNCH
```

```
//SYSPUNCH DD applicable parameters, i.e.,
```

```
//SYSPUNCH DD SYSOUT=B
```

An unblocked data set and a logical record length of 80 characters are assumed. However, the programmer can specify a blocked data set by using the subparameters of the DCB parameter as follows: RECFM=FB, BLKSIZE=n*80, where n is the blocking factor. SYSPUNCH must be on a device where blocking is permitted. For example:

```
//SYSPUNCH    DD   UNIT=SYSSQ,DCB=          X
//                 (RECFM=FB,BLKSIZE=160),  X
//                  LABEL=(,NL)
```

Example 2:

```
          {data-name}
DISPLAY   {literal  }  ...
```

(SYSOUT is the default option)

```
//SYSOUT DD applicable parameters, i.e.,
```

```
//SYSOUT DD SYSOUT=A
```

An unblocked data set and a line width of 120 characters are assumed. However, the programmer can specify an alternate line width and/or a blocked data set by using the DCB parameter. To specify an alternate line width, the subparameters of the DCB parameter are used as follows:

LRECL=line width+1,BLKSIZE=LRECL value.

To specify a blocked data set, the subparameters are used as follows:

RECFM=FBA,LRECL=line width+1,
BLKSIZE=n*(LRECL value)

where n is a blocking factor. SYSOUT must be on a device where blocking is permitted. The extra character in LRECL allows for the carriage control character. For example:

To specify an alternate line width:

```
//SYSOUT      DD  SYSOUT=A,DCB=(LRECL=133,X
//                BLKSIZE=133)
```

To specify a blocked data set:

```
//SYSOUT      DD   DSNAME=PRINTOUT,        X
//                 UNIT=SYSDA,...,         X
//                 DCB=(RECFM=FBA,         X
//                 LRECL=121,              X
//                 BLKSIZE=605),           X
//                 VOLUME=SER=111111
```

Example 3:

```
          {data-name}
DISPLAY   {literal  }... UPON mnemonic-name
```

where mnemonic-name is associated with the words SYSPUNCH or SYSOUT in the Environment Division.

```
  {SYSPUNCH}
//{SYSOUT  }  DD   applicable parameters
```

THE ACCEPT STATEMENT

The ACCEPT statement requires an associated DD statement unless the data is being accepted from the console. The DD statements for each form of the statement are as follows:

Example 1:

```
     ACCEPT data-name
```

(SYSIN is the default option)

```
//SYSIN    DD    applicable parameters
```

Example 2:

```
     ACCEPT data-name FROM mnemonic-name
```

where mnemonic-name is associated with the word SYSIN in the Environment Division.

```
//SYSIN DD applicable parameters, i.e.,

//SYSIN DD   *
        (data)
```

An unblocked data set and a logical record length of 80 characters are assumed. However, the programmer can specify a blocked data set by using the subparameters of the DCB parameter as follows: RECFM=FB, BLKSIZE=n*80, where n is the blocking factor. SYSIN must be on a device where blocking is permitted. For example:

```
//SYSIN       DD   UNIT=2400,...,          X
//                 DCB=(RECFM=FB,          X
//                 BLKSIZE=160),           X
//                 LABEL=(,NL)
```

If a logical record length of other than 80 characters is desired, it must be specified in the LRECL field of the DCB parameter.

THE EXHIBIT OR TRACE STATEMENT

The EXHIBIT or TRACE statement requires a SYSOUT DD statement as discussed for DISPLAY.

Note: If the job step already includes a SYSOUT DD statement for some other use, another may not be inserted.

ABNORMAL TERMINATION DUMP

To obtain a full abnormal termination dump in case the job is abnormally terminated, the following DD statement must be used:

```
//SYSABEND DD applicable parameters, i.e.,

//SYSABEND DD SYSOUT=A
```

USER-DEFINED FILES

Files that are processed in a COBOL program must be described as data sets to the operating system. Whenever a file is specified in a program by a

```
SELECT file-name ASSIGN TO external-name...
```

this file must be described in an FD file-name entry and in a DD statement in the execution-time job step. The ddname in the DD statement is the external-name specified in the ASSIGN TO clause.

Note: The device-number clause of the SELECT statement is treated as comments by the compiler if an actual device number is specified. Actual device allocation is a function of the DD statement.

FILE NAMES AND DATA SET NAMES

In general, the term file, as used in a COBOL program, and the term data set, as used in operating system terminology, have the same meaning. There may be a difference, however, between the file-name and the data set name. The data set name always represents a specific data set. The file-name can, at different times, represent different data sets. The DD statement allows a programmer to select, at the time his program is executed, the specific data set that is to be associated with a particular file-name. This facility can be especially powerful when applied to input data sets.

The file-name is a name known within the COBOL program. Changing a file-name requires changing input/output statements and recompiling the program. Changing a DD statement when a program is executed is a simple procedure.

As an example, consider a COBOL program that might be used in exactly the same way for several different master files. It might contain the clause SELECT MASTER

ASSIGN TO 'MASTERA'.... In that case, the following DD statements, used at different times, would assign the different named data sets to the program:

```
//MASTERA   DD  DSNAME=MASTER1,...
//MASTERA   DD  DSNAME=MASTER2,...
//MASTERA   DD  DSNAME=MASTER3,...
```

If the first DD statement appears in the job step that calls for execution of the program, any reference within the program to MASTER is a reference to the data set named MASTER1; if the second DD statement appears, the reference is to MASTER2; if the third, the reference is to MASTER3.

However, if a file-name within a program is always to be applicable to only a single data set, the names might be written as follows:

```
SELECT TAXRATE ASSIGN TO 'TAXRATE'...
```

The applicable DD statement might be:

```
//TAXRATE   DD   DSNAME=TAXRATE,...
```

Of the names, only the external-name that appears in the ASSIGN TO clause and the ddname of the DD statement must always be the same. The file-name and the data set name may be the same, or they may be different.

SPECIFYING INFORMATION ABOUT A FILE

Some of the information about the file must always be specified in the FD entry. Other information must be specified in the DD statement. For example, the amount of space allocated for a direct-access output file must be specified in the DD statement by the SPACE, SPLIT, or SUBALLOC parameters.

Certain characteristics of files cannot be expressed in the COBOL language, and may be specified in the DD statement for the file by the DCB parameter. This parameter allows the programmer to specify information for completing the data control block associated with the file. (See "Additional File Processing Information" for a discussion of the data control block, and Appendix C for information about the fields of the data control block).

Each file that is named by the SELECT clause requires the use of a file processing technique. Five processing techniques are discussed in this publication. They are the queued sequential (QSAM), basic sequential (BSAM), basic direct (BDAM), queued indexed sequential (QISAM), and basic indexed sequential (BISAM).

THE FILE PROCESSING TECHNIQUES

The file processing technique to be used is determined by the processing that is to be done on the file, the data organization of the file, and whether the data is accessed randomly or sequentially. The ORGANIZATION and ACCESS clauses of a COBOL source program determine the technique to be used. The information to be specified in the DD statement and in the FD entry depends on the technique used. Figure 8 shows the permissible combinations of processing, data access, and organization; clauses in the source program that specify them; and the technique that is used for each combination. In addition, the record formats and characteristics for each technique are listed. For additional information about the capabilities and restrictions for each file processing technique, refer to the publication IBM System/360 Operating System: Supervisor and Data Management Services, Form C28-6646.

Files can be created only by use of QSAM, BSAM and QISAM. Files having fixed type records can be created with each of these access methods. Files having variable type records and undefined records may be created using QSAM or BSAM.

In the following sections, the required and optional COBOL clauses and DD statement parameters for each file processing technique are shown in figures. Included are discussions of special considerations for each technique.

PROCESSING WITH QSAM

Figure 9 shows the COBOL clauses that may be used with QSAM. Special considerations are as follows:

1. The RESERVE clause can be used to specify more buffer areas, allowing overlap of input/output operations with the processing of data. If this clause is not used, additional buffers may be specified by using the BUFNO option in the DD statement. If no additional buffer areas are specified, two buffers are reserved by the system.

File Processing Action	Organization and Access Clauses that Define the Processing	Permissible Record Formats		Device Requirements	File Processing Technique
		Blocked	Unblocked		
Write, read, and update standard sequential file	ORGANIZATION clause is omitted. ACCESS SEQUENTIAL or ACCESS clause is omitted	F,V	F,V,U	Direct Access Magnetic Tape Unit Record	QSAM
Write and read a direct-access file with relative record addressing	ORGANIZATION RELATIVE ACCESS SEQUENTIAL		F	Direct Access	BSAM
Read and update a direct-access file with relative record addressing	ORGANIZATION RELATIVE ACCESS RANDOM		F	Direct Access	BDAM
Write and read a direct-access file with relative track addressing	ORGANIZATION DIRECT ACCESS SEQUENTIAL		F,U,V	Direct Access	BSAM
Read, update, and insert into a direct-access file with relative track addressing	ORGANIZATION DIRECT ACCESS RANDOM		F,U,V	Direct Access	BDAM
Create a direct-access file with indexed sequential organization	ORGANIZATION INDEXED ACCESS SEQUENTIAL or ACCESS cluase omitted	F	F	Direct Access	QISAM
Read and update a direct-access file with indexed sequential organization	ORGANIZATION INDEXED ACCESS SEQUENTIAL or ACCESS clause omitted	F	F	Direct Access	QISAM
Read, update, and insert into a direct-access file with indexed sequential organization	ORGANIZATION INDEXED ACCESS RANDOM	F	F	Direct Access	BISAM

Figure 8. Determining the File Processing Technique

2. If WRITE AFTER ADVANCING is used, record size specified in the FD entry must allow for the carriage control character, but the character will not be printed or punched. For example, if the record size specified in the FD entry is 121, the actual record is 121 characters; however, only 120 characters will be printed or punched.

Note: If the immediate destination of the record is a device that does not recognize a carriage control character, the system assumes that the control character is the first character of the data. If WRITE AFTER ADVANCING is not used, the control character is treated as the first byte of data by the punch or printer.

3. If the input device is the card reader, RECORDING MODE IS F should be specified. If this clause is not specified or RECORDING MODE IS V is specified, the first eight bytes of the record will be interpreted as the control bytes required for files with format V records.

4. If QSAM files are on magnetic tape, the record block size should be at least 18 bytes. Shorter blocks will be treated as noise records.

Figures 10 and 11 show the parameters in the DD statement that may be used with QSAM. All parameters except the DCB are described in "Format of the Job Control Statements." Additional DCB subparameters not shown in these figures are required for use of the sort feature. See "Using the Sort Feature" for information on these parameters.

The DCB subparameters that can be specified in the DD statement for QSAM files are as follows:

```
DCB=[DEN={0|1|2|3}]
    [,TRTCH={C|E|T|ET}]
    [,PRTSP={0|1|2|3}]
    [,MODE={C|E}]
    [,STACK={1|2}]
    [,OPTCD={W|C|WC}]
    [,BLKSIZE=integer]
    [,BUFNO=integer]
    [,EROPT={ACC|SKP|ABE}]
```

DEN={0|1|2|3}
can be used with magnetic tape, and specifies a value for the tape recording density in bits per inch as listed in Table 6. If no value is specified, 200 bits-per-inch is assumed for 7-track tape, 800 bpi for 9-track tape without dual density and 1600 bpi for 9-track tape with dual density.

Table 6. DEN Values

	Tape Recording Density Bits-per-Inch	
	Model 2400	
DEN Value	7 TRACK	9 TRACK
0	200	--
1	556	--
2	800	800
3	--	1600

TRTCH={C|E|T|ET}
is used with 7-track tape to specify the tape recording technique, as follows:

C - Specifies that the data-conversion feature is to be used; if data conversion is not available, only format-F and format-U records are supported by the control program.
E - Specifies that even parity is to be used; if omitted, odd parity is assumed.
T - Specifies that BCD to EBCDIC translation is required.
ET- Specifies that even parity is to be used and BCD to EBCDIC translation is required.

PRTSP={0|1|2|3}
specifies the line spacing on a printer as 0, 1, 2, or 3. If PRTSP is not specified, 1 is assumed.

The PRTSP subparameter is valid only if the unit specified for the file is a printer. It is not valid if the file is a report file, nor is it valid if WRITE...AFTER ADVANCING is specified in the COBOL source program. Single spacing always is assumed for a printer unless other information is supplied.

MODE={C|E}
can be used with a card reader, a card punch or a card-read punch and specifies the mode of operation as follows:
C - The card image (column binary) mode.
E - The EBCDIC code.

If this information is not supplied by any source, E is assumed.

STACK={1|2}
can be used with a card reader, a card punch, or a card-read punch, and it specifies which stacker bin is to receive the card. Either 1 or 2 is specified. If this information is not supplied by any source, 1 is assumed.

OPTCD={W|C|WC}
requests an optional service provided by the system as follows:
W - To perform a write validity check (on direct-access devices only).
C - To process using the chained scheduling method (see the publication IBM System/360 Operating System: Supervisor and Data Management Services, Form C28-6646).
WC- To perform a validity check and use chained scheduling.

If this information is not supplied by any source, none of the services are provided, except in the case of the IBM 2321 direct-access device where OPTCD=W is specified by the operating system.

Type of OPEN Statement	Input/Output Statement	Required Key Clauses	Other Required Clauses	Optional Clauses
INPUT [REVERSED]	READ AT END	Not permitted	ASSIGN TO LABEL RECORDS DATA RECORDS CLOSE	BLOCK CONTAINS RECORD CONTAINS RERUN SAME AREA RESERVE REPORTS ARE USE (options 1,2) CLOSE [REEL] [UNIT]
OUTPUT	WRITE [AFTER ADVANCING]			
I-O	READ AT END REWRITE		ASSIGN TO DIRECT-ACCESS LABEL RECORDS DATA RECORDS CLOSE [UNIT]	

Figure 9. Permissible COBOL Clauses for QSAM

DSNAME	Device Type	UNIT	VOLUME	LABEL	SPACE	SUBALLOC	SPLIT	DISP	DCB: Device Dependent	DCB: General
AS	Direct-Access	AS	AS	SL	AS	AS	AS	{NEW / MOD} [,KEEP / ,PASS / ,CATLG / ,DELETE]	OPTCD=W, WC	OPTCD=C BLKSIZE BUFNO EROPT=ABE
	Magnetic Tape	AS	AS	SL NL NSL	NA	NA	NA		TRTCH, DEN	
	Unit Record	AS	NA	NL	NA	NA	NA	SYSOUT=A	PRTSP, MODE, STACK	EROPT=ACC (printer only) EROPT=ABE

AS = Applicable Subparameters
NA = Not Applicable

• Figure 10. DD Statement Parameters Applicable to QSAM Output Files

DSNAME	Device Type	UNIT	VOLUME	LABEL	SPACE	SUBALLOC	SPLIT	DISP	DCB: Device Dependent	DCB: General
AS	Direct-Access	not required if cataloged		SL	NA	NA	NA	OLD [,KEEP / ,PASS / ,CATLG / ,UNCATLG / ,DELETE]		OPTCD=C BLKSIZE BUFNO
	Magnetic Tape	not required if cataloged		SL NL NSL	NA	NA	NA		TRTCH, DEN	EROPT= ACC,SKP ABE
	Unit Record	Applicable Subparameters	NA	NA	NA	NA	NA		MODE, STACK	

AS = Applicable Subparameters
NA = Not Applicable

• Figure 11. DD Statement Parameters Applicable to QSAM Input and I-O Files

Note: If the validity check is specified, the system verifies that each record transferred from main storage to direct-access storage is written correctly. Standard recovery procedures are initiated if an error is detected.

BLKSIZE=integer
is used to specify the block size. This clause may be used only if BLOCK CONTAINS 0 RECORDS was specified at compile time.

BUFNO=number of buffers
is used to specify the number of buffers to be assigned to the file if no RESERVE clause is specified for the file in the source program. The maximum number is 255. However, the maximum number allowed for an installation is established at system generation time.

EROPT={ACC|SKP|ABE}
specifies the options to be executed if an error occurs in writing or reading a record as follows:
ACC - To accept the error block for processing.
SKP - To skip the error block.
ABE - To terminate the job.

There are two cases when the subparameter can be specified:

- If no error processing declarative (USE sentence, option 2) is specified.
- If an error processing declarative is specified with a normal return.

If no option is specified, ABE is assumed.

Processing with BSAM

Figure 12 shows the COBOL clauses that may be used when processing with BSAM. Special considerations are discussed below.

RELATIVE TRACK ADDRESSING: For files created by the use of BSAM with relative track addressing the compiler ensures that each track on which a record is written is completed as required by the operating system. (The DD statement specifies the amount of space to be allocated to the file.)

F-type Records: For files with F-type records, the compiler generates instructions to specify that dummy records be written to complete the track. These dummy records are written by the system and contain a configuration of all 1s in the first byte of the symbolic key field. They indicate to the system that a record can be added to the file in the space allocated to the dummy record. Since these records are system-generated, the programmer should not write records that include a symbolic key field containing such a configuration in the first byte of the field.

U-type or V-type Records: For files containing U-type or V-type records, capacity records are written to complete the track. A capacity record is a record which signifies that a track has been completed. The number of tracks thus filled will determine the space available for inserting records at a later time. Records are inserted, or added, by replacing dummy records or altering capacity records. Once the file is created, more space cannot be allocated; the extent of the file cannot be increased.

Note: For input files containing V-type records created by the use of BSAM with relative track addressing, capacity records are not made available.

The FILE-LIMIT Clause: If the FILE-LIMIT clause is specified, the number stated in this clause determines the number of tracks that are prepared with dummy or capacity records for the file. This clause may be used to ensure that sufficient space will be available for a file in which records will be inserted at a later time. If the FILE-LIMIT clause is not specified, the number of tracks used may be determined, at the end of file processing, from the dataname specified in the ACTUAL KEY clause. The number of tracks is the value plus 1.

Releasing Unused Tracks: Once a BSAM output file has been closed, records cannot be added to those tracks that were allocated but unformatted (prepared with dummy or capacity records). These unformatted tracks must be released by specifying the RLSE option in the SPACE parameter of the corresponding DD statement. This is done so the tracks will not be counted as part of the file.

Note: To release all of the unused tracks on the last volume, space must have been requested in track or block units (by the SPACE parameter). If the CYL subparameter was specified, the unused tracks of the last partially written cylinder are not released. If the file is then opened as a BDAM file with direct organization and with the APPLY RESTRICTED SEARCH option specified, these unused, unformatted tracks are searched and counted even though they do not contain any accessible records or space.

RELATIVE RECORD ADDRESSING: For files created by the use of BSAM with relative record addressing, the compiler generates instructions to add dummy records to complete the last track of the file when it is closed. These records are compiler-generated and contain a configuration of all 1s in the first byte of the record. Since these records are not system-generated, the user can write records with this configuration in the first byte of the record. In this manner, space can be reserved for later additions to the field.

KEY CLAUSES: For BSAM with relative record addressing, no KEY clause is used. For BSAM with relative track addressing, the SYMBOLIC KEY clause is always required; the ACTUAL KEY clause is required only for output files. The SYMBOLIC KEY clause specifies a data-name that contains the record identifier. The ACTUAL KEY clause specifies a data-name that contains the relative track number on which the record currently being written is placed. The value of the data-name advances sequentially as space on the current track is used up, or as the programmer adds to it. The first track of a BSAM file is: track number=ACTUAL KEY number=0. The second track number is 1. The SYMBOLIC KEY and ACTUAL KEY clauses must never be defined in any file or in the Linkage Section.

CHECKING FOR DUMMY RECORDS: If an input file containing F-type records was created by the use of BSAM with either relative track or relative record addressing, any dummy records in the file will also be made available. The programmer can recognize these records by testing for the presence of the figurative constant HIGH-VALUE in the first position of the record (relative record) or of the symbolic key field (relative track).

CREATING MULTI-VOLUME DATA SETS: The COBOL programmer may create multi-volume data sets for BSAM files with relative track or relative record addressing by means of the CLOSE REEL or UNIT statement. When this statement is executed, the current extent is initialized, volume-switching procedures are executed, and for relative track addressing, the contents of ACTUAL KEY are updated to reflect the relative track number of the last track on the old volume. The primary, and as many secondary allocations as possible, are normally given to the first volume (up to 16 extents per volume), and subsequent volumes are allocated from the secondary specification.

Since all records of a file created using relative record addressing must be created by the COBOL programmer, he may choose to permit volume switching to occur automatically. However, this procedure does not guarantee a specific distribution of records over the volumes, the placement of a particular record on a particular volume, or whether the data set is in fact multi-volume.

To create a file with records distributed as evenly as possible over several volumes, the programmer must calculate how much space his file will require (see Tables 7, 8, and 9) and divide by the number of volumes. The result of this calculation (rounded) should be specified as both the primary and secondary allocation of the SPACE parameter of the associated DD statement. The programmer should execute CLOSE REEL or UNIT before the end of the initially allocated space on the first volume (that is, execute the CLOSE REEL before writing the record which he wants to be first on the second volume).

For example, to distribute 2150 80-byte records as evenly as possible on three 2311 volumes, 34 tracks per volume are required and the SPACE parameter should specify (34,34). After writing the 714th record, the programmer should execute CLOSE REEL or UNIT and continue writing.

When using relative record addressing, if the programmer overestimates the required space and the records do not fill the last track(s), the compiler will write dummy records. These records are included in the relative record count and the programmer must be aware of this when trying to address records on subsequent volumes.

Table 7. Direct-Access Device Overhead Formulas

Device	Bytes Required by Each Data Block			
	Blocks With Keys		Blocks Without Keys	
	Bi	Bn	Bi	Bn
2311	81+1.049(KL+DL)	20+KL+DL	61+1.049(DL)	DL
2314	146+1.043(KL+DL)	45+KL+DL	101+1.043(DL)	DL
2302	81+1.049(KL+DL)	20+KL+DL	61+1.049(DL)	DL
2303	146+KL+DL	38+KL+DL	108+DL	DL
2301	186+KL+DL	53+KL+DL	133+DL	DL
2321	100+1.049(KL+DL)	16+KL+DL	84+1.049(DL)	DL

Bi is any block but the last on the track.
Bn is the last block on the track.
DL is data length.
KL is key length.

Table 8. Direct-Access Storage Device Capacities

Device Type	Volume Type	Track Capacity*	Tracks per Cylinder	No. of Cylinders	Total Capacity*
2311	Disk	3625	10	200	7,250,000
2314	Disk	7294	20	200	29,176,000
2302	Disk	4984	46	246	56,398,944
2303	Drum	4892	10	80	3,913,600
2301	Drum	20483	8	25**	4,096,600
2321	Cell	2000	20***	980***	39,200,000

*Capacity indicated in bytes.
**There are 25 logical cylinders in a 2301 Drum.
***A volume is equal to one bin in a 2321 Data Cell.

If the programmer underestimates the space required, automatic volume switching may occur before the CLOSE REEL or UNIT is executed, since space on the first volume is filled. If this has happened, the CLOSE REEL or UNIT will start a third volume.

If no secondary allocation has been specified and the programmer issues a CLOSE REEL or UNIT statement, the job will terminate abnormally, since the allocation of subsequent volumes is taken from the secondary allocation field of the SPACE parameter.

Note: In the creation of a BSAM output file, performance is improved by specifying the CONTIG subparameter of the SPACE parameter in the DD statement. However, space allocation is more efficient if CONTIG is not specified.

To determine the amount of space a data set requires, the following variables should be considered:

- Device type
- Track capacity
- Tracks per volume
- Cylinders per volume
- Data length (block size)
- Key length
- Device overhead

Device overhead refers to the space required on each track for hardware data, i.e., address markers, count areas, gaps between records, Record 0, etc. Device overhead varies with each device and also depends on whether the blocks are written with keys. The formulas in Table 7 may be used to compute the actual space required for each block, including device overhead.

Table 8 lists device storage capacity, and Table 9 lists capacity in records per track for several direct-access storage devices.

Programmers requiring more detailed information on direct-access storage devices may refer to the publication IBM System/360 Component Descriptions --
2841 Storage Control
2302 Disk Storage, Models 3 and 4
2311 Disk Storage Drive, Model 1
2321 Data Call Drive
2303 Drum Storage,
Form A26-5988.

BSAM PARAMETERS AND SUBPARAMETERS: Figures 13 and 14 show the parameters that may be used in a DD statement when processing with BSAM. BSAM files must be on direct-access devices. Therefore, the UNIT, VOLUME, and SPACE parameters that specify direct-access devices must be used when creating the file. However, note that the SPLIT parameter cannot be used.

The DCB subparameters for BSAM are as follows:

1. The DSORG=DA subparameter must always be specified for COBOL for files created by use of BSAM.
2. The subparameter OPTCD={W} is used to request a service provided by the control program. The subparameter is discussed under QSAM DCB subparameters.

Table 9. Direct-Access Device Track Capacity

Maximum Bytes per Record Formatted without Keys						Records per Track	Maximum Bytes per Record Formatted with Keys					
2311	2314	2302	2303	2301	2321		2311	2314	2302	2303	2301	2321
3625	7294	4984	4892	20483	2000	1	3605	7249	4964	4854	20430	1984
1740	3520	2403	2392	10175	935	2	1720	3476	2383	2354	10122	920
1131	2298	1570	1558	6739	592	3	1111	2254	1505	1520	6686	576
830	1693	1158	1142	5021	422	4	811	1649	1139	1104	4968	406
651	1332	912	892	3990	320	5	632	1288	893	854	3937	305
532	1092	749	725	3303	253	6	512	1049	730	687	3250	238
447	921	634	606	2812	205	7	428	877	614	568	2759	190
384	793	546	517	2444	169	8	364	750	527	479	2391	154
334	694	479	447	2157	142	9	315	650	460	409	2104	126
295	615	425	392	1928	119	10	275	571	406	354	1875	103
263	550	381	346	1741	101	11	244	506	362	308	1688	85
236	496	344	308	1585	86	12	217	452	325	270	1532	70
213	450	313	276	1452	73	13	194	407	294	238	1399	58
193	411	286	249	1339	62	14	174	368	267	211	1286	47
177	377	i64	225	1241	53	15	158	333	245	187	1188	38
162	347	244	204	1155	44	16	143	304	224	166	1102	29
149	321	225	186	1079	37	17	130	277	206	148	1026	21
138	298	209	169	1012	30	18	119	254	190	131	959	15
127	276	196	155	952	24	19	108	233	176	117	899	9
118	258	183	142	897	20	20	99	215	163	104	844	
109	241	171	130	848	15	21	90	198	152	92	795	
102	226	161	119	804	10	22	82	183	142	81	751	
95	211	151	109	763	6	23	76	168	132	71	710	
88	199	143	100	726		24	69	156	123	62	673	
82	187	135	92	691		25	63	144	116	54	638	
77	176	127	84	659		26	58	133	108	46	606	
72	166	121	77	630		27	53	123	102	39	577	
67	157	114	70	603		28	48	114	95	32	550	
63	148	108	64	577		29	44	105	89	26	524	
59	139	102	58	554		30	40	96	83	20	501	

Organization	Type of OPEN Statement	Input/Output Statement	Required Key Clauses	Other Required Clauses	Optional Clauses
DIRECT	INPUT	READ AT END	SYMBOLIC KEY	ASSIGN TO DIRECT-ACCESS LABEL RECORDS ARE STANDARD DATA RECORDS CLOSE	RECORD CONTAINS USE (option 2) FILE-LIMIT (with output files) CLOSE [UNIT] [REEL]
	OUTPUT	WRITE	SYMBOLIC KEY ACTUAL KEY		
RELATIVE	INPUT	READ AT END	Not permitted		RECORD CONTAINS SAME AREA USE (option 2) CLOSE [UNIT] [REEL]
	OUTPUT	WRITE			

• Figure 12. Permissible COBOL Clauses for BSAM

DSNAME	Device	UNIT	VOLUME	LABEL	SPACE	SUBALLOC	SPLIT	DISP	DCB	
									Required	Optional
AS	Direct-Access required	AS	AS	SL	AS RLSE[1]	AS	NA	NEW (,KEEP / ,CATLG / ,PASS / ,DELETE) Note: MOD not meaningful	DSORG=DA	OPTCD={W}

AS = Applicable subparameters
NA = Not applicable
[1]For direct organization only

Figure 13. DD Statement Parameters Applicable to BSAM Output Files

DSNAME	Device	UNIT	VOLUME	LABEL	SPACE, SUBALLOC, SPLIT	DISP	DCB
AS	Direct-Access required	not required if cataloged		SL	NA	OLD (,PASS / ,KEEP / ,CATLG / ,DELETE / ,UNCATLG)	NA

AS = Applicable subparameters
NA = Not applicable

Figure 14. DD Statement Parameters Applicable to BSAM Input Files

Processing with BDAM

Figure 15 shows the COBOL clauses that may be used with BDAM. Special considerations are as follows:

1. For BDAM with relative record addressing the SYMBOLIC KEY is required. It contains the relative record number (or position within the file) of the record to be read or rewritten (updated). (For BSAM with relative record addressing, the position of the logical records in a file is determined relative to the first record of the file, starting with the initial value of zero.) The WRITE verb is not allowed for relative record addressing, because the extent of the file cannot be increased.

2. For BDAM input files with relative track addressing, the ACTUAL KEY clause is used to locate the track on which the search for the record is to begin. The SYMBOLIC KEY clause is used to identify the location of the record on the track.

3. For BDAM I-O files with relative track addressing, the ACTUAL KEY clause is used to locate the track on which the search begins for space to insert a record for the WRITE verb and to locate a record for the READ verb.

Note: The programmer may wish to build a BDAM file with relative track addressing in a random manner. In COBOL, however, the tracks of the file must first be formatted using BSAM. This can be done by opening the file as output and specifying the FILE-LIMIT clause for the file. The programmer only need open and close the file, and the necessary capacity or dummy records will be written on the number of tracks specified in the FILE-LIMIT clause. The programmer can then open the file as I-O and proceed to build his file using BDAM.

Figure 16 shows the DD statement parameters that may be used with BDAM. Note that the SPACE, SPLIT, SUBALLOC, and DCB parameters are not applicable.

Processing Indexed Sequential Files with QISAM and BISAM

Figures 17 and 18 show the DD statement parameters that may be used with QISAM output files and QISAM and BISAM input and I-O files. Note that the SUBALLOC and SPLIT parameters are not applicable for any of these files.

FILE ORGANIZATION: An indexed sequential file is one in which records are arranged on the tracks of a direct-access device in a sequence determined by keys. The key normally is a control field that is an intrinsic part of the information in the record (e.g., the part number), and is specified by the RECORD KEY clause in the source program. Records are written or read by the use of indexes. These indexes provide flexibility in that the programmer can:

- Write and later read or update logical records in a sequential, ascending order (using QISAM) based on the collating sequence of the keys. This is done in a manner similar to that for sequential organization.

- Read or update individual logical records in a random manner (using BISAM). This method is somewhat slower per record than reading according to a collating sequence, since a search for pointers in indexes is required for the retrieval of each record.

- Insert new logical records at any point within the file (using BISAM). Using the indexes, the system locates the proper position for the new record and makes all necessary adjustments so that the order of the records, according to the keys, is maintained.

Indexes: There are two basic types of indexes, track indexes and cylinder indexes. There is one track index for each cylinder in the prime area (see "Indexed Sequential File Areas" for a description of prime area). The track index is written on the first track of the cylinder that it indexes. An entry in the track index contains the identification of a specific track in the cylinder and the highest key on that track.

There is one cylinder index for each file in which prime area data takes up more than one cylinder. The cylinder index contains one entry for each cylinder in the prime area, each entry pointing to the track index in a particular cylinder.

Organization	Type of OPEN Statement	Input/Output Statements	Required Key Clauses	Other Required Clauses	Optional Clauses
DIRECT	INPUT	READ [INVALID KEY]	SYMBOLIC KEY ACTUAL KEY	ASSIGN TO DIRECT-ACCESS LABEL RECORDS ARE STANDARD DATA RECORDS CLOSE	RECORD CONTAINS SAME AREA APPLY (option 1) USE (option 2)
	I-O	READ WRITE REWRITE [INVALID KEY]			
RELATIVE	INPUT	READ [INVALID KEY]	SYMBOLIC KEY		RECORD CONTAINS SAME AREA SAME AREA USE (option 2)
	I-O	READ REWRITE [INVALID KEY]			

Figure 15. Permissible COBOL Statements for BDAM

DSNAME	Device	UNIT	VOLUME	LABEL	SPACE, SUBALLOC, SPLIT	DISP	DCB
Applicable subparameters	Direct-Access Required	not required if cataloged		SL	NA	OLD (,PASS / ,KEEP / ,CATLG / ,UNCATLG / ,DELETE)	NA

NA = Not Applicable

Figure 16. DD Statement Parameters Applicable to BDAM Input and I-O Files

If the file is being read randomly, the system locates the given record by its key after a search of a cylinder index and the track index within the indicated cylinder. If the file is being read sequentially starting with the first record, no index search is necessary.

Indexed Sequential File Areas: The programmer specifies the structure of an indexed file and space to be allocated for it in the DD statement for the file when the file is created. In some instances, more than one DD statement is required. (These DD statements are described in "Using the DD Statements -- Single Volume Files.") The space being allocated must be divided into one, two, or three areas, depending on the needs of the programmer. These areas are: prime area, index area, and overflow area. The overflow area is optional.

Prime Area. The prime area is the area in which data records are written when the file is created or reorganized. These records are in a sequence determined by the record keys. The track indexes also use a portion of the reserved prime area. To reserve prime area space so that new logical records may be inserted without forcing records into an overflow area (described below), dummy records may be written when the file is being created. Dummy records are described later in "Processing with QISAM: Creating Indexed Sequential Files." The prime area may span multiple volumes and may consist of several noncontiguous areas.

Index Area. The index area contains the cylinder indexes and, if requested, master indexes (described later) for the file. This area exists for any file that has a prime area on more than one cylinder. Space for this area will be allocated separately from the prime area if specifically requested. The index area must be contained within one volume, but that volume need not be the same device type as the prime area volume. If not specifically requested, the index area will automatically be constructed in the independent overflow area, or, if none, in the prime area.

Parameter	Applicable Subparameters
ddname	ddname used only for first DD statement of each file.
DSNAME[1]	{dsname / &name} [(INDEX) / (PRIME) / (OVFLOW)]
Device	Direct-access required
UNIT	DEFER not permitted
SEP AFF	Restricted, see "Job Control Procedures"
VOLUME	Volume sequence number subparameter not applicable
LABEL	SL
SPACE	CYL,[,,CONTIG]ABSTR
SUBALLOC	Not applicable
SPLIT	Not applicable
DISP	NEW[2] [,KEEP / ,PASS / ,DELETE]
DCB[3]	Required: DSORG=IS Optional: BUFNO=xxx BLKSIZE OPTCD=[W][M][Y][I][R]

[1]If more than one DD statement is used, elements must be in this order
[2]MOD not meaningful. CATLG allowed only if all areas are allocated with a single DD statement
[3]The DCB parameter should be the same for each DD statement

Figure 17. DD Statement Parameters Applicable to QISAM Output Files

Overflow Area. The overflow area is the area in which space is allocated for records forced from their original (prime) tracks by the insertion of new records. The fact that some records are stored in these areas, physically out of sequence, does not change the ability of QISAM to read the file in a logical sequence. An overflow area need not be specified if records are either not going to be added to the file, or sufficient space has been reserved by writing delete records in the prime area.

There are three ways in which space for an overflow area may be allocated:

1. Space may be requested for an independent overflow area, using the dsname (OVFLOW) DD statement, either on the same volume or on a separate volume of the same device type as that of the prime area.

2. Tracks on each cylinder can be reserved to hold the overflow of that cylinder (cylinder overflow option).

3. If the prime area is not filled when the file is created, the remaining space will be designated as an independent overflow area, even though it is not requested directly.

Additional information about indexed sequential file structure is contained in the publication IBM System/360 Operating System: Supervisor and Data Management Services, Form C28-6646.

PROCESSING WITH QISAM; CREATING INDEXED SEQUENTIAL FILES: The structure of an indexed file and the space to be allocated is specified in the DD statement(s). The space can be subdivided and allocated in several different ways; the allocation specified on the DD statement(s) must be sufficient for all the areas used.

QISAM DD Statement Requirements: The special parameter requirements for DD statements that define new indexed sequential files are discussed below. The discussion is oriented to indexed sequential files on one volume. Many of the parameters used for creating multi-volume files are not discussed here. For more detailed information about parameters for both single and multi-volume files, see the publication IBM System/360 Operating System: Job Control Language, Form C28-6539.

ddname (name field)
The name field of the first or only DD statement defining the indexed sequential file can contain the symbolic identification 'ddname' or procstep.ddname. Succeeding DD statements for the file must not be named.

DSNAME
This parameter must be specified and is coded as follows:

DSNAME= {dsname / &name} [(element)]

The first subparameter, dsname or &name, must be the same in all the DD statements defining one data set. The element subparameter, INDEX, PRIME, or OVFLOW, indicates the type of area defined by the DD statement. If more than one DD statement is used to define a file, the order in which the statements should be placed in the input stream is as follows:

```
DD  DSNAME=dsname(INDEX)
DD  DSNAME=dsname(PRIME)
DD  DSNAME=dsname(OVFLOW)
```

Deviation from this order results in abnormal termination of the job. If the element subparameter is omitted, PRIME is assumed. Note that an indexed sequential file cannot be specified by statements containing only index and overflow elements.

SPACE

This parameter specifies the space to be allocated for each of the separate areas on the device, and must be included. Only cylinder (CYL) or absolute track (ABSTR) requests are permitted, and with ABSTR the designated tracks must encompass an integral number of cylinders. All the DD statements defining one indexed sequential file must specify the same subparameter, either CYL or ABSTR. When all the DD statements specify CYL, all must also specify or omit CONTIG, depending on whether the space allocated is to be contiguous or noncontiguous. The directory or index quantity subparameter of the SPACE parameter is used to request the number of cylinders to be allocated for an index area embedded within the prime area (see "SPACE Parameter" in the section entitled "Job Control Procedures"). An embedded index resides in the middle of a track and saves searching time by first determining which half of the track contains the requested record.

SPLIT

This parameter should never be specified for a QISAM file, either for sharing a cylinder with QISAM files or for sharing it with a QISAM file and another type of file.

DISP

This parameter is written as it would be for any new file that cannot be cataloged. The CATLG subparameter must not be specified unless only one DD statement is used to allocate the file space. (See "Cataloging Files" for additional information about cataloging indexed sequential files.)

Parameter	Applicable Subparameters
ddname	ddname used only for first DD statement of each file
DSNAME[1]	dsname
Device	Direct-access required
UNIT[2]	Applicable subparameter
SEP AFF	Restricted, see "Job Control Procedures"
VOLUME	Applicable subparameters
LABEL	SL
SPACE	Not applicable
SUBALLOC	Not applicable
SPLIT	Not applicable
DISP	OLD[3] [,PASS / ,KEEP / ,DELETE]
DCB	Required: DSORG=IS Optional: BUFNO=xxx (not allowed for BISAM)

[1]Element subparameter must not be used
[2]Not needed if file is cataloged
[3]CATLG, UNCATLG not permitted

Figure 18. DD Statement Parameters Applicable to QISAM and BISAM Input and I-O Files

DCB

This parameter must be specified for each DD statement and is coded as follows:

```
DCB=(DSORG=IS
       [,BUFNO=integer]
       [,OPTCD=[L][Y][I][R][W]
       [M,NTM=integer]]
       [,BLKSIZE=integer])
```

DSORG=IS

The DSORG=IS subparameter is required and indicates that the organization of the file is indexed sequential. The DCB subparameters of all the DD statements defining one file must not conflict. For example, if the OPTCD=Y subparameter appears in the first DD statement, the subsequent DD statements should also contain OPTCD=Y. The subparameters are discussed below.

BUFNO=number of buffers

This subparameter is used to specify the number of buffers to be assigned

to the file if no RESERVE clause is specified for the file in the source program. The maximum number is 255; however, the maximum number allowed for an installation may differ and is established at system generation time.

OPTCD=options
This subparameter is used to tell the system that certain additional facilities are to be provided for this file. Any combination of the following options can be specified for the OPTCD subparameter. If more than one option is specified, the options are written as a character string (i.e., without intervening blanks or commas). Note that if certain of these options are used, an additional subparameter must also be specified as indicated. The following options are unconditionally provided:

1. The operating system supplies OPTCD=L and I.
2. The operating system supplies OPTCD=W for the IBM 2321 direct-access device.

- OPTCD=L: This option requests that the control program delete marked records. Marked records will be deleted when space for new records is required.
- OPTCD=Y: This option requests that a cylinder overflow area be created. It specifies that a certain number of tracks on each cylinder are to be reserved to contain any overflow records from other tracks on that cylinder. Another DCB subparameter, CYLOFL=xx, must also be written. xx specifies the number of tracks on the cylinder to be reserved for the overflow area. The maximum number is 99.
- OPTCD=I: This option requests that an independent overflow area be reserved. It is used in conjunction with DSNAME=dsname (OVFLOW) parameter in the DD statement used to allocate the independent area.
- OPTCD=M: This option requests that a master index be created (see "Additional Facilities" in this section for a discussion of master indexes). Another DCB subparameter, NTM=xx, must also be written. It specifies the maximum number of tracks to be contained in the cylinder index before a higher level index is created. The maximum value that can be specified is 99.
- OPTCD=R: This option requests reorganization criteria feedback, as described in "Reorganizing an Indexed Sequential File" in this section.
- OPTCD=W: This option requests the system to perform a write-validity check. This facility is described under "Processing with QSAM."

The following is an example of how the OPTCD subparameter can be used:

```
DCB=(DSORG=IS,OPTCD=M,NTM=20)
```

This example requests that a master index be created when the cylinder index exceeds 20 tracks.

BLKSIZE=integer
specifies the blocksize. This clause is used only if BLOCK CONTAINS 0 RECORDS was specified at compile time.

Using the DD Statements -- Single Volume Files: The following examples all refer to files that can be contained on one volume. More details about DD statements, including information on multi-volume file allocation, can be found in the publication IBM System/360 Operating System: Job Control Language, Form C28-6539.

All three areas for an indexed sequential file can be contained on a single volume if they are small enough. If this is the case and the programmer elects to allow the system to subdivide storage into the prime and index areas when the file is created, he need only code the following DD statement:

```
//ddname  DD  DSNAME=dsname(PRIME),    X
//              SPACE=(CYL,(no. of      X
//              cylinders)),UNIT=unit,  X
//              DCB=(DSORG=IS,...)
```

This DD statement will produce a prime area with the index area occupying the last cylinder(s) of the space in the prime area. If any tracks are left over on the last cylinder after the index area, they are used as the overflow area; if no tracks are left over, an overflow area does not exist.

If the programmer definitely wants an independent overflow area, he must provide

```
//ddname   DD  DSNAME=dsname(PRIME),    X
//                SPACE=(CYL,(no. of     X
//                cylinders)),UNIT=unit, X
//                VOLUME=SER=222222,     X
//                DCB=(DSORG=IS,...)
//           DD  DSNAME=dsname(OVFLOW),  X
//                SPACE=(CYL,(no. of     X
//                cylinders)),UNIT=unit, X
//                VOLUME=SER=222222,     X
//                DCB=(DSORG=IS,...)
```

These DD statements will produce a prime area and a separate overflow area with the index area at the end of the overflow area. All three areas reside on the same volume.

Note: When more than one DD statement is used, only the first can be named. The others must not have a data definition name (ddname) but all must have the same data set name (dsname).

If the programmer desires more control in the placement of the index area, he can subdivide storage before the data set is created by providing another DD statement as follows:

```
//ddname   DD  DSNAME=dsname(INDEX),    X
//                SPACE=(CYL,(no. of     X
//                cylinders)),UNIT=unit, X
//                VOLUME=SER=333333,     X
//                DCB=(DSORG=IS,...)
//           DD  DSNAME=dsname(PRIME),    X
//                SPACE=(CYL,(no. of     X
//                cylinders)),UNIT=unit, X
//                VOLUME=SER=333333,     X
//                DCB=(DSORG=IS,...)
```

These DD statements will produce two separate areas: index and prime. Each area is on the same volume.

If, along with more control of his index, the programmer wishes an independent overflow area, he can specify a third DD statement as described above. The order will then be:

```
//ddname   DD  DSNAME=dsname(INDEX),...
//           DD  DSNAME=dsname(PRIME),...
//           DD  DSNAME=dsname(OVFLOW),...
```

These DD statements will produce three separate areas: index, prime, and overflow.

Note that the OPTCD subparameter of the DCB parameter in each of the DD statements must specify an independent overflow area (OPTCD=I). All three areas reside on the same volume if so specified in the VOLUME parameter.

Note: The order of the DSNAME parameter elements in all of the examples above must be followed when placing the DD statements into the input stream, or an abnormal termination of the job will result.

The example in Figure 19 defines a new indexed sequential file that consists of three separate areas. All three areas reside on the same volume. The volume is on an IBM 2311 Disk Storage Drive.

Cataloging Files: An indexed sequential file can be cataloged if:

- All the areas of the file are allocated with a single DD statement. Such a file is cataloged in the usual manner by specifying the DISP parameter in the DD statement:

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

- The areas are allocated with more than one DD statement, but all volumes are on the same type of device. Such a file is cataloged using the IEHPROGM utility program (see the publication IBM System/360 Operating System: Utilities, Form C28-6586).

An indexed sequential file that is being created cannot be cataloged if its areas are on different device types. An existing indexed sequential file cannot be cataloged through the specification of the CATLG subparameter of the DISP parameter in the DD statement.

Note: The DD statement or statements defining a new or existing indexed sequential file can appear in cataloged procedures.

```
//FILE     DD  DSNAME=ISM(INDEX),UNIT=2311,SPACE=(CYL,(1)),  X
//                VOLUME=SER=111111,DCB=(DSORG=IS,...)
//           DD  DSNAME=ISM(PRIME),UNIT=2311,SPACE=(CYL,(5)),  X
//                VOLUME=SER=111111,DCB=(DSORG=IS,...)
//           DD  DSNAME=ISM(OVFLOW),UNIT=2311,SPACE=(CYL,(1)), X
//                VOLUME=SER=111111,DCB=(DSORG=IS,...)
```

Figure 19. Example of DD Statements for New Indexed Sequential Files

Calculating Space Requirements: To determine the number of cylinders required for an indexed sequential file, the programmer must consider the number of records that will fit on a cylinder, the number of records that will be processed, and the amount of space required for indexes and overflow areas. In making the computations, additional space is also required for device overhead. Detailed information can be found in the publication IBM System/360 Operating System: Supervisor and Data Management Services, Form C28-6646.

Additional Facilities; The Master Index: QISAM provides a master index facility to avoid inefficient serial searches of large cylinder indexes. The master index provides an index to the cylinder index. The programmer can specify with the DCB parameter in his DD statement(s) (see "QISAM DD Statement Requirements" above) that a master index be built if the size of a cylinder index exceeds a certain number of tracks. Each entry in the master index points to a track of the cylinder index. If the size of the master index exceeds the number of tracks specified in the NTM parameter of the DD statement, the master index is automatically indexed by a higher level master index. Three such higher level master indexes can be constructed.

COBOL Considerations: When creating indexed sequential files, the QISAM file processing technique is used. The following COBOL programming considerations should be noted:

- RECORD KEY Clause. The RECORD KEY clause in the SELECT sentence of the Environment Division is required. It is used to specify the location of the key within the record itself. If the RECORD KEY clause has a PICTURE clause that specifies that the item is binary (COMPUTATIONAL), zero is the lowest number acceptable as the first record. A negative key is considered to be larger than a positive key; therefore, if a record is inserted into the file, a negative key would place the record after those records with positive keys.

- Dummy Records. To reserve space for records to be added at a later time, when creating indexed sequential files, dummy records can be written with the delete code (the figurative constant HIGH-VALUE) in the first byte. Dummy records and deletion are described later in this section.

PROCESSING WITH QISAM; READING OR UPDATING INDEXED SEQUENTIAL FILES: The QISAM file processing technique can be used to read or update an existing indexed sequential file. Adding a record to an already existing file, however, can only be done with BISAM (see "Processing with BISAM" below).

When the QISAM file processing technique is used to read an input file, the READ statement makes available one logical record at a time in an ascending order determined by the record keys. Dummy records are not made available. If there are records in the overflow area, this order will not correspond exactly to the physical order of the records in the file. The file must have been created using QISAM.

When the QISAM file processing technique is used to update an I-O file, the READ and REWRITE statements permit updating-in-place or deletion of a logical record. A logical record is updated-in-place by reading, updating, and rewriting it from the same area. Alteration of record length or insertion of new records is not permitted. A logical record is marked for deletion by moving the figurative constant HIGH-VALUE into the first character position of the record and then using the REWRITE statement. Records in the file that contain this deletion code are not made available.

The discussion that follows is primarily concerned with QISAM files that can be contained on a single volume. Additional information about processing existing QISAM files, including multi-volume files, can be found in the publication IBM System/360 Operating System: Job Control Language, Form C28-6539.

Parameter Requirements: In the DD statement(s) indicating an existing indexed sequential file, the following differences and requirements should be noted:

DCB

The DSORG=IS subparameter must be specified, whereas the BUFNO subparameter is optional. The OPTCD field must not be specified again. Any OPTCD subparameter facilities that were specified when the file was created are in effect as long as the data set exists. For example, if the programmer specified the write-validity check option (OPTCD=W) when he created the file, the option is still in effect at the time of any subsequent WRITE statement. The BLKSIZE subparameter must not be specified.

DSNAME

This parameter is written DSNAME=dsname. The element subparameters (INDEX, PRIME, OVFLOW), must not be written.

DISP

The first subparameter must be OLD. The second subparameter cannot be CATLG or UNCATLG (see "Cataloging Files" above for more information on cataloging indexed sequential files).

For further information about QISAM parameters, see the section "QISAM DD Statement Requirements" above.

Using the DD Statement -- Single-Volume Files: Only one DD statement is needed to specify an existing file if all of the areas are on one volume. The following is an example of a DD statement that can be used when processing a single-volume QISAM file.

```
//ddname   DD  DSNAME=dsname,DCB=          X
//               (DSORG=IS,...),           X
//               UNIT=unit name,DISP=OLD
```

Further details about DD statements for existing single-volume and multi-volume QISAM files can be found in the publication IBM System/360 Operating System: Job Control Language, Form C28-6539.

Additional Facilities; Reorganizing Files: As new records are added to an indexed sequential file, chains of records may be created in the overflow area, if one exists. The access time for retrieving records in an overflow area is greater than that required for retrieving records in the prime area. I-O performance is, therefore, sharply reduced when many overflow records develop. For this reason, an indexed sequential file can be reorganized as soon as the need becomes evident. The system maintains a set of statistics to assist the programmer when reorganization is desired. These statistics are maintained as fields of the file's data control block. The location of the data control block can be passed to a called assembler language program, which can then test its fields. If these statistics are desired, the OPTCD subparameter of the DCB parameter must have included OPTCD=R in each of the DD statements when the file was created. Additional information is contained in the publication IBM System/360 Operating System: Supervisor and Data Management Services, Form C28-6646.

SETL Macro Instruction: For QISAM input and I-O files, retrieval starts with the first non-dummy record in the file. If the programmer wishes to begin processing at a point other than the beginning of the file, he can do so through the use of the SETL macro instruction; however, the programmer must set (with a logical OR) the MACRF field of the DCB for the file prior to the OPEN for the file. He can then specify the record key or device address at which retrieval is to start.

The SETL macro instruction can be used more than once in the course of processing the file; however, an ESETL macro instruction must be specified before the new SETL macro instruction can be executed (see "Calling and Called Programs" for conventions used in calling assembler language programs).

COBOL Considerations: When processing with QISAM, the following COBOL programming considerations should be noted:

- RECORD KEY Clause. The RECORD KEY clause in the SELECT sentence of the Environment Division is required, just as it is when creating the file. Note other record key considerations under "Processing with BISAM."

- Delete Option. In order to keep the number of records in the overflow area to a minimum, and to eliminate unnecessary records, an existing record may be marked for deletion. This is done by moving the figurative constant HIGH-VALUE into the first character position of the record. The record is not physically deleted unless it is forced off its prime track by the insertion of a new record, or if the file is reorganized (see "Additional Facilities" above). The deleted record may be replaced (using BISAM) by a record with an equal key without an error being indicated. Execution of the READ statement in QISAM does not produce a record marked for deletion, whether the record has been physically deleted or not. Dummy records and deletion are discussed further in "Processing with BISAM."

- If, in the same COBOL program, the programmer desires to create an indexed sequential file, close it, and then open it as an input or I-O file, he must specify another file name. The same DD statement(s) may not be used.

- A separate set of DD statements is required for each of the following operations on the same file:

  1. Creating a QISAM file
  2. Opening it as a QISAM I-O or input file
  3. Opening it as a BISAM file

  The same file name is used each time in the DSNAME parameter.

PROCESSING WITH BISAM; READING, WRITING, AND REWRITING INDEXED SEQUENTIAL FILES: The BISAM file processing technique can be used for the direct retrieval of any logical record by its key, the direct update-in-place of any logical record, and the direct insertion of new logical records into an indexed sequential file. Only files created by the use of QISAM can be referred to by BISAM.

Because of their complementary use of the indexed sequential file organization, much of the data given for QISAM in "Processing with QISAM" applies to BISAM. Differences will be noted below.

BISAM Keys: Both symbolic keys and record keys are required when using BISAM. The symbolic key is the field named in the SYMBOLIC KEY clause of the File-Control paragraph of the Environment Division. When a record is read or written using BISAM, it is located by matching the contents of its RECORD KEY field with the contents of the field named in the SYMBOLIC KEY clause.

Since a record key is used to identify a record to the system, the record keys associated with the logical records of the file may be thought of as a table of arguments. The symbolic key may be considered to be a search argument that is compared with entries in the table. When a new record is added to an existing file, the contents of the field named in the SYMBOLIC KEY clause are used to locate the two records in the file between which the new record is to be inserted. The records sought are those having respective RECORD KEY field contents less than and greater than the value in the SYMBOLIC KEY field.

Using WRITE: The programmer can use the WRITE statement to cause the insertion of a new record into an indexed sequentail file. The insertion is done on the basis of the value specified in the symbolic key. Two methods can be used to add records.

In the first method, the key to be added is a new key value. The record is inserted in place so that the order of the keys is maintained. If an overflow area exists, the insertion may cause records to be forced off the prime area track into the overflow area. Dummy records forced off the track in this way are physically deleted and are not written in the overflow area.

In the second method, the key of the record to be added has the same value as that of a known dummy record. If the dummy record has not been physically deleted, it is replaced by the new record. If it has been physically deleted, the record is inserted as if it had a new key value.

Records with a key higher (or lower) then the current highest (or lowest) key of the file may be added. When a record is added, both the record key and symbolic key must have the same value. Except in the case of dummy records, this value must be unique in the file.

Using REWRITE: If the REWRITE statement is being used to update a record in place, the programmer cannot change the size of a record between the time it is read and the time it is rewritten. When a BISAM file is opened as I-O and a REWRITE statement for the file-name is issued anywhere in the program, a REWRITE statement must be given after every READ statement and previous to any other input/output statement for the same file-name. However, a REWRITE statement should not be given if an invalid READ condition occurs.

If the programmer wishes to write a new record (RECORDY) before updating and rewriting the contents of a record he has previously read (RECORDX), he might use the following sequence of instructions:

```
READ statement -- to obtain RECORDX
   .
   .
   .
REWRITE statement -- for RECORDX
WRITE statement -- for RECORDY
READ statement -- for RECORDX
   .
   .
   .
Update contents of RECORDX
   .
   .
   .
REWRITE statement -- for RECORDX
```

Using READ: Records are retrieved on the basis of the value specified in the symbolic key. If the key of a record marked for deletion is specified and the record has not been physically deleted, it will be produced. If the record has been physically deleted, the READ statement will cause an invalid key condition and control will go to the invalid key routine, if specified.

COBOL Considerations: When processing with BISAM, the following COBOL programming considerations should be noted:

- RECORD KEY and SYMBOLIC KEY Clauses. The RECORD KEY and SYMBOLIC KEY clauses in the SELECT sentence of the Environment Division are required. The RECORD KEY clause is used to specify the location of the key within the record itself. The symbolic key is used as a search argument to locate the proper record, and must not be defined within

the file being processed. Note that since a record key is defined within a record, the contents of RECORD KEY are not available after a WRITE statement has been executed for that record.

- TRACK-AREA Clause. Specifying the clause results in a considerable improvement in efficiency when a record is added. If a record is added and the TRACK-AREA clause was not specified for the file, the contents of the SYMBOLIC KEY field are unpredictable after the WRITE statement is executed. In this case, the key must be reinitialized before the next WRITE statement is executed.

Each data set that is defined by a DD statement is either (1) to be created, or (2) has been previously created and is to be retrieved. In either case, the data set must have a disposition; for example, if the data set is being created, the disposition must indicate whether the data set is to be cataloged, kept, or deleted. Other DD statements may simply indicate that the data set is in the input stream or that ultimately the data set is to be printed or punched.

The following sections summarize the DD statement parameters and show examples for various uses of the DD statement. These sections include information about cataloging data sets and creating or referring to generation data groups; examples of cataloged data sets and partitioned data sets are included. For additional information about partitioned data sets refer to the chapter "Libraries." See "A Checklist for Job Control Procedures" for additional examples of the DD statement used in job control procedures.

CREATING A DATA SET

The programmer will ordinarily be concerned with the following parameters when he is creating a data set:

1. The data set name (DSNAME) parameter, which assigns a name to the data set being created.

2. The unit (UNIT) parameter, which allows the programmer to state the type and quantity of input/output devices to be allocated for the data set.

3. The volume (VOLUME) parameter, which allows specification of the volume in which the data set is to reside. This parameter also gives instructions to the system about volume mounting.

4. The space (SPACE), split cylinder (SPLIT), and suballocation (SUBALLOC) parameters, for direct-access devices only, which permit the specification of the type and amount of space required to accommodate the data set.

5. The label (LABEL) parameter, which specifies the type and some of the contents of the label associated with the data set.

6. The disposition (DISP) parameter, which indicates what is to be done with the data set by the system when the job step is completed.

7. The DCB parameter, which allows the programmer to specify additional information to complete the DCB associated with the data set (see "User Defined Files"). This allows additional information to be specified at execution time to complete the DCB constructed by the compiler for a data set defined in the source program.

Figure 20 shows the subparameters that are frequently used in creating data sets. Additional subparameters are discussed in "Format of the Job Control Statements."

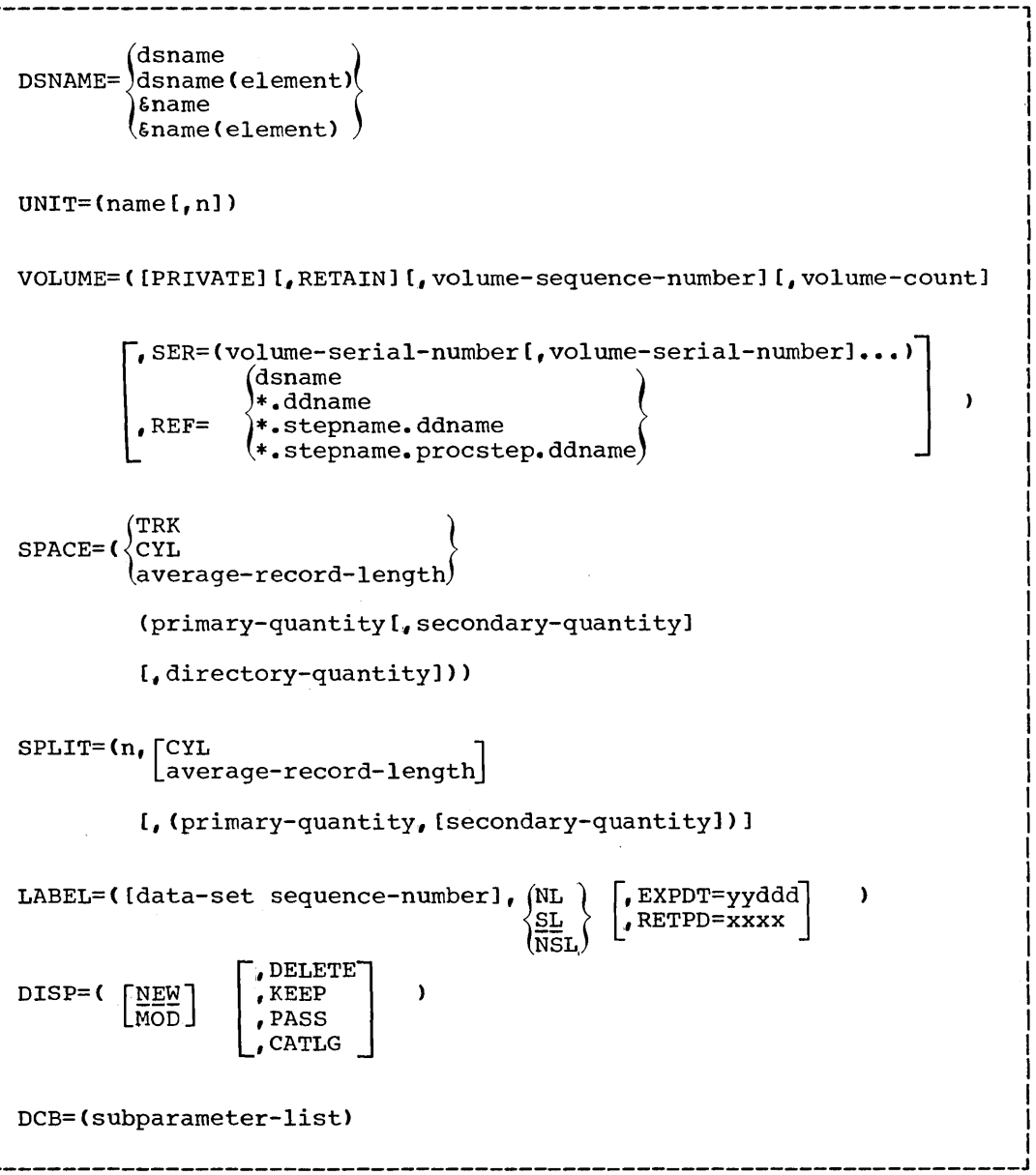


Figure 20. Parameters Frequently Used in Creating Data Sets

Examples of DD Statements Used to Create Data Sets

The following examples show various ways of specifying DD statements for data sets that are to be created. In general, the number of parameters and subparameters that are specified depend on the disposition of the data set at the end of the job step. If a data set is used only in the job step in which it is created and is deleted at the end of this job step, a minimum number of parameters are required. However, if the data set is to be cataloged, more parameters should be specified.

Example 1: Creating a data set for the current job step only.

```
//SYSUT1   DD   UNIT=SYSDA,SPACE=(TRK,(50,10))
```

This example shows the basic required DD statement for creating a data set and storing it on a direct-access device. The UNIT parameter

is required unless the unit information is available from another source. If the data set were to be stored on a unit record or a tape device, the SPACE parameter would not be needed. The operating system assigns a temporary data set name and assumes a disposition of (NEW, DELETE).

Example 2: Creating a data set that is used only for the current job.

```
//SYSLIN    DD DSNAME=&TEMP,DISP=(MOD,PASS),UNIT=SYSSQ,                X
//             SPACE=(TRK,(50))
```

This example shows a DD statement that creates a data set for use in more than one step of a job. The system assigns a unique symbol for the name, and this same symbol is substituted for each recurrence of the &TEMP name within the job. The data set is allocated space on any available direct-access or tape device. If a tape device is selected, the SPACE parameter is ignored. The disposition specifies that the data set is either new or is to be added to (MOD), and is to be passed to the next job step (PASS). This DD statement can be used for specifying the data set that is created as output from the compiler and is to be used as input to the linkage editor. By specifying MOD, separately compiled object modules can be placed in sequential order in the same volume.

Note: If MOD is specified for a data set that does not already exist, the job may be abnormally terminated if a volume reference name, a volume serial number, or the disposition CATLG is specified, or if the dsname is indicated by a backward reference.

Example 3: Creating a data set that is to be kept but not cataloged.

```
//TEMPFILE  DD DSNAME=FILEA,DISP=(,KEEP),SPACE=(TRK,(30,10)),          X
//             UNIT=DIRECT,VOLUME=(,RETAIN,SER=AA70)
```

The example shows a DD statement that creates a data set that is kept but not cataloged. The data set name is FILEA. The disposition (,KEEP) specifies that the data set is being created in this job step and is to be kept. It is kept until a disposition of DELETE is specified on another DD statement. KEEP implies that the volume is to be treated as private. Private implies that the volume is unloaded at the end of the job step but because RETAIN is specified, the volume is to remain mounted until the end of the job unless another reference to it is encountered. DIRECT is a hypothetical device class, containing only direct-access devices. The volume with volume serial number AA70, mounted on a device in this class, is assigned to the data set. Space for the data set is allocated as specified in the SPACE parameter. The data set has standard labels since it is in a direct-access volume.

If the volume serial number were not specified in the above example, the system would allocate space in an available non-private volume. Because KEEP is specified, the volume becomes private. (Another data set cannot be stored on a private volume unless its volume serial number is specified or affinity with a data set on the volume is stated.) The volume serial number of the volume assigned, if applicable, is included in the disposition message for the data set. Disposition messages are messages from the job scheduler, generated at the end of the job step.

Example 4: Creating a data set and cataloging it.

```
//DDNAMEA   DD DSNAME=INVENT.PARTS,DISP=(NEW,CATLG),                   X
//             LABEL=(,,EXPDT=67031),UNIT=DACLASS,                     X
//             VOLUME=(,REF=*.STEP1.DD1),                              X
//             SPACE=(CYL,(5,1),,CONTIG)
```

This example shows a DD statement that creates a data set named INVENT.PARTS and catalogs it in the previously created system catalog. The data set is to occupy the same volume as the data set referred to in the DD statement named DD1 occurring in the job step named STEP1. The UNIT parameter is ignored since REF is specified. Five cylinders are

allocated to the data set, and if this space is exhausted, more space is allocated, one cylinder at a time. The five cylinders are to be contiguous. The disposition (CATLG), implies that the volume is to be private. The INVENT.PARTS is to have standard labels. The expiration date is the 31st day of 1967.

Example 5: Adding a member to a previously created library.

```
//SYSLMOD   DD DSNAME=SYS1.LINKLIB(INVENT),DISP=OLD
```

This DD statement adds a member named INVENT to the link library (SYS1.LINKLIB). When a member is added to a previously created data set, MOD should be specified. The member INVENT takes on the disposition of the library.

Example 6: Creating a library and its first member.

```
//SYSLMOD   DD DSNAME=USERLIB(MYPROG),DISP=(,CATLG),                X
//             SPACE=(TRK,(50,30,3)),UNIT=2311,VOLUME=SER=111111
```

This DD statement creates a library, USERLIB, and places a member, MYPROG, in it. The disposition (,CATLG) indicates that the data set is being created in this job step (NEW is the default condition for the DISP parameter and is indicated by the comma) and is to be cataloged. The data set is to have standard labels. Space is allocated for the data set in a volume on a direct-access device which belongs to the device class 2311. Initially, 50 tracks are allocated to the data set, but if this space is exhausted, more tracks are added 30 at a time. The SPACE parameter must be specified when the library is created, and it must include allocation of space for the directory. SPACE cannot be specified when new members are added. If additional space is required when new members are added, the secondary allocation, if specified, will be used. Three 256-byte records are to be used for the directory. The volume serial number of the volume in which the library is to reside, is 111111.

Example 7: Replacing a member of an existing library.

```
//SYSLMOD   DD DSNAME=MYLIB(CASE3),DISP=OLD
```

This DD statement replaces the member named CASE3 with a new member with the same name. If the named member does not exist in the library, the member is added as a new member. In the above example, the library is cataloged.

Example 8: Creating and adding a member to a library used only for the current job.

```
//SYSLMOD   DD DSNAME=&USERLIB(MYPROG),DISP=(,PASS),UNIT=SYSDA,     X
//             SPACE=(TRK,(50,,1))
```

This DD statement creates and adds a member to a temporary library. It is similar to the DD statement shown in Example 6, except that a temporary name is used and the data set is not cataloged nor kept but is simply passed to the next job step. Since the data set is to be used only for this one job, it is not necessary to specify VOLUME and LABEL information. This statement can be used for a linkage editor job step in which the module is to be passed to the next step.

Note: If DISP=,(DELETE) is specified for a library, the entire library will be deleted.

RETRIEVING PREVIOUSLY CREATED DATA SETS

The parameters that must be specified in a DD statement to retrieve a previously created data set depend on the information that is available to the system about the data set. For example,

1. If a data set in a magnetic-tape or direct-access volume was created and cataloged in a previous job or job step, all information for the data set, such as volume, space, etc., is stored in the catalog and data set label. This information need not be repeated. Only the dsname and disposition parameters need be specified.

2. If the data set was created and kept in a previous job but has not been cataloged, information concerning the data set, such as space, record format, etc., is stored in the data set label. However, the unit and volume information must be specified unless otherwise available.

3. If the data set was created in the current job step, or in a previous job step in the current job, the information in the previous DD statement is available to the system and is accessible by referring to the previous DD statement. Only the dsname and disposition parameters need be specified.

Note: A programmer may wish to change the previous disposition of a data set. For example, if KEEP was specified when the data set was created, the DD statement that retrieves the data set may change the disposition by specifying CATLG.

Figure 21 shows the subparameters that are used to retrieve previously created data sets.

```
DSNAME= { dsname                  }
        { dsname(element)         }
        { *.ddname                }
        { *.stepname.ddname       }
        { &name                   }
        { &name(element)          }

UNIT=(name[,n])

DCB=(subparameter-list)

DISP=( [OLD]  [,DELETE ]
       [SHR]  [,KEEP   ]
       [MOD]  [,PASS   ]
              [,CATLG  ]
              [,UNCATLG] )

LABEL=(subparameter-list)

VOLUME=(subparameter-list)
```

Figure 21. Parameters Frequently Used in Retrieving Previously Created Data Sets

Example 1: Retrieving a cataloged data set.

```
//CALC     DD   DSNAME=PROCESS,DISP=(OLD,PASS)
```

This DD statement retrieves a cataloged data set named PROCESS. No UNIT or VOLUME information is needed. Since PASS is specified, the volume in which the data set is written is retained at the end of the job step. PASS implies that a later job step will refer to the data set. The last step in the job referring to the data set should specify the final disposition. If no other DD statement refers to the data set, the status of the data set as it existed before this job is assumed.

Example 2: Retrieving a data set that was kept but not cataloged.

```
//TEMPFILE DD   DSNAME=FILEA,UNIT=DIRECT,VOLUME=SER=AA70,DISP=OLD
```

This DD statement retrieves a kept data set named FILEA. (This data set is created by the DD statement shown in EXAMPLE 3 in creating data sets.) The data set resides on a device in a hypothetical device class, DIRECT. The volume serial number is AA70.

Example 3: Referring to a data set in a previous job step.

```
//SAMPLE   JOB
//STEP1    EXEC PGM=IEQCBL00,PARM=DECK
           .
           .
           .
//SYSLIN   DD   DSNAME=ALPHA,DISP=(NEW,PASS),UNIT=SYSSQ
//STEP2    EXEC PGM=IEWL
//SYSLIN   DD   *.STEP1.SYSLIN,DISP=(OLD,DELETE)
```

The DD statement SYSLIN in STEP2 refers to the data set defined in the DD statement SYSLIN in STEP1.

Example 4: Retrieving a member of a library.

```
//BANKING  DD   DSNAME=PAYROLL(HOURLY),DISP=OLD
```

The DD statement retrieves a member, HOURLY, from a cataloged library, PAYROLL.

DD STATEMENTS THAT SPECIFY UNIT RECORD DEVICES

A DD statement may simply indicate that data follows in the input stream or that the data set is to be punched or printed. Figure 22 shows the parameters of special interest for these purposes.

{* {DATA}	
SYSOUT=A	
UNIT=name	
DCB=(subparameters);	Note: The DCB parameter can be specified, where permissible, for data sets on unit record devices. For example, it can be specified for compiler data sets (other than SYSUT1, SYSUT2, SYSUT3, and SYSUT4) and data sets specified by the DD statements required for the ACCEPT and DISPLAY statements, when any of these data sets are assigned to unit-record devices.

Figure 22. Parameters Used to Specify Unit Record Devices

Example 1: Specifying data in the card reader.

```
//SYSIN     DD *
```

The asterisk indicates that data follows in the input stream. This statement must be the last DD statement for the job step. The data must be followed by a delimiter statement.

Example 2: Specifying a printer data set.

```
//SYSPRINT  DD SYSOUT=A
```

SYSOUT is the system output parameter; A is the standard device class for printer data sets.

Example 3: Specifying a card punch.

```
//SYSPUNCH  DD SYSOUT=B
```

B is the standard device class for punch devices.

CATALOGING A DATA SET

A data set is cataloged whenever CATLG is specified in the DISP parameter of the DD statement that creates or uses it. This means that the name and volume identification for the data set are placed in a system index called the catalog. (See "Processing with QISAM" in the section "Execution Time Data Set Requirements" for information about cataloging indexed sequential data sets.) The information stored in the catalog is always available to the system; consequently, only the data set name and disposition need be specified in subsequent DD statements that retrieve the data set. See Example 4 in "Creating Data Sets," and Example 1 in "Retrieving Data Sets."

If DELETE is specified for a cataloged data set, any reference to the data set in the catalog is deleted unless the DD statement containing DELETE retrieves the data set in some way other than by using the catalog. If UNCATLG is specified for a cataloged data set, only the reference in the catalog is deleted; the data set itself is not deleted.

Note: A cataloged data set should not be confused with a cataloged procedure (see "Cataloged Procedures").

GENERATION DATA GROUPS

It is sometimes convenient to save data sets as elements or generations of a generation data group (DSNAME=dsname(element)). A generation data group is a collection of successive historically related data sets. Identification of data sets that are elements of a generation data group is based upon the time the data set is added as an element. That is, a generation number is attached to the generation data group name to refer to a particular element. The name of each element is the same, but the generation number changes as elements are added or deleted. The most recent element is 0, the element added previous to 0 is -1, the element added previous to -1 is -2, etc. A generation data group must always be cataloged.

For example, a data group named PAYROLL might be used for a weekly payroll. The elements of the group are:

```
PAYROLL(0)
PAYROLL(-1)
PAYROLL(-2)
```

where PAYROLL(0) is the data set that contains the information for the most current weekly payroll and is the most recent addition to the group.

When a new element is added, it is called element(+n), where n is an integer greater than 0. For example, when adding a new element to the weekly payroll, the DD statement defines the data set to be added as PAYROLL(+1); at the end of the job the system changes its name to PAYROLL(0). The element that was PAYROLL(0) at the beginning of the job becomes PAYROLL(-1) at the end of the job, and so on.

If more than one element is being added in the same job, the first is given the number (+1), the next (+2) and so on.

NAMING DATA SETS

Each data set must be given a name. The name can consist of alphanumeric characters and the special characters hyphen and plus 0 (12-0 multipunch). The first character must be alphabetic. The name can be assigned by the system, it can be given a temporary name, or it can be given a user-assigned name. If no name is specified on the DD statement that creates the data set, the system assigns the data set a unique name for the job step. If a data set is used only for the duration of one job, it can be given a temporary name (DSNAME=&name). If a data set is to be kept but not cataloged, it can be given a simple name. If the data set is to be cataloged it should be given a fully qualified data set name. The fully qualified data set name is a series of one or more simple names joined together so that each represents a level of qualification. For example, the data set name DEPT999.SMITH.DATA3 is composed of three simple names that are separated by periods to indicate a hierarchy of names. Starting from the left, each simple name indicates an index or directory within which the next simple name is a unique entry. The rightmost name identifies the actual location of the data set.

Each simple name consists of one to eight characters, the first of which must be alphabetic. The special character period (.) separates simple names from each other. Including all simple names and periods, the length of a data set name must not exceed 44 characters. Thus, a maximum of 21 qualification levels is possible for a data set name.

Programmers should not use fully qualified data set names which begin with the letters SYS and which also have a P as the 19th character of the name. Under certain conditions, data sets with the above characteristics will be deleted.

This checklist illustrates general job control procedures for compiler, linkage editor, and execution processing. More than one example may be used for a job step. The checklist is intended as an aid in preparing procedures, not as an inclusive list of the options and parameters.

COMPILATION

Figure 23 shows a general job control procedure for a compilation job step. The following cases show how to add to or modify the general procedure to obtain various processing options.

Case 1: Compile Only - No Object Module Produced

Use the general procedure. A listing is produced. It will include the default or specified options of the PARM parameter that affect output. Any diagnostic messages are listed, unless listing of warning messages is suppressed by the FLAGE option of the PARM parameter and only warning messages are produced.

Case 2: Source Module from Card Reader

Modify the end of the procedure as follows:

```
//SYSIN  DD  *
    (source module)
/*
```

If the DD * convention is used, the SYSIN DD statement must be the last DD statement for the job step, and the source module must follow. If another job step follows the compilation, the EXEC statement for that step follows the /* statement.

Case 3: Object Module is to be Punched

Add the statement:

```
//SYSPUNCH  DD  SYSOUT=B
```

Note: If DECK is not the installation default condition, it must be specified in the PARM parameter of the EXEC statement.

```
//jobname  JOB  acctno,name,MSGLEVEL=1
//stepname EXEC PGM=IEQCBL00,PARM=(options)
//SYSUT1   DD   UNIT=SYSDA,[SPACE=(subparms)]
//SYSUT2   DD   UNIT=SYSDA,(SPACE=subparms)
//SYSUT3   DD   UNIT=SYSDA,(SPACE=subparms)
//SYSUT4   DD   UNIT=SYSDA,(SPACE=subparms)
//SYSPRINT DD   SYSOUT=A
//SYSIN    DD   DSNAME=dsname,UNIT=SYSSQ,VOLUME=(subparms),DISP=(OLD,DELETE)
```

Figure 23. General Job Control Procedure for Compilation

Case 4: Object Module is to be Passed to Linkage Editor

Add the statement:

```
//SYSLIN     DD    DSNAME=(subparms),
                   UNIT=SYSDA,
                   SPACE=(subparms),
                   DISP=(MOD,PASS)
```

Note: If LOAD is not the installation default condition, it must be specified in the PARM parameter of the EXEC statement.

Case 5: Object Module is to be Saved

The object module can be saved by cataloging it, by keeping it, or by adding it as a member of a library. Add the SYSLIN statement as follows:

A. Cataloging

```
//SYSLIN  DD  DSNAME=dsname,
                DISP=({NEW},CATLG)
                      {MOD}      ,
                VOLUME=(subparms),
                LABEL=(subparms),
                UNIT={SYSDA}
                     {SYSSQ}

               {SPACE   }
               {SPLIT   } =(subparms)
               {SUBALLOC}
```

B. Keeping

```
//SYSLIN  DD  DSNAME=dsname,
                DISP=({NEW},KEEP)
                      {MOD}      ,
                VOLUME=(subparms),
                LABEL=(subparms),
                UNIT={SYSDA}
                     {SYSSQ}

               {SPACE   }
               {SPLIT   } =(subparms)
               {SUBALLOC}
```

C. Adding a member to an Existing Library

```
//SYSLIN  DD  DSNAME=dsname(member),      X
//              DISP=OLD
```

Case 6: COPY or INCLUDE Statements in COBOL Source Module or a BASIS Card in the Input Stream

Add the SYSLIB DD card(s), as follows:

A. COPY

```
//SYSLIB  DD  DSNAME=copylibname,DISP=OLD
```

B. BASIS

```
//SYSLIB  DD  DSNAME=basislibname,DISP=OLD
```

C. Both BASIS and COPY

```
//SYSLIB  DD  DSNAME=basislibname,DISP=OLD
//          DD  DSNAME=copylibname,DISP=OLD
```

(DD statements for additional copylibs may follow.)

Note: If BASIS or COPY are not the installation default options, they must be specified on the PARM parameter.

LINKAGE EDITOR

Figure 24 shows a general job control procedure for a linkage editor job step. The following cases show how to add to or modify the procedure to obtain various processing options.

Case 1: Input from Previous Compilation in Same Job

Change the SYSLIN statement to

```
//SYSLIN  DD  DSNAME=*.stepname.SYSLIN,    X
//              DISP=(OLD,DELETE)
```

where stepname is the name of the previous compilation job step and ddname is SYSLIN. If the input is to be saved, specify KEEP rather than DELETE.

Case 2: Input from Card Reader

Change the SYSLIN statement and the end of the procedure as follows:

```
//SYSLIN  DD *
    (object module(s))
/*
```

If the DD * convention is used, the SYSLIN DD statement must be the last DD statement in the job step. If another job step follows the linkage editor step, the EXEC statement for that job step follows the /* statement.

```
//jobname   JOB   acctno,name,MSGLEVEL=1
                .
                .
                .
//stepname  EXEC  PGM=IEWL,PARM=(options)
//SYSPRINT  DD    SYSOUT=A
//SYSLMOD   DD    DSNAME=&name(member),UNIT=SYSDA,DISP=(NEW,PASS),SPACE=(subparms)
//SYSLIB    DD    DSNAME=SYS1.COBLIB,DISP=OLD
//SYSUT1    DD    UNIT=SYSDA,SPACE=(subparms)
//SYSLIN    DD    DSNAME=dsname,DISP=OLD
```

Figure 24. General Job Control Procedure for a Linkage Editor Job Step

Case 3: Input Not from Compilation in Same Job

Specify in the SYSLIN DD statement where the object modules to be used as input are stored. (Only one member of a library can be specified in the SYSLIN DD statement.)

Case 4: Output to be Placed in Link Library

Change the SYSLMOD statement as follows:

```
//SYSLMOD  DD DSNAME=SYS1.LINKLIB(member),X
//            DISP=OLD
```

Member is the name of the load module that is to be added to the link library. No other information is needed in the statement.

Case 5: Output to be Placed in Private Library

Change the SYSLMOD statement as follows:

```
//SYSLMOD  DD  DSNAME=dsname(member),     X
//             DISP=OLD
```

Member is the name of the load module to be added, and dsname is the name of an existing library. If the library is not cataloged, UNIT and VOLUME parameters must be specified.

Note: See "Using the DD Statement" for an example of creating a new library and storing the load module as its first member.

Case 6: Output to be Used Only in this Job

Use the general procedure. The load module is stored in a temporary library.

EXECUTION TIME

Figure 25 shows a general job control procedure for an execution-time job step. The following cases show how to add to or modify the general procedure to obtain various processing options.

Case 1: Load Module to be Executed is in Link Library

Use the general procedure, where progname in the EXEC statement is the member name of the load module.

Case 2: Load Module to be Executed is a Member of Private Library

The JOBLIB statement must follow the JOB statement, as in the following statements:

```
//JOB1     JOB   ------

//JOBLIB   DD    DSNAME=MYLIB,              X
//               DISP=(OLD,PASS)
//STEP1    EXEC  PGM=PAYROLL
                .
                .
                .

//STEP2    EXEC  PGM=ACCOUNT
```

```
//stepname EXEC  PGM=progname
//ddname   DD    (parameters for user-specified data sets)
           .
           .
           .
```

Figure 25. General Job Control Procedure for an Execution-Time Job Step

The JOBLIB statement defines the private library MYLIB. No volume or unit parameters are given since the library is cataloged. Since JOBLIB has the disposition PASS, both steps can execute members of the library named in the JOBLIB statement. If only the first step executes a load module from the library, the disposition PASS on the JOBLIB statement need not be included.

Case 3: Load Module to be Executed is Created in Previous Linkage Editor Step in Same Job

Change the EXEC statement as follows:

```
//stepname  EXEC  PGM=*.stepname.SYSLMOD
```

where the stepname following PGM is the name of the linkage editor job step that created the load module.

Case 4: Abnormal Termination Dump

Add the statement:

```
//SYSABEND  DD  SYSOUT=A
```

This statement requests a full dump if abnormal termination occurs during execution.

Case 5: DISPLAY is Included in Source Module

Add the statement:

```
//SYSOUT  DD  SYSOUT=A
```

Case 6: DISPLAY UPON SYSPUNCH is Included in Source Module

Add the statement:

```
//SYSPUNCH  DD  SYSOUT=B
```

Case 7: ACCEPT is Included in Source Module

If the data is in the input stream, add the statement:

```
//SYSIN   DD  *
          (data)
/*
```

(See Case 2 under "General Job Control Procedures for a Compilation Job Step" for a discussion of the DD * convention.)

Case 8: Debug Statements EXHIBIT or TRACE are Included in Source Module

Use the statement (unless it is already included):

```
//SYSOUT  DD  SYSOUT=A
```

Note: If the job step already includes a SYSOUT DD statement for some other use, another need not be inserted.

A cataloged procedure is a set of job control statements that has been placed in a partitioned data set called the procedure library (SYS1.PROCLIB). It can be retrieved from the library by using its member name in an EXEC statement of a job step in the input stream. Frequently used procedures, such as those used for compiling and linkage editing, can be cataloged to simplify their subsequent use.

A cataloged procedure can contain statements for the processing of an entire job, or it can contain statements to process one or more steps of a job, with the remaining steps defined by job control statements in the input stream. A job can use several cataloged procedures, each processing one or more of the job steps. A job can also call for execution of the same cataloged procedure in more than one job step.

This chapter describes the following:

- How to call cataloged procedures
- The types of cataloged procedures, including those supplied by IBM for use with COBOL (F) source programs
- How to add procedures to the procedure library
- How to modify existing procedures for the current job step only
- How to override and add to cataloged procedures
- How to use the DDNAME parameter in cataloged procedures

CALLING CATALOGED PROCEDURES

A cataloged procedure is called by a job that appears in the input stream. The job must consist of a JOB statement and an EXEC statement that specifies the cataloged procedure name in the positional parameter (either procname or PROC=procname). For example:

```
//STEPQ EXEC COBFC
//STEPQ EXEC PROC=COBFC
```

Either of these EXEC statements could be used to call the IBM-supplied cataloged procedure COBFC to process the job step STEPQ.

A job step that calls for execution of a cataloged procedure can also contain DD statements that are applicable to the job steps of the cataloged procedure. A job that calls for execution of a cataloged procedure may, in other steps, call for execution of other cataloged procedures, call for other executions of the same cataloged procedure, or call directly for execution of load modules. The following example shows a job control procedure that calls both cataloged procedures and load modules.

```
//JOB1          JOB
//STEPA         EXEC   COBFC
//COB.SYSIN     DD     *
          (source module)
/*
//STEPL         EXEC   PGM=IEWL
                .
                .
                .
   (DD statements for the linkage editor)
                .
                .
                .
//STEPE         EXEC   PGM=*.STEPL.SYSLMOD
                .
                .
                .
   (DD statements for user-defined files)
                .
                .
                .
```

The IBM-supplied cataloged procedure COBFC for compilation is used to process STEPA. The COB.SYSIN DD statement is required to define the input to the compiler. The remaining statements in the procedure refer to execution of the linkage editor and the subsequent load module.

Data Sets Produced by Cataloged Procedures

Data sets produced during execution of a cataloged procedure can be used in subsequent job steps. They can also be called as follows:

```
//jobname    JOB  1234,J.SMITH
//STEPA      EXEC PROCED
//PROC1.SYSIN DD  *
         (source module)
/*
//stepname   EXEC PGM=*.STEPA.PROC2.SYSLMOD
              .
              .
              .
    (DD statements for user-defined files)
              .
              .
              .
```

The cataloged procedure PROCED is composed of two job steps, PROC1 and PROC2, that compile and linkage edit the source module.

TYPES OF CATALOGED PROCEDURES

The programmer can write his own procedures and catalog them, or he can use the three COBOL cataloged procedures provided by IBM.

PROGRAMMER-WRITTEN CATALOGED PROCEDURES

The programmer can write cataloged procedures, consisting of EXEC and DD statements, which incorporate job control procedures he uses frequently. For example, the programmer may wish to catalog an EXEC statement and the associated DD statements for a job step that specifies execution of a program. In this way the DD statements need not be specified each time the program is executed.

In writing a procedure for cataloging, the programmer must follow these rules:

- Another cataloged procedure cannot be referred to, i.e., only the PGM= progname form in an EXEC statement can be used.

  Note, however, that a cataloged procedure may contain a DD statement that refers to a cataloged data set.

- The following statements cannot be used in a cataloged procedure:

  1. The JOB statement
  2. A DD statement with JOBLIB in the name field
  3. A DD statement with an * in the operand field
  4. A DD statement with DATA in the operand field
  5. The delimiter statement

Adding Procedures to the Procedure Library

The IEBUPDTE utility program is used to add procedures to the procedure library. A description of the use of this program is given in the publication IBM System/360 Operating System: Utilities, Form C28-6586.

In Figure 26, two procedures are added to the procedure library (SYS1.PROCLIB). All control statements are in the input stream.

The first procedure is for a COBOL (F) compilation. Direct-access volumes are specified for the four utility data sets, and 100 tracks are allocated for each utility data set. This cataloged procedure is named COBDA.

The second procedure is also for a COBOL (F) compilation. Unlabeled tape volumes are specified for the four utility data sets. This cataloged procedure is named COBTP.

Job control statements: the EXEC card specifies that the IEBUPDTE program is to be executed, and the PARM=NEW is used because all data is read from one source, i.e., the input stream.

Utility statements: the ADD statement specifies the member name of the procedure, the level modification (00, first run) and the source of the modification (0, user-supplied). The NUMBER statement specifies the sequence numbers for records in the member. The first record of the cataloged procedure is numbered 00000010, and subsequent records are incremented by 00000010.

Note that leading zeros in the NUMBER statement are not necessary, as indicated in the example for the COBTP procedure.

IBM-SUPPLIED CATALOGED PROCEDURES

IBM distributes cataloged procedures with the operating system, which can be incorporated when the system is generated.

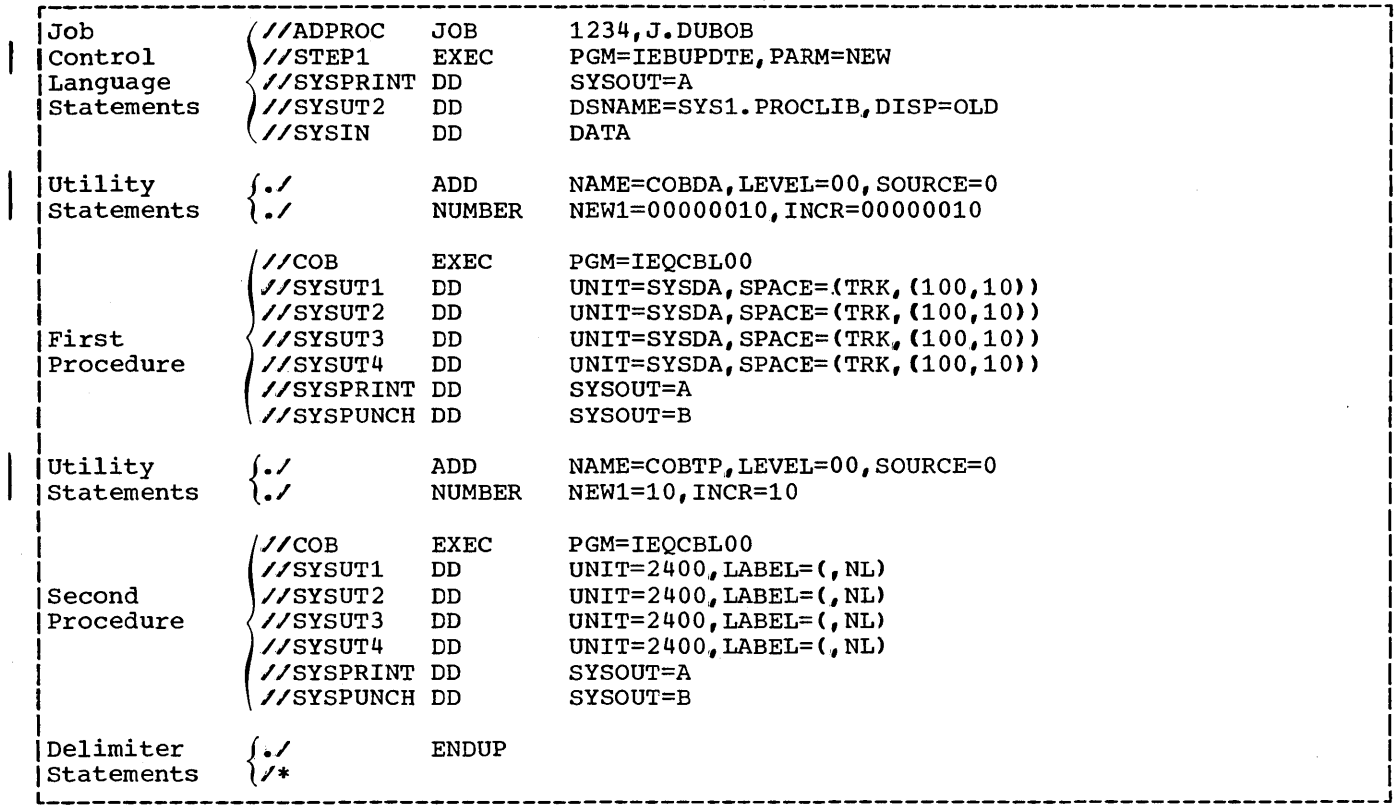


Figure 26. Example of Adding Procedures to the Procedure Library

Three of the procedures are for use with COBOL (F) programs:

1. COBFC providing compilation,
2. COBFLG providing linkage editing and execution, and
3. COBFCLG providing compilation, linkage editing, and execution.

These procedures may be used with any of the job schedulers released as part of the System/360 Operating System. When parameters required by a particular scheduler are encountered by another scheduler not requiring those parameters, either they are ignored or alternative parameters are substituted automatically. For example, if these procedures are used with a sequential scheduler, the following parameters, which are required for multiprogramming with a variable number of tasks (MVT), are treated as follows:

REGION=xxxxK is ignored

SYSOUT=B is interpreted as UNIT=SYSCP

DISP=SHR is interpreted as DISP=(OLD,KEEP)

These three cataloged procedures are shown in Figures 27, 28, and 29. (Space allocations in these procedures, are in terms of record lengths on the 2311 disk storage device.) Note that when DSNAME=&name is used in a DD statement, the specified data set is given a unique name by the operating system, and it is assumed to be a temporary data set which will be deleted when the job is completed. If the data set is to be kept, the DD statement can be overridden with a permanent data set name, and the appropriate parameters can be specified.

Note: If the compiler options are not explicitly supplied with the procedure, default options established at the installation apply. The programmer can override these default options by using an EXEC statement that includes the desired options (see "Overriding and Adding to EXEC Statements" later in this chapter).

Procedure Naming Conventions

Procedure names begin with the abbreviated name of the processor program, which, in the case of the COBOL procedures, is COB.

The processor's abbreviated name is followed by the level indicator (here F) and then by C, L, G or any combination of them. The C indicates compile, the L linkage edit, and the G go (i.e., execute). Hence, procedure COBFC is a single-step procedure which compiles a program using the COBOL (F) processor; COBFCLG is a three-step procedure wherein the first step compiles a program using COBOL (F), the second step linkage edits the output of the first step, and the third step executes the output of the linkage editor.

Step Names in Procedures

In a cataloged procedure, the step name is the same as the abbreviated processor name (LKED). The step that executes a compiled and linkage edited program is named GO.

For example, in the procedure named COBFCLG, the first step is named COB, the second step is named LKED, and the third step is named GO.

Unit Names in Procedures

The two unit names used in IBM-supplied cataloged procedures are as follows:

SYSSQ	any magnetic tape or direct-access device
SYSDA	any direct-access device

A pool of units must be assigned to these unit names during the system generation procedure. For example, only 2311 Disk Storage Drives might be assigned to the SYSSQ name. Then again, both 2400 Magnetic Tape Units and 2311 Disk Storage Drives might be assigned to the SYSSQ name. Once a pool of devices is assigned to these classes, device selection is done by the Job Scheduler.

Data Set Names in Procedures

When DSNAME=&name is used in a DD statement, the specified data set is given a unique name by the scheduler, and it is assumed to be a temporary data set that will be deleted when the job terminates. If the data set is to be retained, the DD statement must be overridden with a permanent data set name and appropriate DISP parameters.

COBFC Procedure

The COBFC procedure is a single-step procedure to execute the COBOL (F) compiler. It produces a punched object deck. Figure 27 shows the statements that make up the COBFC cataloged procedure.

The following DD statement must be supplied in the input stream:

```
//COB.SYSIN DD *    (or appropriate
                    parameters defining an
                    input data set)
```

If the DD * statement is used under PCP the delimiter statement (/*) must follow the source module. Under MVT and MFT, the (/*) statement is not required.

COBFLG Procedure

The COBFLG cataloged procedure is a two-step procedure to linkage edit and execute the output of a COBOL (F) compilation. Figure 28 shows the statements that make up the procedure.

The following DD statement indicating the location of the object module must be supplied in the input stream:

```
//LKED.SYSIN  DD    *( or appropriate
                    parameters)
```

If the COBOL program refers to SYSIN in the execution step, the following DD statement must also be supplied and must be the last of the //GO, cards.

```
//GO.SYSIN  DD      *( or appropriate
                    parameters)
```

If the COBOL program refers to other data sets in the execution step such as user-defined files, DD statements that define these data sets must also be provided.

COBFCLG Procedure

The COBFCLG procedure is a three-step procedure to compile, linkage edit, and execute using the COBOL (F) compiler. Figure 29 shows the statements that make up the procedure.

```
//COB      EXEC PGM=IEQCBL00,PARM='DECK,NOLOAD',REGION=86K
//SYSPRINT DD  SYSOUT=A
//SYSPUNCH DD  SYSOUT=B
//SYSUT1   DD  UNIT=SYSDA,SPACE=(460,(700,100))
//SYSUT2   DD  UNIT=SYSDA,SPACE=(460,(700,100))
//SYSUT3   DD  UNIT=SYSDA,SPACE=(460,(700,100))
//SYSUT4   DD  UNIT=SYSDA,SPACE=(460,(700,100))
```

Figure 27. Statements in the COBFC Procedure

```
//LKED     EXEC PGM=IEWL,PARM='LIST,XREF,LET',REGION=96K
//SYSLIN   DD  DDNAME=SYSIN
//SYSLMOD  DD  DSNAME=&GODATA(RUN),DISP=(NEW,PASS),UNIT=SYSDA,          X
//            SPACE=(1024,(50,20,1))
//SYSLIB   DD  DSNAME=SYS1.COBLIB,DISP=SHR
//SYSUT1   DD  UNIT=(SYSDA,SEP=(SYSLIN,SYSLMOD)),SPACE=(1024,(50,20))
//SYSPRINT DD  SYSOUT=A
//GO       EXEC PGM=*.LKED.SYSLMOD,COND=(5,LT,LKED)
//SYSABEND DD  SYSOUT=A
```

Figure 28. Statements in the COBFLG Procedure

```
//COB      EXEC PGM=IEQCBL00,REGION=86K
//SYSPRINT DD  SYSOUT=A
//SYSUT1   DD  UNIT=SYSDA,SPACE=(460,(700,100))
//SYSUT2   DD  UNIT=SYSDA,SPACE=(460,(700,100))
//SYSUT3   DD  UNIT=SYSDA,SPACE=(460,(700,100))
//SYSUT4   DD  UNIT=SYSDA,SPACE=(460,(700,100))
//SYSLIN   DD  DSNAME=&LOADSET,DISP=(MOD,PASS),UNIT=SYSDA,              X
//            SPACE=(80,(500,100))
//LKED  EXEC  PGM=IEWL,PARM='LIST,XREF,LET',COND=(5,LT,COB),REGION=96K
//SYSLIN   DD  DSNAME=&LOADSET,DISP=(OLD,DELETE)
//            DD  DDNAME=SYSIN
//SYSLMOD  DD  DSNAME=&GODATA(RUN),DISP=(NEW,PASS),UNIT=SYSDA,          X
//            SPACE=(1024,(50,20,1))
//SYSLIB   DD  DSNAME=SYS1.COBLIB,DISP=SHR
//SYSUT1   DD  UNIT=(SYSDA,SEP=(SYSLIN,SYSLMOD)),SPACE=(1024,(50,20))
//SYSPRINT DD  SYSOUT=A
//GO       EXEC PGM=*.LKED.SYSLMOD,COND=((5,LT,COB),(5,LT,LKED))
//SYSABEND DD  SYSOUT=A
```

Figure 29. Statements in the COBFCLG Procedure

The COB job step produces an object module which is input to the linkage editor. Other object modules may be added as illustrated in Example 5 which appears later in this chapter in "Using the DDNAME Parameter."

The following DD statement indicating the location of the source module must be supplied in the input stream:

```
//COB.SYSIN  DD     *( or appropriate
                    parameters)
```

If the COBOL program refers to SYSIN, the following DD statement indicating the location of the input data set must also be supplied:

```
//GO.SYSIN   DD     *( or appropriate
                    parameters)
```

If the COBOL program refers to other data sets, DD statements that define these data sets must also be supplied.

MODIFYING EXISTING CATALOGED PROCEDURES

Existing cataloged procedures can be permanently modified by using the IEBUPDTE utility program described in the publication IBM System/360 Operating System: Utilities, Form C28-6586.

OVERRIDING AND ADDING TO CATALOGED PROCEDURES

Any parameter in a cataloged procedure except the PGM=progname parameter in the EXEC statement can be overridden. Parameters or statements not specified in the procedure can also be added. When a cataloged procedure is overridden or added to, the changes apply only during one execution.

OVERRIDING AND ADDING TO EXEC STATEMENTS

An EXEC statement can be overridden or added to in one of two ways:

1. Specify, in the operand field of the EXEC statement calling the procedure, the keyword, the procedure step-name and the subparameters, as follows:

   COND.procstep=(subparameters)

   If a multi-step procedure is being modified, parameters in the calling EXEC statement must be specified step by step; i.e., the parameters for one step must be specified before those of the next step. If the return code of a cataloged procedure step is to be tested, the name of the step in the procedure (procstep) must be qualified by the name of the step that called for execution of the cataloged procedure (stepname).

2. Specify in the operand field of the EXEC statement calling the procedure only the keywords and subparameters, as follows:

COND=(subparameters)
   If a multi-step procedure is being called, the specified parameters (with the exception of PARM) apply to all steps in the procedure. The PARM keyword parameter overrides the first EXEC statement and nullifies any subsequent PARM keyword parameters. The COND and ACCT parameters apply to all steps in the procedure. To override PARM parameters in job steps other than the first, the previous method can be used.

Note: A parameter in an EXEC statement cannot be partly overridden; it must be overridden in its entirety. Any parameter not overridden remains as originally defined.

Examples of Overriding and Adding to EXEC Statements

This section contains examples of overriding and adding to the EXEC statement. The procedures overridden or added to are the IBM procedures shown in Figures 27, 28, and 29.

Example 1: The following example shows the overriding of one parameter in the EXEC statement of the one procedure step in the IBM-supplied COBFC procedure. The statements appear in the input stream as follows (however, note that in actual use the PARM.COB parameter cannot be continued in this manner):

```
//jobname      JOB  1234,J.SMITH
//STEPA        EXEC COBFC,PARM.COB='DECK, X
//                  NOLOAD,BUF=40000'
//COB.SYSIN    DD   *
          (source module)
/*
```

Note: In the PARM parameter being overridden, the DECK and NOLOAD options were specified. They are included again since the parameter must be overridden in its entirety. The information is enclosed in single quotation marks, since subparameters that contain equal signs must be enclosed in this manner.

Example 2: The following example shows the overriding of two parameters and the adding of another in the EXEC statement of one procedure step of the IBM-supplied COBFCLG procedure. The statements appear in the input stream as shown (however, note that in actual use the COND.LKED and PARM.LKED parameters cannot be continued in this manner):

```
//jobname    JOB  123,J.SMITH
//STEPA      EXEC COBFCLG,COND.LKED=       X
//                (9,LT,STEPA.COB),        X
//                PARM.LKED=               X
//                (MAP,LIST),ACCT=(1234)
//COB.SYSIN  DD   *
          (source module)
/*
```

Note: For the linkage editor job step in the above example, the COND and PARM parameters have been overridden and the ACCT parameter added.

Example 3: The following example shows the overriding of individual parameters in more than one procedure step of the IBM-supplied COBFCLG procedure. The statements appear in the input stream as shown (however, note that in actual use the COND.GO statement cannot be continued in this manner):

```
//jobname    JOB  1234,J.SMITH
//stepname   EXEC COBFCLG,PARM.LKED=XREF, X
//                COND.GO=((5,EQ,COB),    X
//                (5,EQ,LKED))
//COB.SYSIN  DD   *
          (source module)
/*
```

The PARM option XREF replaces the PARM subparameters of the linkage editor job step. The COND option EQ (equal to) replaces the option LT (less than) in the execution job step.

Note that all overriding parameters for one step of the procedure must be specified before those for the next step.

Example 4: The following example shows the overriding of parameters on all EXEC statements in the IBM-supplied COBCLG procedure. The statements appear in the input stream as shown:

```
//jobname    JOB   1234,J.SMITH
//stepname   EXEC  COBCLG,PARM=(LOAD,MAP),X
//                 COND=(3,LT)            X
//                 ACCT=(123456,DEPTQ)
//COB.SYSIN  DD    *
          (source module)
/*
```

The PARM options are added to procedure step COB and nullify the PARM options in the LKED and GO steps. The COND and ACCT parameters apply to all steps in the procedure.

OVERRIDING AND ADDING TO DD STATEMENTS

A DD statement can be overridden or added to by using a DD statement whose name is composed of the procedure step-name which qualifies the ddame of the DD statement being overridden, as follows:

```
//procstep.ddname  DD  (appropriate
                       parameters)
```

Entire DD statements can also be added.

There are rules that must be followed when overriding or adding a DD statement within a step in a procedure.

- Overriding DD statements must be in the same order in the input stream as they are in the cataloged procedure.
- DD statements to be added must follow overriding DD statements.
- A DD statement with an * in the operand field terminates processing of subsequent DD statements in both the procedure and the input stream for the job step, but not necessarily for the job.

There are some special cases that should be kept in mind when overriding a DD statement.

- All parameters are overridden in their entirety, except for the DCB parameter. Within the DCB parameter, individual subparameters may be overridden.
- To nullify a keyword parameter (except the DCB parameter), write, in the overriding DD statement, the keyword, and an equal sign followed by a comma. For example, to nullify the use of the UNIT parameter, specify UNIT=, in the overriding DD statement.
- A parameter can be nullified by specifying a mutually exclusive parameter. For example, the SPACE parameter can be nullified by specifying the SPLIT parameter in the overriding DD statement.
- The DUMMY parameter can be nullified by omitting it and specifying the DSNAME parameter in the overriding DD statement.
- To override DD statements in a concatenation of data sets, one DD statement must be provided for each one in the concatenation. The first must be named, but all subsequent DD statements in the concatenation must not be named. If no change is to be made to a DD statement placed before one that is to be changed, the first overriding statement(s) should have a blank operand.
- If the DDNAME=ddname parameter is specified in a cataloged procedure, it cannot be overridden; rather it can refer to a DD statement supplied at the time of execution.

Examples of Overriding and Adding to DD Statements

This section contains examples of overriding and adding to parameters in DD statements. The procedures overridden or added to are the IBM procedures shown in Figures 27, 28, 29.

The DDNAME parameter is not used in these examples, although it can be useful with the cataloged procedures. The use of the DDNAME parameter is described in detail later in this chapter.

Example 1: The following example shows the overriding of DD statements in the IBM-supplied COBFCLG procedure.

```
//jobname       JOB  1234,J.SMITH
//stepname      EXEC COBFCLG
//COB.SYSLIN    DD   DSNAME=GOFILE
//COB.SYSIN     DD   *
           (source module)
/*
//LKED.SYSLIN   DD   DSNAME=*.COB.SYSLIN,   X
//                   DISP=(OLD,CATLG)
                .
                .
                .
/*

(other DD statements for user-defined
files)
                .
                .
                .
/*
```

The name of the data set in SYSLIN in the procedure step COB is changed to GOFILE. The name of the data set of SYSLIN in the procedure step LKED is changed to a reference to the SYSLIN DD statement in the COB procedure step, and the data set name GOFILE is cataloged.

Example 2: The following example shows the adding of DD statements to the IBM-supplied COBFCLG procedure. Note that if the statement DD * or the statement DD DATA is used, it must be the last to appear in a series of DD statements.

```
//jobname    JOB  1234,J.SMITH
//stepname   EXEC COBFCLG,
//           PARM.COB=(DECK,LOAD,PMAP,COPY)
//COB.SYSPUNCH DD   SYSOUT=B

//COB.SYSLIB    DD   DSNAME=USERLIB,DISP=OLD
//COB.SYSIN     DD   *
             (source module)
/*
//GO.TRANSACT   DD   DSNAME=JUNE,DISP=OLD
                .
                .
                .

(other DD statements for user-defined
files)
                .
                .
                .
/*
```

Note: In the above example SYSLIB and TRANSACT are cataloged data sets. When a data set is cataloged, it is sufficient to refer to it by DSNAME and DISP=OLD.

The PARM.COB option DECK and the SYSPUNCH DD statement are added to obtain a punched object module. The PARM option COPY and the SYSLIB DD statement are added because the source module contains a COPY statement. The PARM option PMAP is added to obtain a listing of the assembler language expansion of the source module.

Example 3: The following example shows overriding and adding to DD statements at the same time in the IBM-supplied COBFC procedure. Note that overriding statements must be in the same order as they appear in the procedure and must precede those statements being added.

```
//jobname       JOB  1234,J.SMITH
//stepname      EXEC COBFC
//COB.SYSUT2    DD   SPACE=,UNIT=SYSSQ,
//COB.SYSLIN    DD   DSNAME=&GOFILE,        X
//                     DISP=(MOD,PASS),     X
//                     UNIT=SYSSQ
//COB.SYSIN     DD   *
           (source module)
/*
      (subsequent job steps)
                .
                .
                .
```

The device class on the COB.SYSUT2 DD statement is changed to SYSSQ, and the SPACE parameter is nullified. Therefore, direct-access devices cannot be allocated. Any tape volumes to be assigned must have standard labels. The COB.SYSLIN DD statement is changed so that it passes the object module to subsequent job steps.

Example 4: The following example shows how to concatenate a data set with a data set defined in the COBFLG procedure.

```
//jobname       JOB   1234,J.SMITH
//stepname      EXEC  COBFLG
                .
                .
                .
//LKED.SYSLIB   DD    [blank operand field]
//              DD    [parameters]
                .
                .
                .
/*
```

Instead of the blank operand field, parameters could have been used to override the SYSLIB statement; the data set defined by the unnamed DD statement would then be concatenated to the data set that was redefined by overriding.

Note that any number of libraries could be concatenated to the SYSLIB data set. For example:

```
//LKED.SYSLIB  DD
//             DD   DSNAME=USERLIB,DISP=OLD
//             DD   DSNAME=MYLIB,DISP=OLD
```

USING THE DDNAME PARAMETER

The DDNAME parameter is used to define a dummy data set that can assume the characteristics of an actual data set, defined by a subsequent DD statement within the step. If a matching DD statement is found, its characteristics, with the exception of its ddname, replace those of the statement using the DDNAME parameter. If a matching DD statement is not found within the step, the data set defined by the DDNAME parameter remains a dummy.

This section contains examples showing the use of the DDNAME parameter with cataloged procedures.

The rules for using the DDNAME parameter are as follows:

- A backward reference (e.g., *.ddname) to a DD statement referred to by a DDNAME parameter cannot be used because the statement that is referred to loses its identity.
- A backward reference to a statement containing a DDNAME parameter can be used, but only after the statement to which the DDNAME parameter refers has been encountered. If a backward reference is used before the dummy data set (defined by DDNAME) has been given real characteristics, these real characteristics will not be transferred to the DD statement that contains the backward reference. For example, if DCB=*.ddname is used (where ddname is the name of a statement containing an unresolved DDNAME parameter), the DCB fields that are transferred are blank.
- Unnamed DD statements can be placed after a statement containing the DDNAME parameter (indicating concatenation), but unnamed DD statements cannot be placed after a statement referred to by a DDNAME parameter.
- The DDNAME parameter can be used five times in a step, but each DDNAME parameter must refer to a different statement.
- The DDNAME parameter cannot be used in a JOBLIB statement.

When using the DDNAME parameter, the following should also be kept in mind.

- The name of the DD statement referred to does not replace the name of the referencing statement.
- If a statement that contains the DDNAME parameter is overridden, it is nullified.
- If overriding is performed with a statement that contains the DDNAME parameter, all parameters in the overridden statement are nullified.

The following DD statements:

```
//S1         EXEC PGM=progname
//D1         DD   DDNAME=D3
//D2         DD   (parameters X,Y,Z)
//D3         DD   (parameters U,T,V)
```

will result in the same data definition produced by the following statements.

```
//S1         EXEC PGM=progname
//D1         DD   (parameters U,T,V)
//D2         DD   (parameters X,Y,Z)
```

EXAMPLES OF USING THE DDNAME PARAMETER

Example 1: The following example shows how to override the first DD statement in a cataloged procedure with a DD * statement, and allow subsequent statements to be processed. Without the DDNAME parameter, replacing the first DD statement with a DD * statement would terminate processing of subsequent statements in the job step. The cataloged procedure (PROC3) is as follows:

```
//STEP1      EXEC PGM=progname
//DD1        DD   (any parameters except *)
//DD2        DD   (any parameters except *)
```

The job procedure in which the overriding takes place appears in the input stream as follows:

```
//JOB1       JOB 1234,J.SMITH
//S1         EXEC PROC3
//STEP1.DD1  DD   DDNAME=D1
//D1         DD   *
```

The STEP1.DD1 statement overrides the DD1 statement; the DD2 statement is processed; then the D1 statement is processed.

Example 2: The following example shows how to override the first DD statement in a cataloged procedure with a DD * statement and how to add a DD statement. The cataloged procedure (PROC3) is as follows:

```
//STEP1     EXEC PGM=progname
//DD1       DD   (any parameters except *)
//DD2       DD   (any parameters except *)
```

The job procedure in which the overriding takes place appears in the input stream as follows:

```
//JOB2      JOB 1234,J.SMITH
//S1        EXEC PROC3
//STEP1.DD1 DD   DDNAME=DD4
//DD3       DD   (any parameters except *)
//DD4       DD   *
```

The DD4 statement effectively overrides the DD1 statement, after the DD2 statement has been processed and the DD3 statement has been added.

Example 3: The following example shows how to concatenate a data set in the input stream with a data set defined by a DD statement in a cataloged procedure. The cataloged procedure (PROC3) is as follows:

```
//STEP1     EXEC PGM=progname
//DD1       DD   (any parameters except *)
//DD2       DD   (any parameters except *)
```

The job procedure in which the concatenation takes place appears in the input stream as follows:

```
//JOB3      JOB 1234,J.SMITH
//S1        EXEC PROC3
//STEP1.DD1 DD   (blank operand field)
//          DD   DDNAME=DD3
//DD3       DD   *
```

The data set in the input stream is concatenated with the data set defined by the DD1 statement after the DD2 statement has been processed.

Example 4: The following example shows how to concatenate a data set in the input stream with a data set defined by a DD statement in a cataloged procedure and how to add a DD statement. The cataloged procedure (PROC3) is as follows:

```
//STEP1     EXEC PGM=progname
//DD1       DD   (any parameters except *)
//DD2       DD   (any parameters except *)
```

The job procedure in which the contatenation takes place appears in the input stream as follows:

```
//JOB4      JOB  1234, J.SMITH
//S1        EXEC PROC3
//STEP1.DD2 DD   (blank operand field)
//          DD   DDNAME=DD4
//DD3       DD   (any parameters except *)
//DD4       DD   *
```

Example 5: The following example shows how the statement DD DDNAME=SYSIN in the IBM-supplied COBFCLG procedure can be used to add more object modules as input to the linkage editor. The statements appear in the input stream as follows:

```
//jobname     JOB 1234,J.SMITH
//stepname    EXEC COBFCLG
                .
                .
                .
//COB.SYSIN DD      *
      (source deck)
/*
//LKED.SYSIN DD     *
      (first object module)
                .
                .
                .
      (last object module)
/*
(//GO. cards)
```

The COBFCLG procedure contains the following two statements in the linkage edit step:

```
//SYSLIN DD DSNAME=LOADSET,DISP=          X
            (OLD,DELETE)
//       DD DDNAME=SYSIN
```

The result of concatenating SYSIN with SYSLIN is that when SYSLIN (input to linkage editor) is read, SYSIN is also read and linked with it. For example, if IHDFDISP is one of the object modules in the SYSIN stream, it will be linked with SYSLIN. The IHDFDISP module from SYS1.COBLIB will not be used.

The Operating System/360 Checkpoint/Restart feature is designed to be used with programs running for an extended period of time when interruptions may halt processing before the end of the job. The feature is available in two operating system environments: The PCP (primary control program) and MVT (multiprogramming with a variable number of tasks).

The programmer may use the feature to offset expected interruptions caused by both internal and external causes. For instance, he may know that a high-priority job is about to be submitted and his program may be interrupted to permit that job to run. The feature allows the interrupted program to be restarted at the beginning of a job step or at a point within a job step. The feature consists of two routines: Checkpoint and Restart.

The Checkpoint routine is invoked by the COBOL load module containing the user's program. It moves information stored in registers and in main storage into a checkpoint record at user-designated points during execution of the program. The programmer specifies these points using the COBOL RERUN clause in the Environment Division.

The Restart routine restarts an interrupted program which has previously written a checkpoint record. The checkpoint record will contain all information necessary to restart the program. Restart can be initiated at any time after the program was interrupted; that is, it may be run immediately after the interrupt occurred, as an automatic restart, or at some later time convenient to the programmer, as a deferred restart.

The COBOL RERUN clause provides linkage to the system checkpoint routine. Hence, any cautions and restrictions on the use of the system checkpoint/restart feature also apply to the use of the RERUN clause.

The Checkpoint/Restart feature is fully described in the publication IBM System/360 Operating System: Supervisor and Data Management Services, Form C28-6646.

TAKING A CHECKPOINT

In order to initiate a checkpoint, the programmer uses Job Control statements and COBOL RERUN clauses. The programmer associates each RERUN clause with a particular COBOL file. The RERUN clause indicates that a checkpoint record is to be written onto a checkpoint data set whenever a specified number of records on that file have been processed. The programmer must define the checkpoint data set on a DD statement. The DD statement describes both a checkpoint data set and a checkpoint method.

CHECKPOINT METHODS

The programmer may elect to take single or multiple checkpoints, as described below:

Single: Only one checkpoint record exists at any given time. After the first checkpoint record is written, any succeeding checkpoint record overlays the previous one. This method is acceptable for most programs. It offers the advantage of saving space on the checkpoint data set and allows the programmer to restart his program at the latest checkpoint.

Multiple (multiple contiguous): Checkpoints are recorded and numbered sequentially. Each checkpoint is saved. This method is used when the programmer may wish to restart a program at a checkpoint other than at the latest one taken.

DD STATEMENT FORMATS

The programmer records checkpoints on tape or direct-access devices. Following are the DD formats to define checkpoint data sets. For Tape:

```
//ddname  DD   DSNAME=data-set-name,        X
//          VOLUME=SER=volser,              X
//          UNIT=deviceno,                  X
//          DISP=({NEW},KEEP),              X
//                {MOD}
//          DCB=(TRTCH=C),LABEL=(,NL)
```

For Direct Access:

```
//ddname  DD   DSNAME=data-set-name,        X
//     VOLUME=(PRIVATE,RETAIN,SER=volser),  X
//     UNIT=deviceno,DISP=(NEW,KEEP),       X
//     SPACE=(TRK,nnn)
```

where:

ddname
is the same as the external-name used in the COBOL RERUN clause to provide a link to the DD statement.

data-set-name
is the name given to each particular data set used to write checkpoint records. This name identifies the checkpoint data set at the Restart procedure. (See "Restarting a Program" in this section.)

volser
identifies the volume by serial number.

deviceno
identifies the device. For tape it indicates the device number for 7-track or 9-track tape. For direct access, it indicates the device number for disk or drum.

MOD
is specified for the multiple contiguous checkpoint method.

NEW
is specified for the single checkpoint method.

KEEP
is specified in order to prevent deletion of the data set at the conclusion of the job step.

nnn
is a decimal number indicating the number of tracks to be allocated to the checkpoint data set. The number of tracks required will depend on two factors.

1. whether single or multiple checkpoints are taken
2. the size of main storage (larger main storage requires more checkpoint records be written).

Note: The DCB parameter is necessary only for 7-track tape conversion; for 9-track tape it is not used.

Following are examples in defining checkpoint data sets.

To write single checkpoint records using tape:

```
//CHECKPT   DD DSNAME=CHECK1,                X
//          VOLUME=SER=ND003,                X
//          UNIT=2400,DISP=(NEW,KEEP),       X
//          LABEL=(,NL)
                .
                .
                .
 ENVIRONMENT DIVISION
                .
                .
                .
   RERUN ON 'CHECKPT' EVERY 500 RECORDS
    OF ACCT-FILE.
```

To write single checkpoint records using disk (note that more than one data set may share the same external-name):

```
//CHEK    DD  DSNAME=CHECK2,                 X
//    VOLUME=(PRIVATE,RETAIN,SER=111111)X
//    UNIT=2314,DISP=(NEW,KEEP),           X
//    SPACE=(TRK,300)
                .
                .
                .
 ENVIRONMENT DIVISION
                .
                .
                .
    RERUN ON 'CHEK' EVERY 200 RECORDS OF
      PAYCODE.
    RERUN ON 'CHEK' EVERY 300 RECORDS OF
      IN-FILE.
```

To write multiple contiguous checkpoint records (on tape):

```
//CHEKPT   DD DSNAME=CHECK3,                 X
//    VOLUME=SER=11111,                      X
//    UNIT=2400,DISP=(MOD,KEEP),           X
//    LABEL=(,NL)
                .
                .
                .
 ENVIRONMENT DIVISION
                .
                .
                .
    RERUN ON 'CHEKPT' EVERY 100 RECORDS
      OF PAY-FILE.
```

The system checkpoint routine advises the operator of the status of the checkpoints taken by displaying information messages on the console.

When a checkpoint has been successfully completed, the following message will be displayed:

IHJ004I jobname (ddname,unit,volser) CHKPT checkid

where checkid is the identification name of the checkpoint taken. Checkid is assigned by the control program as an 8-digit field. The first digit is the letter C, followed by a decimal number indicating the checkpoint. For example, checkid C0000004 indicates the fourth checkpoint taken in the job.

PLANNING CHECKPOINTS

The programmer should give some thought to planning his checkpoints so that they become an integral part in the overall efficiency of his program. Some considerations in deciding how best to implement the checkpoint facility follow.

1. Timing. Whenever a checkpoint is taken, a small delay is encountered in processing the problem program. Checkpoints taken too often will impact the processing time of a program. A programmer should decide whether he can afford to take checkpoints on a certain time schedule, e.g., every twenty or thirty minutes, or if he should take checkpoints more frequently.

2. Data reconstruction. In programs using direct-access files, changes to records will replace previous information; after processing, a record may bear little resemblance to the record it updated. Thus the programmer should be sure he can identify previously processed records. For example, assume that a direct-access device contains a user file consisting of loan records which are periodically updated to reflect interest due. If a checkpoint is taken and other additional records are updated and then an interruption occurs, the records updated after the last checkpoint will be updated again when the program is restarted, unless the programmer controls this condition. (He may set up a date field for each record and update the date each time the record is processed. Then, by investigating the date field after the restart, he can determine whether or not the record was previously updated.)

RESTARTING A PROGRAM

The system Restart routine retrieves the information recorded in a checkpoint record, restores the contents of main storage and all registers, repositions tape and direct-access devices, and then gives control to the COBOL load module. Restart can occur in one of two ways. It may occur automatically at the time an interruption stopped the program, or it may be initiated at a later time as a deferred restart. The type of restart is determined by the RD parameter of the job control language.

THE RD PARAMETER

The RD parameter may appear on either the JOB or the EXEC statement. If coded on the JOB statement, the parameter overrides any RD parameters on the EXEC statement.

If the programmer wishes to have his program restart automatically, he codes RD=R or RD=RNC. RD=R indicates that restart is to occur at the latest checkpoint. The programmer should specify the RERUN clause for at least one data set in his program in order to record checkpoints. If no checkpoint is taken prior to interruption, restart occurs at the beginning of the job step. RD=RNC suppresses the writing of a checkpoint record; thus any restart will occur at the beginning of the job step. RERUN clauses are unnecessary; if any are present, they are ignored.

If the programmer wishes to suppress automatic restart, he codes RD=NR or RD=NC. RD=NR suppresses automatic restart but allows a checkpoint record to be written, provided a RERUN clause has been specified. At restart time, the programmer may choose to restart his program at a checkpoint other than at the beginning of the job step. RD=NC suppresses both restart and checkpoints. In effect, the programmer is ignoring the features of the Checkpoint/Restart facility.

AUTOMATIC RESTART

Automatic restart occurs only at the latest checkpoint taken. (If no checkpoint was taken before interruption, automatic restart occurs at the beginning of the job step.)

In order to restart automatically, a program must request restart through use of the RD parameter, and must receive authorization from the operator. The system displays the following message to request authorization of the restart:

```
xxIEF225D SHOULD jobname.stepname.procstep
RESTART [checkid]
```

The operator will reply in the following form:

```
REPLY xx,'{YES|NO|HOLD}'
```

where YES authorizes restart, NO prevents restart, and HOLD defers restart until the operator issues a RELEASE command, at which time restart will occur. The HOLD option is applicable only in the MVT environment.

Whenever automatic restart is to occur, the system will reposition all devices except unit record devices. With the PCP environment, the operator should manually reload any input card deck into the card reader.

DEFERRED RESTART

Unlike an automatic restart, a deferred restart may occur at any checkpoint, not necessarily the latest one taken.

The programmer requests a deferred restart by means of the RESTART parameter on the JOB card and, if restart is to occur at a particular checkpoint within a job step, a SYSCHK DD statement to identify the checkpoint data set. The formats for these statements are as follows:

```
//jobname JOB ,MSGLEVEL=1,                    X
//            RESTART=(request,[checkid])
//SYSCHK  DD  DSNAME=data-set-name,           X
//            DISP=OLD,UNIT=deviceno,         X
//            DCB=(,RECFM=U,BLKSIZE=nnnn),X
//            VOLUME=SER=volser
```

where:

MSGLEVEL=1
: is required if restart is to occur in an MVT environment

RESTART=(request,[checkid])
: identifies the particular checkpoint at which restart is to occur. Request may take one of the following forms:

*	to indicate restart at the beginning of the job.
stepname	to indicate restart at the beginning of a job step.
stepname.procstep	to indicate restart at a procedure step within the job step.

checkid
: identifies the checkpoint where restart is to occur.

SYSCHK
: is the DDNAME used to identify a checkpoint data set to the control program. The SYSCHK DD statement must immediately precede the first EXEC statement of the resubmitted job.

data-set-name
: must be the same name that was used when the checkpoint was taken. It identifies the checkpoint data set.

deviceno and volser
: identify the device and volume containing the checkpoint data set.

nnnn
: is a decimal number identifying the block size in bytes. The number must not be smaller than 600 and not larger than 32,760.

As an example illustrating the use of these job control statements, a restart of the GO step of a COBFCLG procedure, at checkpoint identifier C0000003, might appear as follows:

```
//jobname   JOB  ,MSGLEVEL=1,                X
//                RESTART=(GO,C0000003)
//SYSCHK    DD   DSNAME=CHEKPT,              X
//                DISP=OLD,UNIT=2400,        X
//                VOLUME=SER=111111,         X
//                DCB=(,RECFM=U,             X
//                BLKSIZE=3625)
//procstep  EXEC COBFCLG
          .
          .
          .
    DD statements similar to original deck
```

The Restart routine uses information from DD statements in the resubmitted job to reset files for use after restart; therefore, care should be taken with any DD statements that may affect the execution of the restarted job step. Attention should be paid to the following:

- During the original execution, a data set meant to be deleted at the end of a job step should conditionally be defined as KEEP rather than DELETE in order to be available if an interruption forces a restart.
- At restart time, a data set created in the original execution (defined as NEW on a DD statement) should be defined as OLD or have a unique name assigned to it.
- At restart time, input data sets on cards should be positioned as they were at the time of the checkpoint. Cards which were read in before the checkpoint should not be read in again. Input data sets on tape or direct-

access devices will be automatically repositioned by the system.

- At restart time, the EXEC statement parameters PGM and COND and the DD statement parameters SUBALLOC and VOLUME=REF must not be used in steps following the restart step if they contain the form stepname or stepname.procstep referring to a step preceding the restart step.

When a deferred restart has been successfully completed, the system will display the following message on the console:

```
IHJ008I jobname RESTARTED
```

and gives control to the user's program.

THE CHECKPOINT MACRO INSTRUCTION (CHKPT)

A checkpoint is taken by using the operating system macro instruction, CHKPT. The format of the instruction used by COBOL is:

```
CHKPT BSAMDCB
```

The format of the data control block (DCB) used by COBOL to describe the checkpoint data set is:

```
BSAMDCB   DCB   DSORG=PS,MACRF=W,
  RECFM=U,DEVD=DA,DDNAME=DUMMY
```

COBOL allows the operating system checkpoint routines to open and close the checkpoint data set at each checkpoint. COBOL does not specify a BLKSIZE parameter in the DCB, therefore permitting a DD statement specification or, by default, the maximum size record allowed on the device. The ddname specified in the user's checkpoint data set DD statement is substituted for DDNAME=DUMMY.

The compiler, linkage editor, COBOL load module, and other system components can produce output in the form of printed listings, punched card decks, diagnostic or informative messages and data sets directed to tape or direct-access devices. This chapter describes the output listings which can be used to document and debug programs and the format of the output modules. The same COBOL program is used for each example. Appendix A shows the output formats in the context of a complete listing generated by a sample program.

COMPILER OUTPUT

The output of the compilation job step may include:

- A printed listing of the job control statements
- Device allocation messages from the job scheduler
- A printed listing of the statements contained in the source module
- A glossary of compiler-generated information about data
- A printed listing of the object code
- Compiler diagnostic messages
- System messages
- Disposition messages from the job scheduler
- An object module

Diagnostic messages associated with the compilation of the source program are automatically generated as output. The other forms of output may be requested in the PARM parameter in the EXEC statement. The level of diagnostic messages printed depends upon the FLAGW or FLAGE options.

All output to be listed is written on the device specified by the SYSPRINT DD statement. Line spacing of the source listing and the number of lines per page can be controlled by the SPACEn and LINECNT options.

Figure 30 contains the compiler output listing shown in Appendix A. Each type of output is numbered, and each format within each type is lettered. The text following the figure is an explanation of the figure.

```
(1)
//JOB3   JOB   PROGRAMMERNAME,MSGLEVEL=1
//STEP3  EXEC  PGM=IEQCBL00,PARM='BUF=13000,SIZE=160000,MAP'
//SYSUT1 DD    DSNAME=&UT1,UNIT=SYSDA,SPACE=(TRK,(100,10))
//SYSUT2 DD    DSNAME=&UT2,UNIT=SYSDA,SPACE=(TRK,(100,10))
//SYSUT3 DD    DSNAME=&UT3,UNIT=SYSDA,SPACE=(TRK,(100,10))
//SYSUT4 DD    DSNAME=&UT4,UNIT=SYSDA,SPACE=(TRK,(100,10))
//SYSLIN DD    DSNAME=PUNCH,UNIT=SYSDA,SPACE=(TRK,(100,10)),          X
//             DISP=(NEW,PASS)
//SYSPRINT DD  SYSOUT=A
//SYSIN  DD    *
(2)
IEF236I ALLOC. FOR JOB3     STEP3
IEF237I SYSUT1   ON 190
IEF237I SYSUT2   ON 190
IEF237I SYSUT3   ON 190
IEF237I SYSUT4   ON 190
IEF237I SYSLIN   ON 190
IEF237I SYSIN    ON 00C
```

Figure 30. Examples of Compiler Output (Part 1 of 3)

(3)

```
00018    100180 DATA DIVISION.
00019    100190 FILE SECTION.
00020    100200 FD  FILE-1
00021    100210      LABEL RECORDS ARE OMITTED
00022    100220      BLOCK CONTAINS 100 CHARACTERS
00023    100230      RECORDING MODE IS F
00024    100240      DATA RECORD IS RECORD-1.
00025    100250   01 RECORD-1.
00026    100260      02 FIELD-A  PICTURE IS X(20).

00053    100530 PROCEDURE DIVISION.
00054    100540 BEGIN. READY TRACE.
00055    100550      NOTE THAT THE FOLLOWING OPENS THE OUTPUT FILE TO BE CREATED
00056    100560      AND INITIALIZES COUNTERS.

00057    100570 STEP-1. OPEN OUTPUT FILE-1. MOVE ZERO TO COUNT, NUMBER.
00058    100580      NOTE THAT THE FOLLOWING CREATES INTERNALLY THE RECORDS TO BE
00059    100590      CONTAINED IN THE FILE, WRITES THEM ON TAPE, AND DISPLAYS
00060    100600      THEM ON THE CONSOLE.
00061    100610 STEP-2. ADD 1 TO COUNT, ADD 1 TO NUMBER, MOVE ALPHA (COUNT) TO

00077    100770 STEP-8. CLOSE FILE-2 .
00078    100780          STOP RUN.
```

(4)

```
   (A)          (B)  (C)          (D)      (E)     (F)          (G)          (H)        (I)
INTRNL NAME    LVL SOURCE NAME  BASE     DISPL   INTRNL NAME  DEFINITION   USAGE      R  O  Q
DNM=1-146      FD  FILE-1       DCB=01           DNM=1-146                 QSAM
DNM=1-164      01  RECORD-1     BLI=1     000    DNM=1-164    DS 0CL20     GROUP
DNM=1-184      02  FIELD-A      BLI=1     000    DNM=1-184    DS 20C       DISP
```

(5)

```
        MEMORY MAP
   (A)
   TGT                              00160

SAVE AREA                           00160
SWITCH CELL                         001A8
TALLY CELL                          001AC
UNUSED                              001B0
   (B)
LITERAL POOL (HEX)

00358 (LIT+0)      00000001  001AF0E9  C0000000

      DISPLAY LITERALS (BCD)

00364 (LTL+12)    'WORK-RECORD'
   (C)
   PGT                              00318

OVERFLOW CELLS                      00318
VIRTUAL CELLS                       00318
PROCEDURE NAME CELLS                0031C
GENERATED NAME CELLS                00330
DCB ADDRESS CELLS                   00348
VNI CELLS                           00350
LITERALS                            00358
DISPLAY LITERALS                    00364
```

(6)

```
(A)    (B)         (C)       (D)                  (E)      (F)                              (G)
57     *STEP-1
                   000384    58 F0 C 000                   L     15,000(0,12)               V(IHDFDISP)
                   000388    05 1F                         BALR  1,15
                   00038A    000140                        DC    X'000140'
                   00038D    05E2E3C5D760F1                DC    X'05E2E3C5D760F1'
57      OPEN       000394    58 10 C 030                   L     1,030(0,12)                DCB=1
                   000398    50 10 D 1AC                   ST    1,1AC(0,13)                PRM=1
                   00039C    92 8F D 1AC                   MVI   1AC(13),X'8F'              PRM=1
                   0003A0    41 10 D 1AC                   LA    1,1AC(0,13)                PRM=1
                   0003A4    0A 13                         SVC   19
                   0003A6    58 10 C 030                   L     1,030(0,12)                DCB=1
                   0003AA    58 F0 1 030                   L     15,030(0,1)
                   0003AE    05 EF                         BALR  14,15
                   0003B0    50 10 D 198                   ST    1,198(0,13)                BLI=1
                   0003B4    58 70 D 198                   L     7,198(0,13)                BLI=1
57      MOVE       0003B8    D2 01 6 000 C 040             MVC   000(2,6),040(12)           DNM=1-257    LIT+0
                   0003BE    D2 01 6 01C C 040             MVC   01C(2,6),040(12)           DNM=1-305    LIT+0
61     *STEP-2
                   0003C4                         PN=01    EQU   *
                   0003C4    58 F0 C 000                   L     15,000(0,12)               V(IHDFDISP)
```

Figure 30. Examples of Compiler Output (Part 2 of 3)

```
                 (A)        (B) (C)                    (D)
(7)  CARD   ERROR MESSAGE
     77     IEQ1080I-W     PERIOD PRECEDED BY SPACE. ASSUME END OF SENTENCE.


(8)  IEF285I   UT1.JOB3                                  DELETED
     IEF285I   VOL SER NOS= 111111.
     IEF285I   UT2.JOB3                                  DELETED
     IEF285I   VOL SER NOS= 111111.
     IEF285I   UT3.JOB3                                  DELETED
     IEF285I   VOL SER NOS= 111111.
     IEF285I   UT4.JOB3                                  DELETED
     IEF285I   VOL SER NOS= 111111.
     IEF285I   PUNCH                                     PASSED
     IEF285I   VOL SER NOS= 111111.
     IEF285I   SYSOUT                                    SYSOUT
     IEF285I   VOL SER NOS= SCRTCH.
```

Figure 30. Examples of Compiler Output (Part 3 of 3)

(1) The listing of the job control statements associated with this job step. These statements are listed because MSGLEVEL=1 is specified in the JOB statement.

(2) Allocation messages from the job scheduler. These messages provide information about the device allocation for the data sets in the job step. They appear after the job control statements in the compile, linkage editor, and execution job steps. For example:

```
IEF237I   SYSUT1 ON 190
```

indicates that the data set for SYSUT1 has been assigned to the device 190.

(3) The source module listing. The statements in the source module are listed exactly as submitted except that a compiler-generated card number is listed in the first column of each line. This number is referred to in a diagnostic message and in the object code listing. The source module is not listed when the NOSOURCE option is specified.

The following notations may appear on the listing:

C Denotes that the statement was inserted with a COPY or INCLUDE statement.

** Denotes that the card is out of sequence.

I Denotes that the card was inserted with an INSERT or BASIS card.

If DATE-COMPILED is specified in the Identification Division, any sentences in that paragraph are replaced in the listing by the date of compilation in the following format:

```
DATE-COMPILED.  month day year
```

(4) The glossary: The glossary is listed when the MAP or DMAP option is specified. The glossary contains information about names in the COBOL source program.

A and F. The internal name generated by the compiler. This name is used in the compiler object code listing to represent the name used in the source program.

B. A normalized level number. This level number is determined by the compiler as follows: the first level number of any hierarchy is always 01, and increments for other levels are always by one. Only level numbers 03 through 49 are affected; level numbers 77 and 88 and FD, SD, and RD indicators are not changed.

C. The data name that is used in the source module.

Note that the following Report Writer internally generated data-names can appear under the SOURCE NAME column:

CTL.LVL - Used to keep track of control levels.
GRP.IND - Used by coding for GROUP INDICATE clause.
TER.COD - Used by coding for TERMINATE.
FRS.GEN - Used by coding for GENERATE.
-nnnn - Generated report record associated with the file on which the report is to be printed.
RPT.RCD - Build area for print record.
CTL.CHR - First or second position of RPT.RCD. Used for carriage control character.
RPT.LIN - Beginning of actual information which will be displayed. Second or third position of RPT.RCD.
E.nnnn - Field within RPT.RCD. Overlays part of RPT.RCD, depending on the column clause.
S.nnnn - Used for elementary level with SUM clause, but not with data name.
N.nnnn - Used to save the total number of lines used by a report group when relative line numbering is specified.

D and E. For data names, these columns contain information about the address in the form of a base and displacement. For file names, the column contains information about the associated DCB and DECB, if any.

G. This column defines storage for each data item. It is represented in assembler-like terminology. Table 10 refers to information in this column.

H. Usage of the data name. For FD entries, the file organization is identified. For group items containing a USAGE clause, the information in the clause is identified. For group items not containing a USAGE clause, GROUP is identified. For elementary items, the information in the USAGE clause is identified.

I. An asterisk under column:

R-Indicates that the data-name redefines another data-name.
O-Indicates that an OCCURS clause has been specified for that data-name.
Q-Indicates that the data-name is or contains the DEPENDING ON object of the OCCURS clause.

(5) Global tables and literal pool: The Global table is listed when the MAP or PMAP option is specified unless SUPMAP was specified and an E-level diagnostic was generated. A global table contains easily addressable information needed by the object program for execution. For example, in the Procedure Division output coding (3) the address of the first instruction under STEP-1 (OPEN OUTPUT FILE-1) would be found in the PROCEDURE NAME CELLS portion of the Program Global Table (PGT).

A. The Task Global Table (TGT). This table consists of switches, addresses and work areas used by the in-line program. Subtask Global Tables (SGT) contain similar information used by the random processing procedures. One TGT exists for the entire program; one SGT exists for each random processing procedure.

C. The Program Global Table (PGT). This contains the remaining addresses and the literals used by the object program.

The literal pool (B) lists the collection of the literals in the program, with duplications removed. These literals include those specified by the programmer (e.g., MOVE 'ABC' TO DATA-NAME) and those generated by the compiler (e.g., to align decimal points in arithmetic computation). The literals are divided into two groups: those that are referred to by instructions (marked "LITERAL POOL") and those that are referred to by the calling sequences to object time subroutines (marked "DISPLAY LITERALS").

(6) The object code listing: The object code listing is produced when the MAP or PMAP option is specified. The actual object code listing contains:

A. The compiler-generated card number. The number refers to the COBOL statement in the source module which contains the verb that is listed under column B.

B. The COBOL procedure-name and the COBOL verb. The procedure-name is indicated by an asterisk.

The statement within which the COBOL verb appears determines the information under columns C, D, F, and G.

Table 10. Glossary Definition and Usage

Type	Definition	Usage
GROUP FIXED LENGTH	DS 0CLNC	GROUP
ALPHABETIC	DS NC	DISP
ALPHANUMERIC	DS NC	DISP
GROUP VARIABLE LENGTH	DS VLI=N	GROUP
REPORT	DS NC	RPT
STERLING REPORT	DS NC	RPT-ST
EXTERNAL DECIMAL	DS NC	DISP-NM
EXTERNAL FLOATING POINT	DS NC	DISP-FP
INTERNAL FLOATING POINT	DS 1F	COMP-1
	DS 1D	COMP-2
BINARY	DS 1H, 1F OR 2F	COMP
INTERNAL DECIMAL	DS NP	COMP-3
STERLING NON-REPORT	DS NC	DISP-ST
FILE (FD)	RPD=0	FILE PROCESSING TECHNIQUE
CONDITION (88)	BLANK	BLANK
REPORT DEFINITION (RD)	BLANK	BLANK
SORT DEFINITION (SD)	BLANK	BLANK
Under the definition column, N= size in bytes, except in group variable length where it is a variable length cell number.		

C. The relative location, in hexadecimal notation, of the object code instruction in the module.

D. The actual object code instruction in hexadecimal notation.

E. The procedure-name number. A number is assigned only to procedure-names referred to in other Procedure Division statements.

F. The object code instruction in the form that closely resembles assembler language (displacements are in hexadecimal notation).

G. Compiler-generated information about the operands of the generated instruction. This includes names and relative locations of literals. Table 11 refers to information in this column.

Note: The programmer can produce a condensed listing by specifying CLIST as an option in place of MAP or PMAP. The CLIST option produces only the source card number, the EBCDIC representation of the verb name, and the location of the first generated instruction, as follows:

```
54    READY   000380   57   OPEN    000394
57    MOVE    0003B8   61   ADD     000444
61    ADD     000450   61   MOVE    00045C
```

⑦ Diagnostic messages: The diagnostic messages associated with the compilation are always listed. The format of the diagnostic message is:

A. Compiler-generated line number. This is the number of a line in the source module related to the error.

B. Message identification. The message identification for COBOL (F) compiler diagnostic messages always begins with the symbols IEQ.

C. The severity level. There are four severity levels as follows:

W Warning - This level indicates that an error was made in the source program. However, it is not serious enough to hinder the execution of the program. These Warning messages are listed only if FLAGW is specified.

C Conditional - This level indicates that an error was made but the compiler usually makes a corrective assumption. The statement containing the error is retained. Execution can be attempted for the debugging value.

Table 11. Symbols Used in the Listing and Glossary to Define Compiler-Generated Information

Symbol	Meaning
DNM	SOURCE DATA NAME
SAV	SAVE AREA CELL
SWT	SWITCH CELL
TLY	TALLY CELL
WC	WORKING CELL
TS	TEMPORARY STORAGE CELL
SBI	SECONDARY BASE LOCATOR
BLI	BASE LOCATOR
BLL	BASE LOCATOR FOR LINKAGE SECTION
ON	ON COUNTER
PFM	PERFORM COUNTER
PSV	PERFORM SAVE
VN	VARIABLE PROCEDURE NAME
DEC	DECB ADDRESS
SBS	SUBSCRIPT ADDRESS
XSW	EXHIBIT SWITCH
XSA	EXHIBIT SAVE AREA
PRM	PARAMETER
PN	SOURCE PROCEDURE NAME
GN	GENERATED PROCEDURE NAME
DCB	DCB ADDRESS
VNI	VARIABLE NAME INITIALIZATION
LTL	LITERAL
TS2	TEMPORARY STORAGE (NON-ARITHMETIC)
RSV	REPORT SAVE AREA
SAV2	SAVE AREA 2
SAV3	SAVE AREA 3
V(BCD-N)	VIRTUAL

E Error - This level indicates that a serious error was made. Usually the compiler makes no corrective assumption. The statement or operand containing the error is dropped. Execution of the program should not be attempted.

D Disaster - This level indicates that a serious error was made. Compilation is not completed. Results are unpredictable.

There is a correlation between severity level and the return codes referred to by the COND parameter. For example, a compilation in which a D-level error is detected will generate a return code of 16.

D. The message text. The text identifies the condition that caused the error and indicates the action taken by the compiler.

Since Report Writer generates a number of internal data items and procedural statements, some error messages may reflect internal names. In cases where the error manifests itself mainly in these generated routines, the error messages may indicate the card number of the RD entry for the report under consideration. In addition, there are errors that may indicate the card number of the card upon which the statement containing the error ends rather than the card upon which the error occurred.

Internal name formats for Report Writer are discussed under Glossary.

A complete list of compiler diagnostic messages is contained in the last appendix in this publication.

⑧ Disposition messages from the job scheduler: These messages contain information about the disposition of the data sets, including volume serial numbers of volumes in which the data sets resides.

THE OBJECT MODULE

The object module contains the external symbol dictionary, the text of the program, and the relocation dictionary. It is followed by an END statement that marks the end of the module. For more detailed information about the external symbol dictionary, text, and relocation dictionary refer to the publication IBM System/360 Operating System: Linkage Editor, Form C28-6538.

An object module deck is punched if the DECK option is specified unless SUPMAP was specified and an E-level diagnostic was generated, and if a SYSPUNCH DD Statement is included. An object module is written in an output volume if the LOAD option is specified unless SUPMAP was specified and an E-level diagnostic was generated, and if a SYSLIN DD statement is included.

LINKAGE EDITOR OUTPUT

The output of the linkage editor job step may include:

- A printed listing of the job control statements
- A map of the load module after it has been processed by the linkage editor

- A cross reference list
- Informative messages
- Diagnostic messages
- Disposition messages
- A listing of the linkage editor control statements
- A load module which must be assigned to a library

Any diagnostic messages or informative messages associated with the linkage editor are automatically generated as output. The other forms of output may be requested by the PARM parameter in the EXEC statement. All output to be listed is written in the data set specified by the SYSPRINT DD statement.

Figure 31 is an example of linkage editor output listing. It shows the job control statements, informative messages, and module map. The different types of output are numbered and each type to be explained is lettered. The text following the figure is an explanation of the figure.

```
1) //STEP4  EXEC  PGM=IEWL,PARM='XREF'
   //SYSUT1 DD    DSNAME=&UT1,UNIT=SYSDA,SPACE=(TRK,(100,10))
   //SYSLMOD DD   DSNAME=GOJOB(GO),UNIT=SYSDA,SPACE=(TRK,(100,10,1)),     X
   //             DISP=(NEW,PASS)
   //SYSLIB DD    DSNAME=SYS1.COBLIB,DISP=OLD
   //SYSPRINT DD  SYSOUT=A
   //SYSLIN DD    DSNAME=PUNCH,UNIT=SYSDA,SPACE=(TRK,(100,10)),          X
   //             DISP=(OLD,DELETE)

2) IEF236I ALLOC. FOR JOB3     STEP4
   IEF237I SYSUT1   ON 190
   IEF237I SYSLMOD  ON 190
   IEF237I SYSLIB   ON 190
   IEF237I SYSLIN   ON 190

3)     E-LEVEL LINKAGE EDITOR OPTIONS SPECIFIED--XREF

4) IEW0000   GO        NOW ADDED TO DATA SET

                                        ----  CROSS REFERENCE TABLE  ----
          (A)                                  (B)
   CONTROL SECTION                      ENTRY
5)   NAME     ORIGIN  LENGTH              NAME   LOCATION     NAME   LOCATION     NAME   LOCATION     NAME   LOCATION

   TESTRUN       00     6AC
   IHDFDISP*    6B0     58C

          (A)          (B)              (C)
   LOCATION  REFERS TO SYMBOL  IN CONTROL SECTION
6)    330          IHDFDISP       IHDFDISP

          (C)
   ENTRY ADDRESS       00
5)      (D)
   TOTAL LENGTH       C3C

   IEF285I   UT1.JOB3                              DELETED
   IEF285I   VOL SER NOS= 111111.
   IEF285I   GOJOB                                 PASSED
   IEF285I   VOL SER NOS= 111111.
7) IEF285I   SYS1.COBLIB                           KEPT
   IEF285I   VOL SER NOS= 111111.
   IEF285I   SYSOUT                                SYSOUT
   IEF285I   VOL SER NOS= SCRTCH.
   IEF285I   PUNCH                                 DELETED
   IEF285I   VOL SER NOS= 111111.
```

Figure 31. Linkage Editor Output Showing Module Map and Cross Reference List

① The job control statements. These statements are listed because MSGLEVEL=1 is specified on the JOB statement for this job, shown in Figure 30.

② Allocation messages from the job scheduler. These messages provide information about the device allocation for the data sets in the job step. For example, the message

```
IEF2371   SYSUT1 ON 190
```

indicates that the data set for SYSUT1 has been assigned to the device 190.

③ Linkage editor informative message. This message lists the PARM options that were specified.

④ Linkage editor informative message. This is a disposition message describing the disposition of the load module.

A. Name of the load module specified in the DSNAME parameter of the SYSLMOD DD statement

B. Text of message

⑤ Module map. The module map is listed when either the XREF or the MAP option is specified in linkage editor processing. The module map shows all control sections in the output module and all entry names in each control section. The control sections are arranged in ascending order according to their assigned origins. All entry names are listed below the control section in which they are defined. Each COBOL program is a control section, and any COBOL library subroutine is a separate control section.

A. Control section. Under this heading the name, origin, and length of each control section is listed.
Name. The name of the control section. This name is the PROGRAM-ID name in the main COBOL program or a called program. Each control section that is obtained from a library by an automatic library call is indicated by an asterisk.
Origin. The relative origin in hexadecimal notation.
Length. The number of bytes in each control section in hexadecimal notation.

B. Entry. The entry names within each control section and their relative location. A called program may have more than one entry point. For a called COBOL program name, the entry point is the same as the name that is specified by the ENTRY statement in the source program.

C. Entry address. The relative address of the instruction with which processing of the module begins. It will always be INIT1 if the COBOL program is the main program of the load module.

D. Total length. The total number of bytes, in hexadecimal notation, of the load module. It is the sum of the lengths of all control sections.

⑥ The cross reference list. The cross reference list, as well as a module map, is listed if the XREF object is specified. MAP and XREF should not be specified together. The cross reference list provides the following information:

A. Location. The relative location in the program where another program is called.

B. Refers to symbol. The name of the entry point of the called program.

C. In control section. The control section that contains the entry point.

For example, 328 is the location where another program is called. IHDFDISP is the entry point of the called program. IHDFDISP is the control section that contains the entry point, IHDFDISP.

If XREF is specified, the cross reference list appears before the Entry Address.

⑦ Disposition messages from the job scheduler. These messages contain information about the disposition of the data sets.

Comments on the Module Map and Cross Reference List

The severity of linkage editor diagnostics may affect the production of module map and cross reference list.

Since various processing options will affect the structure of the load module, the text of the module map and cross reference list will sometimes provide additional information. For example, the load module may have an overlay structure. In this case, a module map will be listed for each segment in the overlay structure. The cross reference list is the same as that previously discussed, except that segment numbers also are listed to indicate the segment in which each symbol appears.

Listing the Linkage Editor Control Statements: If the LIST option is specified, linkage editor control statements, such as OVERLAY and LIBRARY, are listed.

Linkage Editor Messages

The linkage editor may generate informative or diagnostic messages. A complete list of these messages is included in the publication IBM System/360 Operating System Linkage Editor, Form C28-6538.

COBOL LOAD MODULE EXECUTION OUTPUT

The output generated by program execution (in addition to data written in program output files) can include:

- Data displayed on the console, or on the printer
- Punched data
- Messages to the operator
- System informative messages
- System diagnostic message
- A system dump

A dump and system diagnostic messages are generated automatically by program execution only if the program contains errors that cause abnormal termination.

Figure 32 shows an example of output from the execution job step. The following text is an explanation of the figure.

① The job control statements. These statements are listed because MSGLEVEL=1 is specified in the JOB statement for this job.

② The job allocation messages from the job scheduler. These messages indicate the device that is allocated for each data set defined for the job step.

③ Disposition messages from the job scheduler. These messages are contained in the publication IBM System/360 Operating System: Messages and Codes, Form C28-6631.

```
①  //STEP5  EXEC  PGM=*.STEP4.SYSLMOD
   //SYSOUT DD    SYSOUT=A
   //SYSABEND DD  SYSOUT=A
   //SAMPLE DD    UNIT=2400,LABEL=(,NL)

②  IEF236I ALLOC. FOR JOB3     STEP5
   IEF237I PGM=*.DD ON 190
   IEF237I SAMPLE   ON 282

③  IEF285I   GOJOB                                                  PASSED
   IEF285I   VOL SER NOS= 111111.
   IEF285I   SYSOUT                                                 SYSOUT
   IEF285I   VOL SER NOS=       .
   IEF285I   SYSOUT                                                 SYSOUT
   IEF285I   VOL SER NOS=       .
   IEF285I   AAAAAAAA.AAAAAAAA.AAAAAAAA.AAAAAAAA.00000048 DELETED
   IEF285I   VOL SER NOS= LGL001.
   IEF285I   GOJOB                                                  DELETED
   IEF285I   VOL SER NOS= 111111.
```

Figure 32. Execution Job Step Output

REQUESTS FOR OUTPUT

The programmer may request output in a number of ways:

1. He can request data to be displayed by using the DISPLAY statement.

2. Messages to the operator can also be displayed on the console when requested in the source program (DISPLAY UPON CONSOLE).

3. The programmer can request a full dump, in case his program is terminated abnormally, by including

   ```
   //SYSABEND   DD   SYSOUT=A
   ```

   in the job control procedure.

Note: Under MVT the SPACE parameter should also be included in the DD statement. For example:

```
//SYSABEND DD SYSOUT=A,                    X
              ,SPACE=(125,(200.1000)RLSE).
```

Dumps are explained in the chapter "Program Checkout."

OPERATOR MESSAGES

The COBOL load module may issue operator messages. In the messages, xx denotes a system-generated 2-character numeric field, which is used to identify the program issuing the message.

The following message is generated by STOP literal.

xx IEQ000A Text is provided by the object program.

Explanation: This message is issued at the programmer's discretion to indicate possible alternative action to be taken by the operator.

Operator Response: Follow the instructions given both by the message and on the job request form supplied by the programmer. If the job is to be resumed, issue a REPLY command with a text field that contains any 1-character message.

The following message is generated by an ACCEPT...FROM CONSOLE.

xx IEQ990D 'AWAITING REPLY'

Explanation: This message is issued by the object program when operator intervention is required.

Operator Response: Issue a REPLY command. (The contents of the text field should be supplied by the programmer on the job request form.)

xx IEQ999I NO DD CARD FOR DDNAME

Explanation: The operating system could not find a corresponding ddname on a DD statement for a file assigned in a SELECT clause.

Operator Response: User should have indicated either to continue processing or to cancel the job. Note that if the user elects to continue processing, any READ or WRITE encountered for the file will result in an ABEND.

SYSTEM OUTPUT

Informative and diagnostic messages may appear in the listing during execution of any job step. Further information about these messages is found in the publication IBM System/360 Operating System: Messages and Codes, Form C28-6631.

Each of these messages contains an identification code in the first three columns of the message to indicate the portion of the operating system that generated the message. Table 12 lists these codes, along with identification for each.

Table 12. System Message Identification Codes

Code	Identification
IEA	An on-line console message from the supervisor.
IEC	An on-line console message from data management.
IEF	A message from the job scheduler.
IEQ	A message from the COBOL compiler.
IER	A message from the Sort program.
IET	A message from the assembler.
IEW	A message from the linkage editor.
IHB	A message from the supervisor and data management.

A programmer using the COBOL (F) compiler under the System/360 Operating System has several methods available to him for testing his programs, debugging them, and revising them for increased efficiency in operation.

The COBOL debugging language can be used by itself or in conjunction with other COBOL statements. A dump can also be used for program checkout.

THE DEBUG LANGUAGE

The COBOL (F) debugging language is designed to aid the COBOL programmer in producing an error-free program in the shortest possible time. The sections that follow discuss the use of the debug language and other methods of program checkout.

The three debug language statements are TRACE, EXHIBIT, and ON. Any one of these statements can be used as often as necessary. They can be interspersed throughout a COBOL source program, or they can be in a packet in the input stream to the compiler.

Program checkout may not be desired after testing is completed. A debug packet can be removed after testing. This allows elimination of the extra object program coding generated for the debug statements.

The output produced by the TRACE and EXHIBIT statements is listed on the system logical output device (SYSOUT). If these statements are used, the SYSOUT DD statement must be specified in the execution time job step.

The following discussions describe ways to use the debug language.

FOLLOWING THE FLOW OF CONTROL

The READY TRACE statement causes each section and paragraph name to be listed on the system output unit when control passes to that point. The output appears as a list of unqualified procedure-names.

To reduce the number of names that are generated and the time taken to generate them, a trace can be stopped with a RESET TRACE statement. The READY TRACE/RESET TRACE combination is helpful in examining a particular area of the program. The READY TRACE statement can be coded so that the trace begins before control passes to that area. The RESET TRACE statement can be coded so that the trace stops when the program has passed the area. The two trace statements can be used together where the flow of control is difficult to determine, e.g., with a series of PERFORM statements or with nested conditionals.

Another way to control the amount of tracing, so that it is done conditionally, is to use the ON statement with the TRACE statement. When the COBOL compiler encounters an ON statement, it sets up a mechanism such as a counter which is incremented, during execution, whenever control passes through the ON statement. For example, if an error occurs when a specific record is processed, the ON statement can be used to isolate the problem record. The statement should be placed where control passes only once for each record that is read. When the contents of the counter equal the number of the record (as specified in the ON statement), a trace can be taken on that record. The following example shows a way in which the 200th record could be selected for a TRACE statement.

```
Col.
1       8
        RD-REC
        .
        .
        .
*DEBUG RD-REC       TRY
        PARA-NM-1.     ON 200 READY TRACE.
                       ON 201 RESET TRACE.
```

If the TRACE statement were used without the ON statement, every record would be traced.

An example of a common program error is failing to break a loop or unintentionally creating a loop in the program. If many iterations of the loop are required before it can be determined that there is a program error, the ON statement can be used to initiate a trace only after the expected number of iterations has been completed.

Note: If an error occurs in an ON statement, the diagnostic may refer to the previous statement number.

DISPLAYING DATA VALUES DURING EXECUTION

A programmer can display the value of a data item during program execution by using the EXHIBIT statement. The three forms of this statement display (1) the names and values of the data-names listed in the EXHIBIT statement (EXHIBIT NAMED) whenever the statement is encountered during execution; (2) the values of the data-names listed in this statement only if the value has changed since the last execution (EXHIBIT CHANGED); and (3) the data-names listed in the statement and the value of the data-names only if the values have changed since the previous execution (EXHIBIT CHANGED NAMED).

Note: The total length of all items displayed with EXHIBIT CHANGED cannot exceed 32,767 bytes.

Data can be used to check the accuracy of the program. For example, the programmer can display specified fields from records, work the calculations himself, and compare his calculations with the output from his program. The coding for a payroll problem could be:

```
Col.
1       8
        .
        .
        .
       GROSS-PAY-CALC.
           COMPUTE GROSS-PAY =
           RATE-PER-HOUR * (HRSWKD
           + 1.5 * OVERTIMEHRS).
       NET-PAY-CALC.
           .
           .
           .
*DEBUG NET-PAY-CALC.
       SAMPLE-1. ON 10 AND
           EVERY 10 EXHIBIT NAMED
           RATE-PER-HOUR, HRSWKD,
           OVERTIMEHRS, GROSS-PAY.
```

This coding will cause the values of the four fields to be listed for every tenth data record before net pay calculations are made. The output could appear as:

```
RATE-PER-HOUR = 4.00 HRSWKD = 40.0
     OVERTIMEHRS = 0.0 GROSS-PAY = 160.00

RATE-PER-HOUR = 4.10 HRSWKD = 40.0
     OVERTIMEHRS = 1.5 GROSS-PAY = 173.23

RATE-PER-HOUR = 3.35 HRSWKD = 40.0
     OVERTIMEHRS = 0.0 GROSS-PAY = 134.00
```

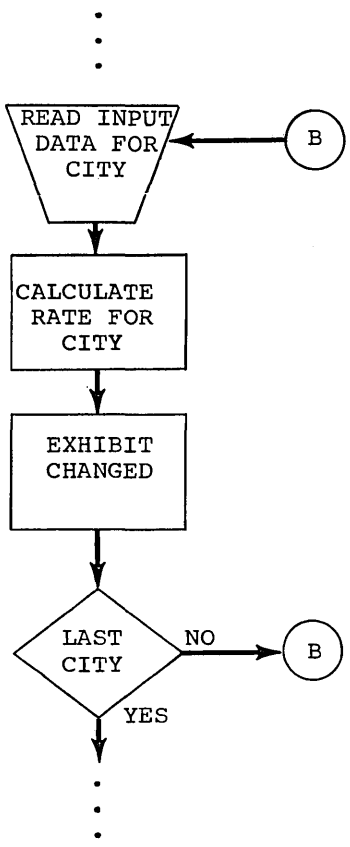
Note: Decimal points are included in this example for clarity, but actual printouts depend on the data description in the program.

The preceding is an example of checking at regular intervals (every tenth record). A check of any unusual conditions can be made by using various combinations of COBOL statements in the debug packet. For example:

```
IF OVERTIMEHRS GREATER THAN 2.0 EX-
   HIBIT NAMED PAYRCDHRS
```

In connection with the previous example, this statement could cause the entire pay record to be displayed whenever an unusual condition (overtime exceeding two hours) is encountered.

The EXHIBIT CHANGED statement also can be used to monitor conditions that do not occur at regular intervals. The values of data-names are listed only if the value has changed since the last execution of the statement. For example, suppose the program calculates postage rates to various cities. The flow of the program might be:



The EXHIBIT CHANGED statement in the program could be:

```
EXHIBIT CHANGED STATE CITY RATE
```

The output from the EXHIBIT CHANGED statement could appear as:

```
01  01  10
    02  15
    03
    04  10
02  01
    02  20
    03  15
    04
03  01  10
     .
     .
     .
```

The first column contains the code for a state, the second column contains the code for a city, and the third column contains the code for the postage rate. The value of a data-name is listed only if it is changed since the previous execution. For example, since the postage rate to city 02 and 03 in state 01 are the same, the rate is not printed for city 03.

The EXHIBIT CHANGED NAMED statement lists the data-name and the value of that data-name if the value has changed. For example, the program might calculate the cost of various methods of shipping to different cities. After the calculations are made, the following statement could be in the program:

```
EXHIBIT CHANGED NAMED STATE CITY RAIL
   BUS TRUCK AIR
```

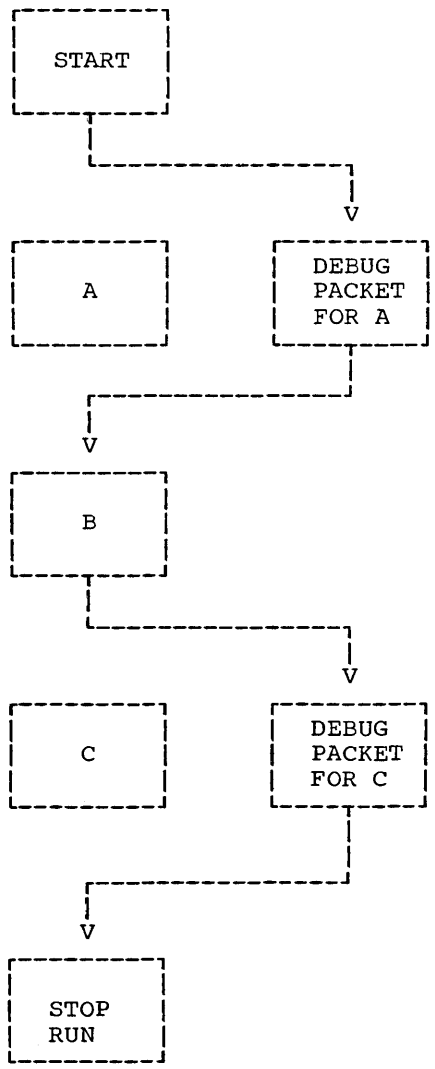
The output from this statement could appear as:

```
STATE = 01 CITY = 01 RAIL = 10 BUS =
   14 TRUCK = 12 AIR = 20

CITY = 02

CITY = 03 BUS = 06 AIR = 15

CITY = 04 RAIL = 30 BUS = 25 TRUCK =
   28 AIR = 34

STATE = 02 CITY = 01 TRUCK = 25

CITY = 02 TRUCK = 20 AIR = 30
     .
     .
     .
```

Note that a data-name and its value are listed only if the value has changed since the previous execution.

TESTING A PROGRAM SELECTIVELY

A debug packet allows the programmer to select a portion of the program for testing. The packet can include test data and can specify operations the programmer wants to be performed. When the testing is completed, the packet can be removed. The flow of control can be selectively altered by the inclusion of debug packets, as illustrated in the following example of selective testing of B:



In this program, A creates data, B processes it, and C prints it. The debug packet for A simulates test data. It is first in the program to be executed. In the packet, the last statement is GO TO B,

which permits A to be bypassed. After B is executed with the test data, control passes to the debug packet for C, which contains a GO TO statement that transfers control to the end of the program, bypassing C.

TESTING CHANGES AND ADDITIONS TO PROGRAMS

If a program runs correctly but changes or additions can make it more efficient, a debug packet can be used to test changes without modifying the original source program.

If the changes to be incorporated are in the middle of a paragraph, the entire paragraph, with the changes included, must be written in the debug packet. The last statement in the packet should be a GO TO statement that transfers control to the next procedure to be executed.

There are usually several ways to perform an operation. Alternative methods can be tested by putting them in debug packets.

The source program library facility can be used for program checkout by placing a source program in a library (see "Libraries"). Changes or additions to the program can be tested by using the BASIS card and any number of INSERT and DELETE cards. Such changes or additions remain in effect only for the duration of the run.

A debug packet can also be used in conjunction with the BASIS card to debug a program or to test deletions or additions to it. The debug packet is inserted in the input stream immediately following the BASIS card and any INSERT or DELETE cards.

DUMPS

If a serious error occurs during execution of a program, the job is abnormally terminated; any remaining steps are bypassed, and a dump is generated. The programmer can use the dump for program checkout. (However, any pending transfers to an external device may not be completed. For example, if a READY TRACE statement is in effect when the job is abnormally terminated, the last procedure-name may not appear on the external device.) In cases where the abnormal termination does not go to completion, a dump is not produced. This situation may cause duplicate name definition when the next job is run, and is discussed at the end of this section.

If a SYSABEND DD statement has been included in the execution-time job step, the system will provide the programmer with a printout, in hexadecimal and EBCDIC format, of main storage. Those areas occupied by the program, and its data at the time the error occurred, will be included. This printout is called an abnormal termination dump and is identified by the heading *** ABDUMP REQUESTED ***.

If a SYSABEND DD statement is not included in the execution-time job step, or its specification has been destroyed, an indicative dump is produced. This dump does not contain a printout of main storage.

Both dumps include a completion code designating the condition that caused the termination. The completion code consists of a system code and a user code. Only one of the codes is nonzero. A nonzero system code indicates that the control program detected the error.

A dump cannot be requested in the COBOL language.

The explanation of the system-generated completion codes and a complete description of the dumps are contained in the publication IBM System/360 Operating System: Programmer's Guide to Debugging, Form C28-6670.

ERRORS THAT CAN CAUSE A DUMP

Following is a discussion of some error conditions that can cause a program to be abnormally terminated and a dump to be listed. Input/output errors and errors caused by invalid data are discussed.

Input/Output Errors

Errors can occur while a COBOL file is being processed. For example, an uncorrectable input/output error can occur during data transmission. If the file being processed is organized sequentially and no error-processing declarative has been specified for the file, the job is terminated. If it is a QSAM file, the job will be terminated if no declarative and no EROPT option in the DD statement have been specified.

Another error that can cause termination is an attempt to read a file whose records are larger than those described in the source program. The chapter "Additional File Processing Information" contains more information about input/output errors.

Errors Caused By Invalid Data

The abnormal termination of a job may result if a data item in invalid format is operated on by a statement in the Procedure Division. This is particularly true of COMPUTATIONAL-3 (internal decimal) and numeric DISPLAY (external decimal) items.

Invalid format may be caused by (1) an invalid configuration in the data itself, or (2) a data description that does not match the data item, which results in incorrect processing of the data. Some of the program errors that might cause an abnormal termination are:

1. A data item in the Working-Storage Section is not initialized before it is used, and invalid data is picked up.

2. For an item whose usage is COMPUTATIONAL, COMPUTATIONAL-1, or COMPUTATIONAL-2, either the alignment is incorrect, or the description of the item does not specify the proper alignment. Some examples of when this can occur are:

   a. A redefining entry contains one or more of the above items and the redefined entry is not properly aligned.

   b. A record in the Linkage Section of a called program is described by an 01 entry and contains one or more of the above items, and the corresponding argument in the calling program is not properly aligned.

   c. A file, containing one or more of the above items, is blocked, but required interrecord slack bytes were not inserted when the file was created. If the file is later read as an input file, the alignment may not be correct.

3. An input file contains invalid data or data incorrectly specified by its data description. For example, the contents of the sign position of an internal or external decimal data item in the file may be invalid. The compiler does not generate a test to check the sign position for a valid configuration before the item is used as an operand.

4. No conversion or editing is done in a move involving a group item. Therefore, if a group item is moved to a group item and the subordinate data descriptions are incompatible, the new data in the receiving field may not match the corresponding data descriptions.

5. The SIZE ERROR option is not specified for the COMPUTE statement, the results of the calculation are larger than the specified resultant COMPUTATIONAL data-name, and the results are then used in subsequent calculations.

6. The SIZE ERROR option is not specified for a DIVIDE statement, and an attempt is made to divide by zero.

7. The USAGE specified for a redefining data item is different from the USAGE specified for the redefined item, and the item is referred to by the wrong name for the current content.

8. If a record contains a data item described by an OCCURS clause with the DEPENDING ON data-name option, data items in the record may be affected by a change in the value of data-name during the course of program execution. This may result in incorrectly described data. Additional information about how to correct this situation is included in the chapter "Programming Techniques."

9. The data description in the Linkage Section of a called program does not correctly describe the data defined in the calling program.

Other Errors

1. No DD statement is included for a file described in the source program and an attempt is made to open the file. A system console message is written; the programmer can elect to direct the operator to continue processing his program, but any READ or WRITE associated with the unlocated file will result in an ABEND. (See "Appendix H: Compiler Diagnostic Messages" for the format of the generated error message.)

2. A file is not opened and execution of a READ or WRITE statement for the file is attempted, or a MOVE to a record area in the file is attempted.

3. A GO TO statement, with no procedure name following it, is not properly initialized with an ALTER statement before the first execution of the GO TO statement.

4. Reference is made to an item in a file after end of data. This includes the use of the TERMINATE statement of the

Report Writer feature, if the CONTROL FOOTING, PAGE FOOTING, or REPORT FOOTING contain items that are in the file (e.g., SOURCE data-name, where data-name refers to an item in the file).

5. Block size for an F-format file is not an integral multiple of the record length.

6. A value specified for ACTUAL KEY is beyond the extents of the data set.

7. When adding records to a BDAM file, there is not enough room for a record from the track specified to the end of the data set (there may be room on earlier tracks, but it will not be used).

8. A READ is issued for a data set referenced on a DD DUMMY statement. The AT END condition is sensed immediately and any reference to a record in the data set produces unpredictable results.

9. Under MVT, a STOP RUN statement is executed before all files are closed.

In addition to errors that can result in an abnormal termination, errors in the source program can occur that cause parts of the program to be overlaid and the corresponding object code instructions to become invalid. If an attempt is then made to execute one of these instructions, an abnormal termination may result.

Some reasons for this are: the operation code of the instruction is invalid, the instruction results in a branch to an area containing invalid instructions, the instruction results in a branch to an area outside the program, such as an address protected area.

Some COBOL source program errors that can cause this overlaying are:

1. Using a subscript whose value exceeds the maximum specified in the associated OCCURS clause.

2. Using a data-name as a counter whose value exceeds the maximum value valid for that counter.

USING THE ABNORMAL TERMINATION DUMP

The programmer can determine the cause of an abnormal termination with the following material:

1. The COBOL program object code listing

2. A knowledge of the layout of the COBOL object module

3. The full abnormal termination dump in conjunction with the linkage editor map or cross reference list

The chapter entitled "Output" contains a format description of the linkage editor output and of the COBOL object code listing. Figure 30 in that chapter shows the layout of the COBOL program object module.

Note: The information in this section about the use of the abnormal termination dump applies only when running under PCP or MFT. For information about the abnormal termination dumps under MVT, see the publication IBM System/360 Operating System: Programmer's Guide to Debugging, Form C28-6670. Note that under the MVT option no indicative dumps are given.

The abnormal termination dump provides the address at which the load module has been loaded (load address) and the address of the instruction that caused the interrupt. The programmer computes the load module area by adding the load address to the load module length, as shown in the linkage editor output. It is now possible to determine whether the instruction falls within the load module. If it does not, the interrupt could have resulted from an improper branch to a point outside the load module or an error occurring in another part of the system.

If the instruction does fall within the load module, the programmer now determines in which part: the main program, a COBOL library subroutine, or a called program. The ranges of the various parts are determined by adding their relative origins, as shown in the linkage editor output, to the load address.

If the instruction occurred in an object module generated for a COBOL program, (i.e., the main program), the programmer can determine whether or not the instruction was one of the generated object code instructions. He can determine the address of the first instruction in the Procedure Division (as found in the object code listing) by adding its relative location to the location of the object module (load address plus relative origin). If it was one of the object code instructions, a similar technique can be used to locate the exact instruction. If it was not one of these instructions, the error has occurred in another part of the object module. Control possibly went there because of an improper branch.

If the instruction which initiated the dump occurred in a COBOL library subroutine, or if the original program called another program and the instruction occurred in the called program, the instruction can be located by a similar technique. The linkage editor cross reference list indicates the locations where the call to the program or subroutine in question was made.

The sample COBOL program and its output in Figure 33 illustrates in detail how an object code listing, cross reference list, and abnormal termination dump can be used together. Circled numbers in the example refer to the corresponding numbers in the text which follows. Note that all values are expressed in hexadecimal format.

```
00001          IDENTIFICATION DIVISION.
00002          PROGRAM-ID.  'ABEND'.
00003          REMARKS.
00004              THIS IS A PROGRAM TO ILLUSTRATE THE ABNORMAL TERMINATION.
00005          ENVIRONMENT DIVISION.
00006          CONFIGURATION SECTION.
00007          SOURCE-COMPUTER. IBM-360 E50.
00008          OBJECT-COMPUTER. IBM-360 E50.
00009          DATA DIVISION.
00010           WORKING-STORAGE SECTION.
00011              01  RECORDA.
00012                02  A  PICTURE S9(4)  VALUE  1234.
00013                02  B REDEFINES A   PICTURE  S9(7)  COMPUTATIONAL-3. (9)
00014          PROCEDURE DIVISION.
00015              COMPUTE B = B + 1.
00016              STOP RUN.          (8)
```

INTRNL NAME	LVL	SOURCE NAME	BASE	DISPL	INTRNL NAME	DEFINITION	USAGE	R	D	Q
DNM=1-030	01	RECORDA	BLI=1	000	DNM=1-030	DS 0CL4	GROUP			
DNM=1-049	02	A	BLI=1	000	DNM=1-049	DS 4C	DISP-NM			
DNM=1-059	02	B	BLI=1	000	DNM=1-059	DS 4P	COMP-3	*		

MEMORY MAP

TGT	00048
SAVE AREA	00048
SWITCH CELL	00090
TALLY CELL	00094
UNUSED	00098
ENTRY-SAVE	000A0
UNUSED	000A4
WORKING CELLS	000A8
OVERFLOW CELLS	001D8
TEMPORARY STORAGE CELLS	001D8
TEMPORARY STORAGE-2 CELLS	001D8
SBL-D CELLS	001D8
BLL CELLS	001D8
BL-I CELLS	001D8
SUBADR-I CELLS	001DC
ONCTL-I CELLS	001DC
PFMCTL-I CELLS	001DC
PFMSAV-I CELLS	001DC
VN-I CELLS	001DC
DECBADR-I CELLS	001DC
SAVE AREA =2	001DC
SAVE AREA =3	001DC
XSASW-I CELLS	001DC
XSA-I CELLS	001DC
PARAM CELLS	001DC
RPTSAV AREA	001DC

LITERAL POOL (HEX)

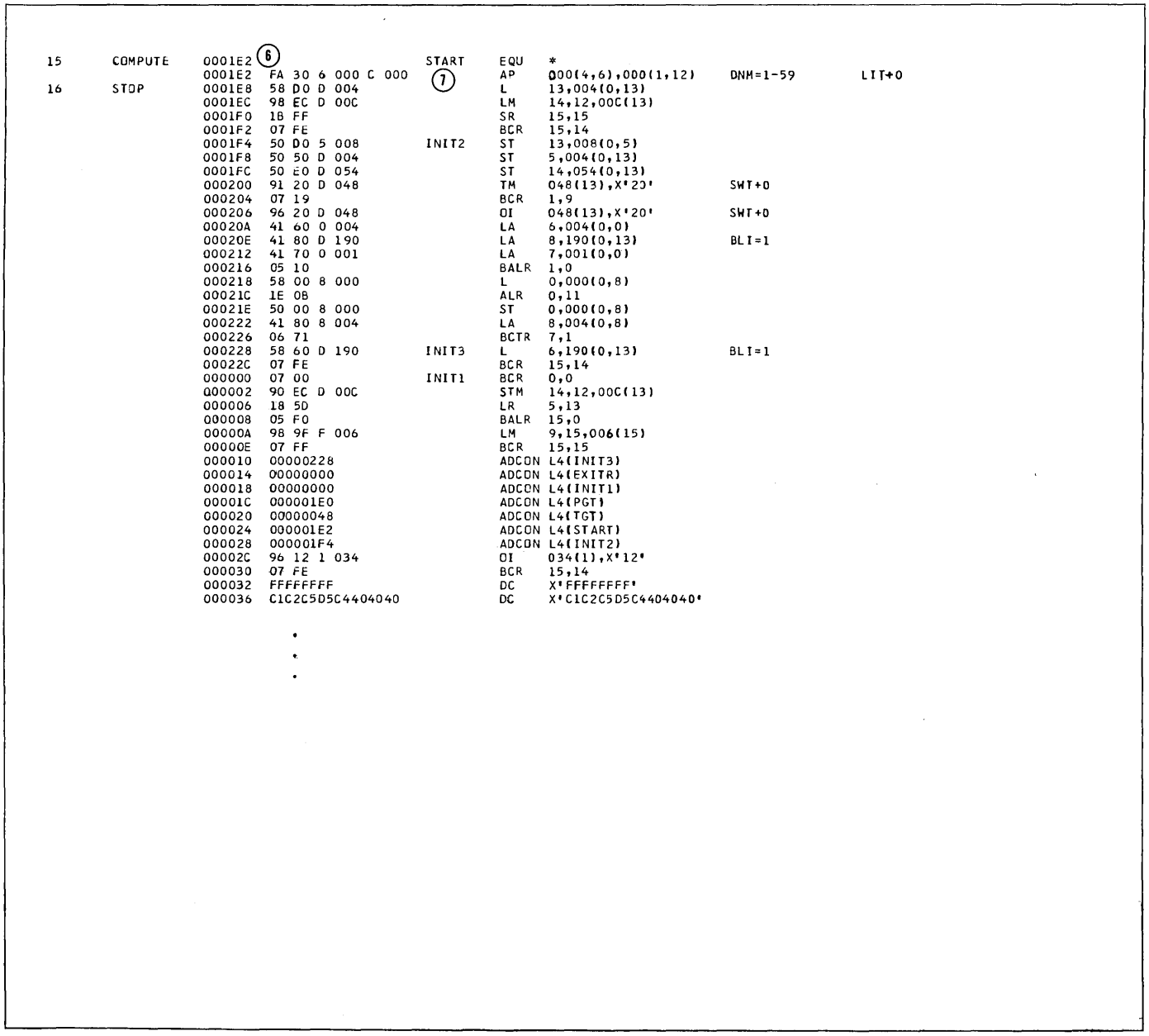
001E0 (LIT+0) 1C

PGT	001E0
OVERFLOW CELLS	001E0
VIRTUAL CELLS	001E0
PROCEDURE NAME CELLS	001E0
GENERATED NAME CELLS	001E0
DCB ADDRESS CELLS	001E0
VNI CELLS	001E0
LITERALS	001E0
DISPLAY LITERALS	001E1

REGISTER ASSIGNMENT

REG 6 BLI=1

• Figure 33. COBOL Program with Abnormal Termination Dump (Part 1 of 3)



```
15      COMPUTE  0001E2 (6)                    START   EQU   *
                 0001E2  FA 30 6 000 C 000      (7)     AP    000(4,6),000(1,12)   DNM=1-59      LIT+0
16      STOP     0001E8  58 D0 D 004                    L     13,004(0,13)
                 0001EC  98 EC D 00C                    LM    14,12,00C(13)
                 0001F0  1B FF                          SR    15,15
                 0001F2  07 FE                          BCR   15,14
                 0001F4  50 D0 5 008            INIT2   ST    13,008(0,5)
                 0001F8  50 50 D 004                    ST    5,004(0,13)
                 0001FC  50 E0 D 054                    ST    14,054(0,13)
                 000200  91 20 D 048                    TM    048(13),X'20'        SWT+0
                 000204  07 19                          BCR   1,9
                 000206  96 20 D 048                    OI    048(13),X'20'        SWT+0
                 00020A  41 60 0 004                    LA    6,004(0,0)
                 00020E  41 80 D 190                    LA    8,190(0,13)          BLI=1
                 000212  41 70 0 001                    LA    7,001(0,0)
                 000216  05 10                          BALR  1,0
                 000218  58 00 8 000                    L     0,000(0,8)
                 00021C  1E 0B                          ALR   0,11
                 00021E  50 00 8 000                    ST    0,000(0,8)
                 000222  41 80 8 004                    LA    8,004(0,8)
                 000226  06 71                          BCTR  7,1
                 000228  58 60 D 190            INIT3   L     6,190(0,13)          BLI=1
                 00022C  07 FE                          BCR   15,14
                 000000  07 00                  INIT1   BCR   0,0
                 000002  90 EC D 00C                    STM   14,12,00C(13)
                 000006  18 5D                          LR    5,13
                 000008  05 F0                          BALR  15,0
                 00000A  98 9F F 006                    LM    9,15,006(15)
                 00000E  07 FF                          BCR   15,15
                 000010  00000228                      ADCON L4(INIT3)
                 000014  00000000                      ADCON L4(EXITR)
                 000018  00000000                      ADCON L4(INIT1)
                 00001C  000001E0                      ADCON L4(PGT)
                 000020  00000048                      ADCON L4(TGT)
                 000024  000001E2                      ADCON L4(START)
                 000028  000001F4                      ADCON L4(INIT2)
                 00002C  96 12 1 034                    OI    034(1),X'12'
                 000030  07 FE                          BCR   15,14
                 000032  FFFFFFFF                      DC    X'FFFFFFFF'
                 000036  C1C2C5D5C4404040              DC    X'C1C2C5D5C4404040'
                         .
                         .
                         .
```

Figure 33. COBOL Program with Abnormal Termination Dump (Part 2 of 3)

```
   E-LEVEL LINKAGE EDITOR OPTIONS SPECIFIED--XREF,LIST
IEW0000   RUN       NOW ADDED TO DATA SET

                                   ----  CROSS REFERENCE TABLE  ----

  CONTROL SECTION                      ENTRY
    NAME    ORIGIN  LENGTH              NAME   LOCATION     NAME   LOCATION     NAME   LOCATION     NAME   LOCATION
  ABEND         00     22E (5)

  LOCATION  REFERS TO SYMBOL  IN CONTROL SECTION

 ENTRY ADDRESS       00
 TOTAL LENGTH       22E (4)

          .
          .
          .

  * * * A B D U M P  R E Q U E S T E D * * *
 JOB AbEND            STEP GO              TIME 003219   DATE 99366

 COMPLETION CODE      SYSTEM = 0C7 (1)

 PROGRAM INTERRUPTION (DATA) AT LOCATION 008202

 INTERRUPT AT 008208 (2)

 PSW AT ENTRY TO ABEND  FF15000D D0008208

 TCB  002258  RB  0003FD68   PIE     00000000   DEB 0003FCFC   TIOT 0003FF4C   CMP 800C7000   TRN 00000000
              MSS 000022E0   PK/FLG 10912408   FLG 000000FE   LLS  00000000   JLB 00000000   JSE 00000000
              FSA 0403FFB8   TCB     00000000   TME 000022F4

 ACTIVE RBS                                                  (3)

 PRB  008000  NM RUN          SZ/STAB 004A00C0   USE/EP 00008020   PSW FF15000D D0008208   Q 000000   WT/LNK 00002258

 SVRB 03FE70  NM SVC-401C     SZ/STAB 000CD072   USE/EP 00005828   PSW FF040033 400059E4   Q D103D1   WT/LNK 00008000
              RG 0-7   00008060    50008238    0000006C    03002259    00007BA0    0003FFB8    00008060    00000000
              RG 8-15  000081FC    00008248    00008020    00008020    00008200    0000836B    00008202    00008214

 SVRB 03FD68  NM SVC-105A     SZ/STAB 000CD072   USE/EP 00005828   PSW FF04000E 0003FC0C   Q DA03DA   WT/LNK 0003FE70
              RG 0-7   000000E0    000082A8    00000918    4000582A    00000000    0003F908    00008250    800059CE
              RG 8-15  000022E0    0000831E    00040000    00002258    00002258    00008288    50005A70    10008202

          .
          .
          .
```

Figure 33. COBOL Program with Abnormal Termination Dump (Part 3 of 3)

1. The completion code, (1), in the dump indicates the condition causing the abnormal termination. If the system part of the code is nonzero, the explanation is found in the publication IBM System/360 Operating System: Programmer's Guide to Debugging, Form C28-6670. In this example, invalid data is the reason for termination.

2. The INTERRUPT AT hhhhhh entry, (2), gives the hexadecimal address of the instruction following the instruction that initiated the interrupt and caused the dump. This address can be used to determine the relative location of the instruction in the load module (see item 4 below). In the example, the address is 8208.

3. To determine the main storage area occupied by the load module, add the total length of the module, in hexadecimal format, to its load address. The load address can be obtained from the USE/EP entry, (3), of the first ACTIVE RBS (Request Blocks) specification. The last six digits of this entry are the address of the entry point (INIT1) in the COBOL program. In this case, the address is 8020 in hexadecimal format.

   The total length of the load module is indicated in the TOTAL LENGTH entry, (4), in the linkage editor output (22E, in the example). The highest location in the load module is:

   8020+22E=824E

   Thus, the range is from 8020 to 824E. Since the 8208 address falls within this range, the instruction initiating the dump must be within the load module.

4. To determine the relative location within the load module of the instruction indicated in the INTERRUPT entry, subtract the load address from the address of the instruction. In the example, this becomes:

   8208-8020=1E8

5. To determine whether or not the instruction occurred in the object module generated for the program, compare its relative location (1E8) with the length, (5), of the object module. If the relative location is greater, the instruction occurred within one of the COBOL library subroutines and can be located by comparing its relative location with the relative origin of the subroutines. In this example, 1E8 is less than 22E, so the instruction occurred in the object module.

6. To determine whether or not the instruction was one of the object code instructions generated as a result of a statement in the Procedure Division of the source program, compare its relative location with the relative location of the first generated instruction in the Procedure Division, (6). In this example, the relative location of the instruction is greater than that of the first generated instruction (1E8>1E2), and so it can be found by locating the corresponding relative location. The immediately preceding object code instruction then is the instruction that initiated the dump (7). In this example, it is an instruction generated as a result of a COMPUTE statement. Checking back to the source program listing, the corresponding statement (8), is located and 'B' is seen to be the data-name that caused the trouble. B is defined at (9) as a COMPUTATIONAL-3 or packed decimal item, but the value at B is there as a result of a VALUE clause for A, the item that B redefines. This value is in zoned decimal format since there is no USAGE clause specified. The configuration of A is invalid for B and results in an interrupt.

INCOMPLETE ABNORMAL TERMINATION

If a job is abnormally terminated and the abnormal termination process goes to completion, the following procedures are carried out:

- A dump (ABDUMP) is produced by the system.
- The data sets in the job steps are disposed of as specz ified in the DISP parameter (i.e., kept, deleted, etc.). This is indicated in the job scheduler disposition messages produced for the job step.
- Temporary data sets, including those passed from previous job steps, are deleted.

When the abnormal termination process does not go to completion, none of these procedures will be carried out. Those data sets in the job step that were in existence previous to the point at which the error condition occurred will remain in effect. For data sets on direct-access volumes, the names will remain tabulated in the Volume

Table of Contents (VTOC) of the volume (see the chapter "Additional File Processing Information" for additional information on the VTOC). The result is that space needed by a subsequent job will be unavailable, or, if the same job is then rerun, duplicate name definitions will result for those data sets that are newly created in the job step. This is true for temporary data sets for which the system has assigned the name, as well as data sets for which the programmer has assigned the name.

SCRATCHING DATA SETS

To avoid duplicate name definition and to ensure that space will be available for newly created data sets, the programmer can scratch his direct-access volume data sets by using the utility program IEHPROGM. To scratch such a data set means to remove its data set label (which includes its name) from the VTOC and to make the space assigned to it available for reallocation. Scratching does not uncatalog any cataloged data sets. This is done by the UNCATLG option of the IEHPROGM.

Note: The information in this section about scratching data sets applies only when running under PCP. Under the MFT and MVT options, direct-access volume data sets are scratched automatically. For use of the system utilities under these options, see the publication IBM System/360 Operating System: Utilities, Form C28-6586.

If a DSNAME parameter has been specified in the DD statement for the data set, the IEHPROGM utility program requires the name of the data set. For data sets named by the programmer, the specified name is the dsname. For data sets for which the DSNAME=&name convention has been used, an internal name of the format

```
jobname.name
```

is assigned by the system, where jobname is the name of the job and name is in the &name. If no DSNAME parameter was specified, an internal name of the format

```
AAAAAAAA.AAAAAAAA.AAAAAAAA.AAAAAAAA.
   nnnnnnnn
```

is assigned by the system, where n denotes a digit between 0 and 9. For data sets with no DSNAME parameter there exists an option by which the programmer can specify that all such data sets on the volume be scratched, without having to specify their names.

If the programmer wishes to obtain a listing of the names of all the data sets on a volume, including system-assigned internal names, he can use the utility program IEHLIST. This program provides a listing of the VTOC of the volume.

Information on how to use these utility programs is contained in the publication IBM System/360 Operating System: Utilities, Form C28-6586. The following example illustrates a job control procedure that might be used to scratch temporary data sets:

```
//SCR        JOB      ,SCRATCH,MSGLEVEL=1
//STEP1      EXEC     PGM=IEHPROGM
//SYSPRINT   DD       SYSOUT=A
//DD1        DD       UNIT=2311,DISP=OLD
//DD2        DD       UNIT=2311,DISP=OLD,  X
//                    VOLUME=SER=222222
                .
                .
                .
//SYSIN      DD       *
           SCRATCH   DSNAME=GOJOB.TEMP,   X
//                   VOL=2311=222222,PURGE
           SCRATCH   VTOC,VOL=2311=222222,X
//                   SYS,PURGE
                .
                .
                .
/*
```

In this example, the SYSPRINT DD statement specifies the output data set for the listing and the DD1 DD statement specifies the system residence volume. The other DD statements specify the volume serial number of the mountable volumes on which the data sets have been written. These DD statements are needed to allocate the required devices. The first SCRATCH statement scratches a data set for which DSNAME=&TEMP had been specified on the DD statement, and the second SCRATCH statement scratches all data sets on the volume for which no DSNAME parameter had been specified.

Note that the possibility of duplicate name definition also applies to cataloged procedures in which temporary data sets are used.

For those procedures that are executed often, the programmer may wish to include, at the beginning of his job, a procedure to scratch all temporary data sets.

This chapter describes some techniques for increasing the efficiency of a COBOL program. It is organized according to the COBOL clause to which the techniques are related. The clauses are arranged alphabetically under the division (Data or Procedure) to which they belong.

DATA DIVISION

The compiler assigns a permanent register to the first 4096-byte area of items defined in the Working-Storage Section and five registers to the first five data sets defined in the File Section. Thus, for optimum addressing of storage by the control program, the programmer should define the most used data items in Working-Storage and the most frequently accessed data sets first.

When writing Data Division statements, the COBOL programmer need not concern himself with data problems such as decimal-point alignment and mixed data formats; the compiler generates extra instructions to perform the necessary adjustments.

The Procedure Division operations that most often require adjustments include the MOVE statement, the IF statement when used in a relation test, and the arithmetic operations. Extra code can be avoided by more efficient use of Data Division clauses, such as the OCCURS, REDEFINES, PICTURE, and USAGE clauses.

OCCURS Clause

SUBSCRIPTS: If a subscript is represented by a constant, the location of the subscripted data item within the table or list is resolved at compile time.

If a subscript is represented by a data-name, the location is resolved at execution time. The location is not recalculated when the value of data-name is changed as a result of a data transfer to an elementary item to which data-name relates by redefinition. The more efficient format, in this case, is COMPUTATIONAL, with PICTURE size less than five integers.

THE DEPENDING ON OPTION: If a data item described by an OCCURS clause with the DEPENDING ON data-name option is followed by nonsubordinate data items, a change in the value of data-name during the course of program execution will have the following effects:

1. The size of any group described by or containing the related OCCURS clause will reflect the new value of data-name.
2. The location of any nonsubordinate items following the item described with the OCCURS clause will be affected by the new value of data-name. If the user wishes to preserve the contents of these items, the following procedure can be used: prior to the change in data-name, move all nonsubordinate items following the variable item to a work area; after the change in data-name, move all the items back.

For example, assume that the Data Division of a program contains the following coding:

```
01 ANYRECORD
   02 A PICTURE S999 COMPUTATIONAL-3.
   02 TABLEA PICTURE S999 OCCURS 100
      TIMES DEPENDING ON A.
   02 GROUPB
      Subordinate data items.
               End of record.
```

GROUPB items are not subordinate to TABLEA, which is described by the OCCURS clause. Assuming that WORKB is a work area with the same data structure as GROUPB, the following procedural coding could be used:

```
1  MOVE GROUPB TO WORKB

2  COMPUTE new value of A

3  MOVE WORKB TO GROUPB
```

The above statements can be avoided by putting the OCCURS...DEPENDING ON clauses at the end of the record.

Note: Data-name can also change because of a change in the value of an item that redefines it. In this case, the group size and the location of nonsubordinate items as described in the two preceding paragraphs are indeterminable.

DEPENDING ON Used More Than Once: Problems may arise when OCCURS...DEPENDING ON is used more than once, or on more than one level, in the same logical record. For example, in the description of an output file, the following two uses of the OCCURS clause are valid:

```
FD F1.
01 R1.
   02 A USAGE non computational.
   02 B USAGE non computational.
   02 G-1.
      03 X OCCURS integer TIMES
                    DEPENDING ON A.
      03 Y OCCURS integer TIMES
                    DEPENDING ON B.
```

and

```
FD F2.
01 R2.
   02 A USAGE non computational.
   02 B USAGE non computational.
   02 G-2.
      03 X OCCURS integer TIMES
                    DEPENDING ON A.
         04 Y OCCURS integer TIMES
                    DEPENDING ON B.
```

The problem arises when either A or B do not contain information that is valid for their USAGE. If A contains invalid data for its usage, then the value of B cannot be changed, and vice versa.

This situation can be avoided by using either of the following two methods:

a.
```
FD...
01 R1.
   02 DUMMY-GROUP-OF-A-AND-B.
      03 A.
      03 B.
   02 GROUP A.
      03 X OCCURS integer TIMES
                    DEPENDING ON A.
      03 Y OCCURS integer TIMES
                    DEPENDING ON B.

WORKING-STORAGE SECTION.
01 WORKING-STORAGE-GROUP-A-AND-B.
   02 DUMMY-A.
   02 DUMMY-B.
```

First, values are moved to DUMMY-A and DUMMY-B instead of to A and B as would be done ordinarily. Next, WORKING-STORAGE-GROUP-A-AND-B is moved to DUMMY-GROUP-OF-A-AND-B, and the record is properly initialized.

b. Environment Division:

```
APPLY WRITE-ONLY ON F-1.
```

Data Division:

```
FD F-1.
01 RCD-NAME PICTURE X
      (maximum length of record).

WORKING-STORAGE SECTION.
01 R-N.
   02 A.
   02 B.
   02 GROUP A.
      03 X OCCURS integer TIMES
                    DEPENDING ON A.
      03 Y OCCURS integer TIMES
                    DEPENDING ON B.
```

The record is described and worked on in the Working-Storage Section. Then, with the execution of WRITE RCD-NAME FROM R-N, the correct result is obtained.

Additional Considerations: Further restrictions on the use of the OCCURS... DEPENDING ON clause are noted below:

- When OCCURS...DEPENDING ON is specified for one record of a file and data-name is in the same record, the relative location of data-name in all records for that file must contain valid information of the same usage clause and size.

- When more than one record of a file contains the OCCURS...DEPENDING ON clause, the longest possible record must be defined first.

- When more than one record of an input file contains the OCCURS...DEPENDING ON clause, or when the record with the OCCURS...DEPENDING ON clause is not the first record described under the FD entry, once a READ statement has been executed for the file, the programmer must determine whether or not he is working on a record other than the first record described under the FD entry for the file. If it is not the first, and if it contains an OCCURS... DEPENDING ON clause, the programmer must move a data-name controlling the OCCURS...DEPENDING ON items to itself to initialize the processing of that record correctly. For example, if the following record is read in and its description is not the first one under the FD entry:

```
01 REC1.
   02 A, PICTURE S9.
   02 B, PICTURE S9.
   02 GROUP ONE.
      03 R OCCURS integer TIMES
                    DEPENDING ON A,
                     PICTURE X(20).
      03 S OCCURS integer TIMES
                    DEPENDING ON B,
                     PICTURE X(20).
```

then one of the following statements must be executed before processing of the record can begin:

```
MOVE A TO A.
MOVE B TO B.
```

- When a record in an output file contains an OCCURS...DEPENDING ON clause, following each WRITE the length of all variable length groups and the location of nonsubordinate data items following the OCCURS clause must be recalculated. All data-names of the OCCURS... DEPENDING ON clause that are defined in the File Section are assumed to contain the maximum declared value. For those data-names that are not defined in the File Section the current contents of the OCCURS...DEPENDING ON data-name are used. For example, if the following are Data Division entries for output files:

```
FILE SECTION.

01   R1.

     02 A, PICTURE S9(4).

     02 B, OCCURS 5 TIMES DEPENDING
        ON A.

01   R2.

     02 C, OCCURS 5 TIMES DEPENDING
        ON D.

WORKING-STORAGE SECTION.

77   D, PICTURE S9(4).
```

  when the following code is executed in the Procedure Division:

```
MOVE 1 TO A, D.  WRITE R1.  WRITE R2.
```

  B is re-initialized to 5 occurrences while C remains at one occurrence.

- When OCCURS...DEPENDING ON data-name is used more than once in the same logical record, the different data-names must all appear in the same Data Division Section. For example, if the multiple OCCURS...DEPENDING ON data-name appears in a record defined in the File Section, all the data-names must also be defined in the File Section.

REDEFINES Clause

REUSING DATA AREAS: The main storage area can be used more efficiently by writing different data descriptions for the same data area. For example, the coding that follows shows how the same area can be used as a work area for the records of several input files that are not processed concurrently:

```
WORKING-STORAGE SECTION.

     01 WORK-AREA-FILE1.

          Largest record description for
            FILE1

     01 WORK-AREA-FILE2 REDEFINES
        WORK-AREA-FILE1.

          Largest record description for
            FILE2
          .
          .
          .
```

ALTERNATE GROUPINGS AND DESCRIPTIONS: Program data can be described more efficiently by providing alternate groupings or data descriptions for the same data. As an example of alternate grouping, suppose a program makes references both to a field and to its subfields where it would be more efficient to describe the subfields with different usages. This can be done with the REDEFINES clause as follows:

```
     01 PAYROLL-RECORD.

        02 EMPLOYEE-RECORD PICTURE X(28).

        02 EMPLOYEE-FIELD REDEFINES
           EMPLOYEE-RECORD.

              03 NAME PICTURE X(24)
                 DISPLAY.

              03 NUMBER PICTURE S9(5)
                 COMPUTATIONAL.

        02 DATE-RECORD PICTURE X(10).
```

As an example of different data descriptions specified for the same data, the following illustrates how a table can be initialized:

```
        02 VALUE-A.

              03 A1 PICTURE S9(9)
                 COMPUTATIONAL VALUE IS
                 ZEROES.

              03 A2 PICTURE S9(9)
                 COMPUTATIONAL VALUE IS 1.
             .
             .
             .

        02 TABLEA REDEFINES VALUE-A PICTURE
           S9(9) COMPUTATIONAL OCCURS 100
           TIMES.
```

DECIMAL-POINT ALIGNMENT: Procedure Division operations are most efficient when the decimal positions of the data items involved are aligned. If they are not, the compiler generates instructions to align the decimal positions before any operation involving the data items can be executed.

Assume, for example, that a program contains the following instructions:

```
77 A PICTURE S999V99
77 B PICTURE S99V9
      .
      .
      .
   ADD A TO B
```

Time and internal storage space are saved by defining B as:

```
77 B PICTURE S99V99
```

If it is inefficient to define B differently, a one-time conversion can be done, as explained under "Data Format Conversion."

FIELDS OF UNEQUAL LENGTH: When a data item is moved to another data item of a different length, no adjustment has to be made. However, if the items are external decimal items, the compiler generates instructions to insert zeros in the high-order positions of the receiving field, if it is the larger.

This generation of extra instructions can be avoided if the sending field is described with a length equal to or greater than the receiving field.

SIGN USAGE: The absence or presence of a plus or minus sign in the description of an arithmetic field often can affect the efficiency of a program. The following paragraphs discuss some of the considerations.

Decimal Items: The sign position in an internal or external decimal item can contain:

1. A plus or minus sign. If S is specified in a PICTURE clause, a plus or minus sign is inserted if (a) the item is in the Working-Storage Section and a VALUE clause has been specified or if (b) a value for the item is assigned as a result of an arithmetic operation during execution of the program. If an external decimal item is punched, printed, or displayed, an overpunch will appear in the low-order digit. In EBCDIC code, the configuration for low-order zeros is not normally a printable character. Low-order digits of positive values will be represented by one of the letters A through I (digits 1 through 9); low-order digits of negative values will be represented by one of the letters J through R (digits 1 through 9).

2. A hexadecimal F. If S is not specified in the PICTURE clause, an F is inserted in the sign position if (a) the item is in the Working-Storage Section and a VALUE clause has been specified or if (b) a value for the item is developed by the execution of the program. An F is treated as positive, but is not an overpunch.

3. An invalid configuration. If an internal or external decimal item contains an invalid configuration in the sign position, and if the item is involved in a Procedure Division operation, the program will be abnormally terminated.

Unsigned items, that is, items for which no S has been specified, are treated as absolute values. The compiler tries to ensure that a positive value is present for these items. Whenever a value (signed or unsigned) is stored in, or moved in an elementary move, to an unsigned item, a hexadecimal F is stored in the sign position of the unsigned item. For example, if an arithmetic operation involves signed operands and an unsigned result field, compiler-generated code will insert an F in the sign position of the result field when the result is stored.

For internal and external decimal items used as input, it is the user's responsibility to ensure that the input data is valid. The compiler does not generate a test to ensure that the configuration in the sign position is valid.

When a group item is involved in a move, the data is moved without regard to the level structure of the group items involved. The possibility exists that the configuration in the sign position of a subordinate numeric item may be destroyed. Therefore, caution should be exercised in moves involving group items with subordinate numeric fields or with other group operations such as READ or ACCEPT.

Binary Items: An S must be specified in the PICTURE clause of a binary item, since a binary item is always treated as a signed item.

USAGE Clause

DATA FORMATS: As shown in Figure 34, COMPUTATIONAL, COMPUTATIONAL-1, and COMPUTATIONAL-2 items have specific boundary alignment requirements. To ensure correct alignment, either the programmer or the compiler may have to add slack bytes. To avoid having to add the slack bytes, items that require double-word alignment can be defined first (COMPUTATIONAL-2), items that require full-word alignment can be defined next (COMPUTATIONAL-1 and COMPUTATIONAL items whose PICTURE is S9(5) through S9(18)), and items that require half-word alignment are defined last (COMPUTATIONAL items whose PICTURE is S9 through S9(4)). Slack bytes are fully discussed in the publication IBM System/360 Operating System: COBOL Language, Form C28-6516.

DATA FORMAT CONVERSION: Operations involving mixed, elementary numeric data formats require conversion to a common format. This usually means that additional storage is used and execution time is increased. The code generated must often move data to an internal work area, perform any necessary conversion, and then execute the indicated operation. Often, too, the result may have to be converted in the same way.

If it is impractical to use the same data formats throughout a program, and if two data items of different formats are frequently used together, a one-time conversion can be effected. For example, if A is defined as a COMPUTATIONAL item and B as a COMPUTATIONAL-3 item, A can be moved to a work area that has been defined as COMPUTATIONAL-3. This move causes the data in A to be converted to a COMPUTATIONAL-3 format. Whenever A and B are used in a Procedure Division operation, the work area can be referred to, rather than A. Using this technique, the conversion is performed only once, instead of being done once for each operation.

Moves to and from group items, without the CORRESPONDING option, as well as comparisons involving group items, are done without conversion.

The seven data conversion statements below contain information that indicates the type of conversions performed on mixed elementary data formats by the code generated for each statement.

Numeric DISPLAY to COMPUTATIONAL-3:

To Move Data: Converts DISPLAY data to COMPUTATIONAL-3 data.

To Compare Data: Converts DISPLAY data to COMPUTATIONAL-3 data.

To Perform Arithmetic Operations: Converts DISPLAY data to COMPUTATIONAL-3 data.

Numeric DISPLAY to COMPUTATIONAL:

To Move Data: Converts DISPLAY data to COMPUTATIONAL-3 data and then to COMPUTATIONAL data.

To Compare Data: Converts DISPLAY to COMPUTATIONAL or converts both DISPLAY and COMPUTATIONAL data to COMPUTATIONAL-3 data.

To Perform Arithmetic Operations: Converts DISPLAY data to COMPUTATIONAL-3 or COMPUTATIONAL data.

COMPUTATIONAL-3 to COMPUTATIONAL:

To Move Data: Moves COMPUTATIONAL-3 data to a work field and then converts COMPUTATIONAL-3 data to COMPUTATIONAL data.

To Compare Data: Converts COMPUTATIONAL data to COMPUTATIONAL-3 or vice-versa, depending on the size of the field.

To Perform Arithmetic Operations: (Same as for a comparison).

COMPUTATIONAL to COMPUTATIONAL-3:

To Move Data: Converts COMPUTATIONAL data to COMPUTATIONAL-3 data in a work field, then moves the work field.

To Compare Data: Converts COMPUTATIONAL to COMPUTATIONAL-3 data or vice-versa, depending on the size of the field.

To Perform Arithmetic Operations: (Same as for a comparison).

COMPUTATIONAL to Numeric DISPLAY:

To Move Data: Converts COMPUTATIONAL data to COMPUTATIONAL-3 data and then to DISPLAY data.

To Compare Data: Converts DISPLAY to COMPUTATIONAL or both COMPUTATIONAL and DISPLAY data to COMPUTATIONAL-3 data, depending on the size of the field.

To Perform Arithmetic Operations: Depending on the size of the field, converts DISPLAY data to COMPUTATIONAL data, or both DISPLAY and COMPUTATIONAL data to COMPUTATIONAL-3 data, in which case the result is generated in a COMPUTATIONAL-3 work area and then con-

verted and moved to the DISPLAY result field.

COMPUTATIONAL-3 to Numeric DISPLAY:

To Move Data: Converts COMPUTATIONAL-3 data to DISPLAY data.

To Compare Data: Converts DISPLAY data to COMPUTATIONAL-3 data. DISPLAY data to COMPUTATIONAL-3 data. The result is generated in a COMPUTATIONAL-3 work area and is then converted and moved to the DISPLAY result field.

Numeric DISPLAY to Numeric DISPLAY:

To Perform Arithmetic Operations: Converts all DISPLAY data to COMPUTATIONAL-3 data. The result is generated in a COMPUTATIONAL-3 work area and is then converted to DISPLAY and moved to the DISPLAY result field.

Special Considerations for DISPLAY and COMPUTATIONAL Fields

NUMERIC DISPLAY FIELDS: Zeros and blanks are not inserted in numeric DISPLAY fields by the instruction set. When numeric DISPLAY data is moved, the compiler generates instructions that insert any necessary zeros or blanks in the DISPLAY fields. When numeric DISPLAY data is compared, and one field is smaller than the other, the compiler generates instructions to move the smaller item to a work area where zeros or blanks are inserted.

COMPUTATIONAL FIELDS: COMPUTATIONAL fields can be aligned on either a halfword or fullword boundary. If an operation involves COMPUTATIONAL fields of different lengths, the halfword field is automatically expanded to a fullword field. Therefore, mixed halfword and fullword fields require no additional operations.

COMPUTATIONAL-1 AND COMPUTATIONAL-2 FIELDS: If an arithmetic operation involves a mixture of short-precision and long-precision fields, the compiler generates instructions to expand the short-precision field to a long-precision field before the operation is executed.

COMPUTATIONAL-3 FIELDS: The compiler does not have to generate instructions to insert high-order zeros for ADD and SUBTRACT statements that involve COMPUTATIONAL-3 data. The zeros are inserted by the instruction set.

Data Formats in the Computer

The following examples illustrate how the various COBOL data formats appear in the computer in EBCDIC (Extended Binary-Coded-Decimal Interchange Code) format. More detailed information about these data formats appear in IBM System/360 Principles of Operation, Form A22-6821.

Numeric DISPLAY (External Decimal): Suppose the value of an item is -1234, and the PICTURE and USAGE are:

```
PICTURE 9999 DISPLAY.

        or

PICTURE S9999 DISPLAY.
```

The item appears in the computer in the following forms respectively:

```
| F1 | F2 | F3 | F4 |
 Byte

| F1 | F2 | F3 | D4 |
 Byte
```

Hexadecimal F is treated arithmetically as plus in the low-order byte. The hexadecimal character D represents a negative sign.

COMPUTATIONAL-3 (Internal Decimal): Suppose the value of an item is +1234, and its PICTURE and USAGE are:

```
PICTURE 9999 COMPUTATIONAL-3.

        or

PICTURE S9999 COMPUTATIONAL-3.
```

The item appears in the computer in the following forms, respectively:

```
| 01 | 23 | 4F |
      Byte

| 01 | 23 | 4C |
      Byte
```

Hexadecimal F is treated arithmetically as plus. The hexadecimal character C represents a positive sign.

Type of Data	Bytes Required	Boundary Alignment Required	Typical Usage	Converted for Arithmetic Operations	Special Characteristics
DISPLAY (external decimal)	1 per digit	No	Input from cards, output to cards, listings	Yes	May be used for numeric fields up to 18 digits long. Fields over 15 digits require extra instructions if used in computations.
DISPLAY (external floating point)	1 per character (except for V)	No	Input from cards, output to cards, listings	Yes	Converted to internal floating point long precision format via COBOL library subroutine.
COMPUTATIONAL-3 (internal decimal)	1 byte per 2 digits plus 1 byte for the low order digit	No	Input to a report item Arithmetic fields Work areas	Sometimes when a small COMPUTATIONAL-3 item is used with a small COMPUTATIONAL item.	Requires less space than DISPLAY. Convenient form for decimal alignment. Can be used in arithmetic computations without conversion.
COMPUTATIONAL (binary)	2 if 1≤N≤4 4 if 5≤N≤9 8 if 10≤N≤18 where N is the number of 9s in the picture	half-word full-word full-word	Subscripting Arithmetic fields	Sometimes for both mixed and unmixed usages	Rounding and ON SIZE ERROR tests are cumbersome if calculated result is greater than 9(9). Always must be signed. Fields of over 9 digits require more handling.
COMPUTATIONAL-1 (internal floating point)	4 (short-precision)	full-word	Fractional exponentiation	No	Tends to produce less accuracy if more than 17 significant digits are required and if the exponent is big. Requires floating-point feature.
COMPUTATIONAL-2 (internal floating point)	8 (long-precision)	double-word	Fractional exponentiation when more precision is required	No	Same as COMPUTATIONAL-1

Figure 34. Data Format Characteristics

Note: Since the low-order byte of an internal decimal number always contains a sign field, an item with an odd number of digits can be stored more efficiently than an item with an even number of digits. Note that a leading zero is inserted in the above example.

COMPUTATIONAL (Binary): Suppose the value of an item is 1234, and its PICTURE and USAGE are:

```
PICTURE S9999 COMPUTATIONAL.
```

The item appears in the computer in the following form:

```
|0|000 | 0100| 1101 | 0010 |
 Sign
Position
```

A 0-bit in the sign position means the number is positive. Negative numbers are represented in two's complement form; thus, the sign position of a negative number will always contain a 1-bit.

For example -1234 would appear as follows:

```
|1| 111| 1011| 0010 | 1110 |
 Sign
Position
```

Binary Item Manipulation: A binary item is allocated storage ranging from one halfword to two words, depending on the number of 9s in its PICTURE. Figure 35 is an illustration of how the compiler allocates this storage. Note that it is possible for a value larger than that implied by the PICTURE to be stored in the item. For example, PICTURE S9(4) implies a maximum value of 9,999, although it could actually hold 32,767.

Since, in most cases, a binary item is manipulated with respect to the actual storage capacity allotted to it, the programmer can ignore this situation. In the following cases, however, he must be careful of his data:

1. When the ON SIZE ERROR option is used, the size test is made on the basis of the maximum value allowed by its PICTURE, although the result field is left unchanged. When no ON SIZE ERROR option is used, however, the result of a computation could exceed the actual capacity of the field in which it is stored. Subsequent computations would then be made with erroneous data.

2. When a binary item is displayed or exhibited, the value used is a function of the number of 9s in the picture.

3. When the actual value of a positive number is significantly larger than its picture value, a 1 could result in the sign position of the item, causing the item to be treated as a negative number in subsequent operations.

Figure 36 illustrates four cases pertaining to an item described as PICTURE S9 COMPUTATIONAL and having an initial value of zero. One halfword of storage has been allocated, and no ON SIZE ERROR option is involved. Note that if the ON SIZE ERROR option had been specified, it would have been executed for cases B, C, and D. The final value would remain as it was initially, i.e., zero.

COMPUTATIONAL-1 or COMPUTATIONAL-2 (Floating Point): Suppose the value of an item is +1234, and its USAGE is COMPUTATIONAL-1, the item appears in the computer in the following form:

```
|0|100  0011|0100 1101 0010 0000 0000 0000|
 S 1       7 8                            31
```

S is the sign position of the number.

A 0-bit in the sign position indicates that the sign is plus.

A 1-bit in the sign position indicates that the sign is minus.

This form of data is referred to as floating-point. The example illustrates short-precision floating-point data. In long-precision, the fraction (characteristic, mantissa, exponent, etc.) length is 56 bits. (For a detailed explanation of floating-point representation, refer to IBM System/360 Principles of Operation, Form A22-6821.)

PROCEDURE DIVISION

A program can often be made more efficient in the Procedure Division with some of the techniques described below.

Intermediate Results

The compiler treats arithmetic statements as a succession of operations and sets up intermediate result fields to contain the results of these operations. Examples of such statements are the COMPUTE

PICTURE	Maximum Working Value	Assigned Storage
S9 through S9(4)	32,767	one halfword
S9(5) through S9(9)	2,147,483,647	one word
S9(10) through S9(18)	9,223,372,036,854,775,807	two words

Figure 35. Relationship of PICTURE to Storage Allocation

Case	Actual Hexa-Decimal Value	Decimal Value	Hexadecimal Value in Allocated Storage	Decimal Value in Allocated Storage	Display or Exhibit Value
A	0000 0008	+8	0008	+8	8
B	0000 000A	+10	000A	+10	0
C	0000 C350	+50000	C350	-15536	6
D	0001 0001	+32769	0001	+1	1

Figure 36. Treatment of Varying Values in a Data Item of PICTURE S9

statement, arithmetic verbs, and statements containing arithmetic expressions. Appendix E in the publication IBM System/360 Operating System: COBOL Language, Form C28-6516 describes the algorithms used by the compiler to determine the number of places reserved for intermediate result fields.

Intermediate Results and Binary Data Items: If an operation involving binary operands requires an intermediate result greater than 18 digits, the compiler converts the operands to internal decimal before performing the operation. If the result field is binary, the result will be converted from internal decimal to binary.

If an intermediate result will not be greater than nine digits, the operation is performed most efficiently as binary data fields.

Intermediate Results and COBOL Library Subroutines: If a decimal multiplication operation requires an intermediate result greater than 30 digits, a COBOL library subroutine is used to perform the multiplication. The result of this multiplication is then truncated to 30 digits.

A COBOL library subroutine is used to perform division if (1) the divisor is equal to or greater than 15 digits, (2) the length of the divisor plus the length of the dividend is greater than 16 bytes, or (3) the scaled dividend is greater than 30 digits (a scaled dividend is a number that has been multiplied by a power of ten in order to obtain the desired number of decimal places in the quotient).

Intermediate Results Greater than 30 Digits: Whenever the number of digits in a decimal intermediate result field is greater than 30, the field is truncated to 30 digits. A warning message will be generated at compilation time, but program flow will not be interrupted at execution time. This truncation may cause a result to be invalid.

If the possibility exists that an intermediate result field may exceed 30 digits, truncation can be avoided by the specification of floating-point operands (COMPUTATIONAL-1 or COMPUTATIONAL-2); however, accuracy may not be maintained.

Intermediate Results and Floating-Point Data Items: If an operation that results in a floating-point intermediate result field involves an exponent overflow, the job will be abnormally terminated. (If the binary or internal decimal data is in accord with its data description, no interrupt can occur because of an overflow condition in an intermediate result. This is due to the truncation described in the preceding paragraph.)

Intermediate Results and the ON SIZE ERROR Option: The ON SIZE ERROR option applies only to final calculated results and not to intermediate result fields.

MOVE Statement

Performing a MOVE for an item greater than 256 bytes in length requires the generation of more instructions than are required for a MOVE of an item of 256 bytes or less.

OPEN and CLOSE Statements

There are two ways to open or close several files. For example, with OPEN they are:

```
OPEN INPUT file-name-A  file-name-B
    file-name-C
```

and

```
OPEN INPUT file-name-A
OPEN INPUT file-name-B
OPEN INPUT file-name-C
```

Each opening or closing of a file requires the use of main-storage area that is directly proportional to the number of files being opened. Opening or closing more than one file with the same statement is faster than using a separate statement for each file. Separate statements, however, require less storage.

If an output file has not been closed when a STOP statement is encountered, the operating system will write the current contents of the buffers prior to closing the file. As a result, extraneous data may be placed on the output file.

The Report Writer feature permits the programmer to specify the format of a printed report. In the Data Division, the programmer describes once only the names and formats of the reports he wants produced. In the Procedure Division he writes the statements that generate the reports. Detailed information on using the Report Writer feature can be found in the publication IBM System/360 Operating System: COBOL Language, Form C28-6516.

RECORD LENGTH AND FORMAT

The length of a report output record is always 144 characters. The RECORD CONTAINS clause of the File Description entry associated with the report is implicitly overridden by the COLUMN clause in the Report Section of the Data Division, and, therefore, does not determine the length of a report record.

The format of a report output record (i.e., the RECORDING MODE), if not specified, will be determined by the compiler.

USING CONTROL ITEMS IN CONTROL FOOTINGS

When control names are used as information to be displayed after control breaks, the programmer should remember that it is the change in a control name that brings about a break. The control name displayed in the heading or footing may therefore be incorrect.

For example, the programmer might be using the SOURCE clause to print COST FOR JANUARY $X.XX as a footing when the end of the January records has been reached. However, the end of the January records occurs when the control name (MONTH) changes to FEBRUARY, and the footing is thus put out as COST FOR FEBRUARY $X.XX. This situation can be avoided by saving the control name in a temporary area.

SUMMING TECHNIQUE

The object program can be made more efficient with respect to execution time by keeping in mind the fact that Report Writer source coding is treated as though the programmer had written the program in COBOL without the Report Writer feature. Therefore, a complex source statement or series of statements will generally be executed faster than simple statements which perform the same function. The example below shows two coding techniques for the Report Section of the Data Division. Method 2 uses the more complex statements.

```
RD.....CONTROLS ARE YEAR MONTH WEEK DAY.

Method 1: 01 TYPE CONTROL FOOTING YEAR.
             02 SUM COST.
          01 TYPE CONTROL FOOTING MONTH.
             02 SUM COST.
          01 TYPE CONTROL FOOTING WEEK.
             02 SUM COST.
          01 TYPE CONTROL FOOTING DAY.
             02 SUM COST.

Method 2: 01 TYPE CONTROL FOOTING YEAR.
             02 SUM A.
          01 TYPE CONTROL FOOTING MONTH.
             02 A SUM B.
          01 TYPE CONTROL FOOTING WEEK.
             02 B SUM C.
          01 TYPE CONTROL FOOTING DAY.
             02 C SUM COST.
```

Method 2 will execute faster. One addition will be performed for each day, one more for each week, and one for each month. In Method 1, four additions will be performed for each day.

LEVELS

The compiler expects only levels 01 and 02 in Report Group descriptions, that is, a group entry followed by elementary entries. The 01 entry must contain a TYPE clause and may contain a LINE and a NEXTGROUP clause. Level numbers higher than 02 should not be used. The 02 entries must contain a SOURCE, VALUE, or a SUM clause and must not contain a TYPE or a NEXTGROUP clause.

TALLY

TALLY may be used as the operand of SOURCE and SUM clauses.

OUTPUT LINE OVERLAY

The Report Writer output line is put together by the use of an internal redefines specification indexed by the integer that specifies the column. There is no check to prevent overlay on any line. In the following example,

```
02  COLUMN 10 PICTURE X(20) VALUE
    'xxx...'.

02  COLUMN  12  PICTURE X(5) VALUE,
    etc.
```

the length of 20 beginning in column 10 followed by a specification for column 12 could cause field overlay.

LINE-COUNTER

Line and page counters may be referred to in the Procedure Division. The generated program relies entirely on the value of LINE-COUNTER to maintain correct alignment. Therefore, if the programmer references LINE-COUNTER using its value, and not changing it, he need not keep track of his alignment.

PAGE BREAKS

The Report Writer page break routine operates independently of the routines that are executed after any control breaks (except that a page break will occur as the result of a NEXT GROUP NEXT PAGE or LINE NEXT PAGE clause). Thus, the programmer should be aware of the following facts:

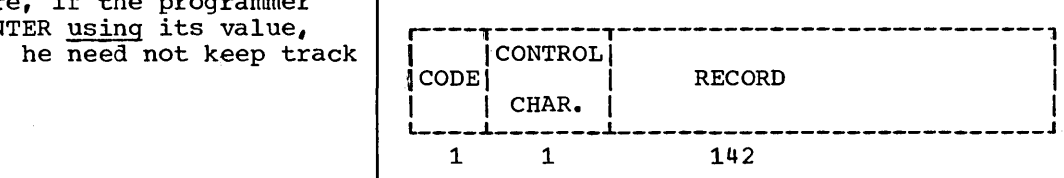
1. A Control Heading is not printed after a Page Heading except as a result of the first GENERATE statement. If the programmer wishes to have the equivalent of a Control Heading at the top of each page, he must include the information and data to be printed as part of the Page Heading. But since only one Page Heading may be specified for each report, he should be selective in considering his Control Heading because this Control Heading will be the same for each page, and may be printed at inopportune times (see "Control Footings and Page Format" in this chapter).

2. GROUP INDICATE items are not printed after page breaks unless a break has also occurred for the control item governing the GROUP INDICATE clause.

WITH CODE CLAUSE

When more than one report is being written on a file and the reports are to be selectively written, a unique 1-character code must be given for each report. (A mnemonic-name is specified in the RD-level entry for each report and is associated with the code in the Special-Names paragraph of the Environment Division.) This code will be written as the first character of each 144-character record that is written on the file. If the code character is omitted, reports cannot be selectively printed.

When the programmer wishes to write a report from this file, he needs merely to read a record, check the first character for the desired code, and have it printed if the desired code is found. The record will be printed starting from the third character, as is seen here:



The WRITE AFTER ADVANCING (CONTROL | CHAR) LINES feature must be used.

CONTROL FOOTINGS AND PAGE FORMAT

Depending on the number and size of Control Footings (as well as the page depth of the report), all of the specified Control Footings may not be printed on the same page if a control break occurs for a high-level control. When a page condition is detected before all required Control Footings are printed, the Report Writer will print the Page Footing (if specified), skip to the next page, print the Page Heading (if specified) and then continue to print Control Footings.

If the programmer wishes all of his Control Footings to be printed on the same page, he must format his page in the RD-level entry for the report (by setting the LAST DETAIL integer to a sufficiently low line number) to allow for the necessary space.

In order to use the System/360 Operating System Sort/Merge program, Sort Feature statements are written in the COBOL source program. These statements are described in the publication IBM System/360 Operating System: COBOL Language, Form C28-6516. The Sort/Merge program itself is described in the publication IBM System/360 Operating System: Sort/Merge Program, Form C28-6543. The chapter "Machine Considerations" in this publication contains information about system requirements when the Sort Feature is used.

DD statements must be written in the execution-time job steps of the procedure to describe the data sets used by the sort program. DD statements for data sets used during the sort process are described in the section "Sort DD Statements."

Note: The SORT/MERGE CHECKPOINT/RESTART feature is not available to the programmer who uses the COBOL SORT statement.

SORT DD STATEMENTS

Three types of data sets can be defined for the sort program in the execution time job step: input, output, and work. In addition, data sets must be defined for the use of the system during the sorting operation.

SORT INPUT DD STATEMENTS

When the USING option is specified in the SORT statement, the input data set must be associated with the ddname SORTIN. This name must appear as the external name in an ASSIGN clause in the COBOL source program.

When INPUT PROCEDURE is specified, a SORTIN DD statement is not required.

SORT OUTPUT DD STATEMENTS

When the GIVING option is specified in the SORT statement, the output data set must be associated with the ddname SORTOUT. This name usually appears as the external name in an ASSIGN clause in the COBOL source program, except in the special case where both USING and GIVING options are used and both refer to the same file-name. In this case, the external-name in the ASSIGN clause will be 'SORTIN'. See the section entitled "Sharing Devices Between Tape Data Sets" later in this chapter.

When the OUTPUT PROCEDURE option is specified, a SORTOUT DD statement is not required.

File Processing Technique for SORTIN and SORTOUT

When either SORTIN or SORTOUT is specified, the QSAM file processing technique must be used. QSAM is discussed in the chapter "User-Defined Files" and in the publication IBM System/360 Operating System: COBOL Language, Form C28-6516.

SORT WORK DD STATEMENTS

The sort program requires at least three work data sets. The ddname for each DD statement is in the form SORTWKnn, where nn is a decimal number. The ddnames for the required data sets must be SORTWK01, SORTWK02, and SORTWK03. Up to 29 other work data sets may be defined, but their ddnames must be consecutively numbered, beginning with 04.

SORTWKnn Data Set Considerations

Intermediate data sets (i.e., SORTWKnn data sets) for a sort may be assigned to either magnetic tape or direct-access devices. All of the intermediate storage for one sort must be assigned to the same device type. It may not be on both 7-track tape and 9-track tape units in the same sort. Any one of the following devices may be used as intermediate storage for a sort:

IBM 2400-series Magnetic Tape Unit (7-track)
IBM 2400-series Magnetic Tape Unit (9-track)
IBM 2311 Disk Storage Drive
IBM 2301 Drum Storage

The publication IBM System/360 Operating System: Sort/Merge Program, Form C28-6543, contains more detailed information about these devices.

When the COBOL program is not performing the sort operation, the space assigned to an intermediate data set may be used for a data set not involved in the Sort. For example, assume that in a program, an output data set is created after a sort operation is completed. That output data set can be assigned to the space that was used by a sort intermediate data set. Before the space assigned to an intermediate data set is reused, an OPEN statement must be issued to make the space available.

The same applies to an input data set not involved in the sort. Suppose a data set, DSA, is read at some point in the program before the sort takes place, and DSA is not to be used again. The space can later be made available for one of the intermediate sort data sets as long as it meets the space requirements for these data sets. After DSA is read, a CLOSE statement must be issued so that the space will be available for the sort operation. Note that if a data set is reused, the information for the first use is destroyed.

Two of the ways to reassign intermediate data set space are as follows:

1. An FD entry can be used in the COBOL source program to describe the characteristics of the data set that is not involved in the sort. An ASSIGN clause must be specified with the appropriate SORTWKnn ddname as an external name.

2. An extra DD statement is used with a ddname that is the external name in the ASSIGN clause. The UNIT parameter in one of the DD statements (usually the SORTWKnn statement) should specify an affinity to the other DD statement in the UNIT parameter, as follows:

   UNIT=AFF=ddname

   The AFF subparameter causes the system to assign to the data set the same unit occupied by the data set specified in the named DD statement, which must be in the same job step. For example, the source statement

   ...ASSIGN TO 'PAYROLL'....

   might be associated with the following DD statements:

```
//PAYROLL    DD  UNIT=2400,...
//SORTWK02   DD  UNIT=AFF=PAYROLL...
```

   This method can be used only to reassign intermediate data set space on tape.

DD STATEMENT REQUIREMENTS FOR SORT DATA SETS

The Sort Feature requires that certain parameters be included in the DD statements that define data sets used by the program. These parameters are summarized in Table 13.

Table 14 is a summary of the DCB subparameters that are required by the sort program if the DCB parameter is used. The information specified in the DCB subparameter should not contradict what is specified in the COBOL program.

The DCB subparameters are DEN, TRTCH, RECFM, LRECL, and BLKSIZE. The DEN and TRTCH subparameters are explained in the section "Processing with QSAM" in "User Defined Files."

The format of the RECFM subparameter is as follows:

```
RECFM= {FB}
       {VB}
```

F: Fixed-length records
V: Variable-length records
B: Blocked records

Note that B must be included in the RECFM subparameter even if no BLOCK CONTAINS clause is specified. In such a situation, records are considered to have a blocking factor of one.

For a format V record, add four bytes for the control field to the maximum record length. For format F, logical record length = size of record.

The BLKSIZE is determined as follows:

RECFM	BLKSIZE Value	Block Clause	Format
FB	integer * record length	RECORDS option	F
FB	integer	CHARACTERS option	F
VB	(integer * max record length) + (4* integer) + 4	RECORDS option	V
VB	integer + 4	CHARACTERS option	V

Table 13. Summary of DD Statement Parameters Required by the Sort Feature

Parameter	Condition under Which Required	Summary of Parameter Value	If Parameter Omitted
DSNAME	The DD statement defines a labeled input data set (e.g., SORTIN), or the data set being created is to be kept or cataloged (e.g., SORTOUT).	Specifies fully qualified name or temporary name of the data set.	System assigns a unique
DCB	The data set is used as the input or output data set (SORTIN or SORTOUT).	Specifies information required by the SORT/MERGE program.	----
UNIT	The input data set is neither cataloged nor passed, or the data set is being created.	Specifies (symbolically or actually) type and quantity of I/O units required by the data set.	----
SPACE	The DD statement defines a direct-access data set, and the data set is being created.	Specifies amount of space needed to contain the data set.	----
VOLUME	The input data set is neither cataloged nor passed, or the output data set is on direct access and is to be kept or cataloged.	Specifies information used to identify the volume or volumes occupied by the data set.	----
LABEL	The default value is not applicable.	Specifies information about labeling and retention for the data set.	System assumes standard labeling.
DISP	The default value is not applicable.	Indicates status and disposition of the data set.	System assumes a new data set and deletes it after job step ends.

Table 14. Summary of DCB Subparameters Required by the Sort Feature

Subparameter	Condition Under Which Required	Summary of Subparameter Value
DEN	The data set is located on a 2400-series tape unit, and is recorded in 7-track format.	Specifies density at which the tape is recorded.
TRTCH	The data set is located on a 2400-series tape unit, and is recorded in 7-track format.	Specifies technique used to record 8-bit bytes on a 7-track tape.
RECFM	The DCB parameter is required.	Specifies format of records in the data set. FB or VB is required.
LRECL	The DCB parameter is required. LRECL is optional.	Specifies maximum length (in bytes) of logical records in the data set.
BLKSIZE	The DCB parameter is required.	Specifies maximum length (in bytes) of physical records in the data set.

SORTIN DD Statement

For a sort, the SORTIN data set may be a previously created data set that may be cataloged or uncataloged. The SORTIN data set can be on 7-track tape when SORTWKnn data sets are on 9-track tape. The following example shows DD statement parameters that could be used to define a previously created and cataloged input data set:

```
//SORTIN   DD   DSNAME=INPT,                X
//               DISP=(OLD,DELETE),         X
//               DCB=(RECFM=FB,             X
//               BLKSIZE=800,LRECL=80)
```

These parameters cause the system to search the catalog for a data set named INPT (DSNAME parameter). When found, the data set is associated with the ddname SORTIN and used by the sort program. The control program obtains the unit assignment and volume serial number from the catalog, and displays a mounting message to the operator. The DISP parameter indicates that the data set has already been created (OLD). It also indicates that the data set should be deleted (DELETE) after the current job step. The DCB parameter indicates that the data set contains fixed-length blocked records (RECFM) with a physical block length of 800 bytes (BLKSIZE) and a logical record length of 80 bytes (LRECL).

SORTOUT DD Statment

The SORTOUT DD statement must define all of the characteristics of the output data set. The following example shows DD statement parameters that could be used to characterize an output data set:

```
//SORTOUT  DD   DSNAME=OUTPT,UNIT=2400,   X
//               DISP=(NEW,CATLG),         X
//               DCB=(RECFM=FB,            X
//               LRECL=90,BLKSIZE=900)
```

The DISP parameter indicates that the data set is unknown to the operating system (NEW) and that it should be cataloged (CATLG) under the name OUTPT (DSNAME parameter). The UNIT parameter specifies that the data set is on a 2400-series tape unit. The DCB parameter specifies a fixed-length blocked data set (RECFM) with a logical record length of 90 bytes (LRECL) and a physical block size of 900 bytes (BLKSIZE). The 9-track format is assumed.

SORTWKnn DD Statements

SORTWKnn data sets may be contained in tape or direct-access volumes. When direct-access space is assigned, only the primary allocation is used by the sort routine, and it must be contiguous.

SORTWKnn data sets:

1. May not be on 7-track tape when the SORTIN data set is on 9-track tape.
2. May be on 7-track tape when the SORTOUT data set is on 9-track tape.
3. Cannot use the data conversion feature if they are on 7-track tape. The TRTCH subparameter must reflect this.
4. May be on 9-track tape when the SORTIN data set is on 7-track tape.

SORTWKnn Example A: The following DD statement parameters could be used to define a tape intermediate storage data set:

```
//SORTWK01 DD   UNIT=2400,LABEL=(,NL)     X
//              VOLUME=SER=DUMMY
```

These parameters specify an unlabeled data set on a 2400-series tape unit. Since the DSNAME parameter is omitted, the system assigns a unique name to the data set. The omission of the DISP parameter causes the system to assume that the data set is new and that it should be deleted at the end of the current job step. The 2400-series tape units are explicitly of the 9-track format.

SORTWKnn Example B: The following DD statement parameters could be used to define a direct-access intermediate storage data set:

```
//SORTWK01 DD   UNIT=2311,                 X
//              SPACE=(TRK,(200),,CONTIG),
```

These parameters specify a disk (2311) data set with a standard label (LABEL parameter default value). The SPACE parameter specifies that the data set is to be allocated 200 contiguous tracks. The system assigns a unique name to the data set and deletes it at the end of the job step.

SYSTEM DD STATEMENTS

In addition to the DD statements used by the SORT program itself, the following must be included in the execution time job step:

```
//SYSOUT   DD   SYSOUT=A
//SORTLIB  DD   DSNAME=SYS1.SORTLIB,      X
//             DISP=OLD
```

These DD statements are for the use of other components of the system that are concerned with the sorting operation.

Note: At system generation time, SORT diagnostic messages can be assigned to print either on the console or on the unit designated SYSOUT. If the system is generated to write SORT messages on SYSOUT, these messages may overprint any COBOL output assigned to SYSOUT. For example, if the programmer has selected SYSOUT on which to print a report in the output procedure of a COBOL SORT program, any SORT messages will be interspersed within that report. If it is not possible to assign the SORT messages to the console, the programmer should assign his COBOL output to temporary files and print the reports at a later time.

SHARING DEVICES BETWEEN TAPE DATA SETS

A single tape unit may be assigned to two sort data sets when the data sets are one of the following pairs:

- The input data set (SORTIN) and the first intermediate storage data set (SORTWK01).
- The input data set (SORTIN) and the output data set (SORTOUT).

The AFF subparameter of the UNIT parameter can be used to associate the SORTIN data set with either the SORTWK01 data set or the SORTOUT data set. It can appear in the DD statement for SORTWK01 or SORTOUT.

To use the Sort Feature with both the USING and GIVING options, where input and output are on the same tape unit, SORTOUT does not need to be defined in a SELECT... ASSIGN statement, provided that SORTIN is so defined. Only one FD entry is required, and that file-name will appear as the object of both the USING and GIVING clauses. (In this case, a SORTOUT DD statement must be present at execution time specifying UNIT=AFF with the SORTIN tape unit.)

USING MORE THAN ONE SORT STATEMENT IN A JOB

More than one SORT statement may be used in a single program or in two or more programs that are combined into a single load module. If more than one USING clause is specified, all of the files named in the USING clauses must be associated with the SORTIN DD statement. However, only one SORTIN DD statement can be specified in a single job step. Therefore, if more than one file is to be sorted during execution of a single job step, the INPUT PROCEDURE option can be used instead, and the file to be sorted need not be associated with SORTIN.

Similarly, the OUTPUT PROCEDURE option can be used in place of the GIVING option.

SORT PROGRAM EXAMPLE

The following control cards could be used with the sample Sort feature program contained in the publication IBM System/360 Operating System: COBOL Language, Form C28-6516:

```
//SORTEST      JOB   NY83870165,'J.SMITH',X
//                   MSGLEVEL=1
//SORTJS3      EXEC  COBFCLG
//COB.SYSIN    DD    *
                .
                .
                .
     (COBOL source program)
                .
                .
                .
//GO.SORTWK01  DD    UNIT=SYSDA,SPACE=     X
//                   (TRK,(200),,CONTIG)
//GO.SORTWK02  DD    UNIT=SYSDA,SPACE=     X
//                   (TRK,(200),,CONTIG)
//GO.SORTWK03  DD    UNIT=SYSDA,SPACE=     X
//                   (TRK,(200),,CONTIG)
//GO.SORTOUT   DD    UNIT=183,LABEL=(,NL),X
//                   VOLUME=SER=NONE,      X
//                   DCB=(RECFM=FB,        X
//                   LRECL=72,             X
//                   BLKSIZE=72)
//GO.SYSOUT    DD    SYSOUT=A
//GO.SORTLIB   DD    DSNAME=SYS1.SORTLIB,  X
//                   DISP=OLD
//GO.SYSIN     DD    UNIT=182,LABEL=(,NL),X
//                   VOLUME=SER=DUMMY
```

In the GO.SYSIN and GO.SORTOUT DD statements the numbers in the UNIT parameters are names assigned to tape units in a specific installation and are not standard names. The minimum number of SORTWKnn data sets are used; the sort operation can be optimized by using additional work data sets (see the publication IBM System/360 Operating System: Sort/Merge, Form C28-6543).

CATALOGING SORT DD STATEMENTS

Since repeated use of the Sort Feature often involves the same execution time DD statements, the user may wish to catalog them (see "Cataloged Procedures").

SORT DIAGNOSTIC MESSAGES

The messages generated by the Sort Feature are listed in the publications IBM System/360 Operating System, Sort/Merge, Form C28-6543, and IBM System/360 Operating System: Messages, Completion Codes, and Storage Dumps, Form C28-6631. The identifying characters in a sort message are IER.

LINKAGE WITH THE SORT/MERGE PROGRAM

Communication between the Sort/Merge program and the COBOL program is maintained by the COBOL library subroutine IHDFSORT.

If the INPUT PROCEDURE option of the SORT statement is specified, exit E15 of the Sort/Merge program is used. The return code indicating "insert records" is issued when a RELEASE statement is encountered, and the return code indicating "do not return" is issued when the end of the procedure is encountered.

If the OUTPUT PROCEDURE option is specified, exit E35 of the Sort/Merge program is used. The return code indicating "delete records" is issued when a RETURN statement is encountered, and the return code indicating "do not return" is issued when the end of the procedure is encountered.

Completion Codes

The Sort/Merge program returns a completion code upon termination. This code may be interrogated by the COBOL program. The codes are:

```
 0 - Successful completion of Sort/
     Merge
16 - Unsuccessful completion of Sort/
     Merge
```

SUCCESSFUL COMPLETION: When a Sort/Merge application has been successfully executed, a completion code of zero is returned and the sort terminates.

UNSUCCESSFUL COMPLETION: If the sort, during execution, encounters an error that will not allow it to complete successfully, it returns a completion code of 16 and terminates. (Possible errors include an out-of-sequence condition or an uncorrectable I-O error.) The publication IBM System/360 Operating System: Sort/Merge, Form C28-6543 contains a detailed description of the conditions under which this termination will occur.

The returned completion code will be stored in a special register called TALLY by the COBOL library subroutine; an unsuccessful termination of the sort may then be tested for and appropriate action specified. Note that the contents of TALLY will change with the execution of a SORT statement. The following is an example of the use of TALLY with the sort feature:

```
SORT SALES-RECORDS ON ASCENDING KEY
CUSTOMER-NUMBER, DESCENDING KEY DATE,
USING FN-1, GIVING FN-2.

IF TALLY NOT EQUAL TO ZERO, DISPLAY
'SORT UNSUCCESSFUL' UPON CONSOLE, STOP
RUN.
```

Libraries are an integral part of the operating system. Some libraries have system-supplied names and system-supplied data. Other libraries have system-supplied names, but the data they contain may be specified by a user. Still other libraries have user-supplied names and user-supplied data.

Libraries, in general, are made up of partitioned data sets. Any library with a user-supplied name and user-supplied data always is a single partitioned data set, which is a collection of independent sets of sequentially organized data, called members. All of the members within a partitioned data set have the same characteristics such as record format. When used to store programs, a partitioned data set containing load modules can contain only load modules; it cannot contain both load modules and object modules.

Each partitioned data set is headed by a directory of entries pointing to the members that make up the library. Each member has a unique member name. A partitioned data set must reside on a single direct-access device, but some libraries can consist of a concatenation of more than one partitioned data set.

Figure 37 shows the format of a library that is a single partitioned data set of four members. Space for the members of such a library and its directory is requested in the SPACE parameter of the DD statement when the library is created. Additional members can be added to a library at a later time. If additional space is required to store a member, allocation will be made in the amount specified by the secondary allocation in the SPACE parameter of the DD statement that was used when the library and its first member were created. Additional space cannot be allocated for the directory, however. Directory space is allocated for the entire library when the library is created. If the original allocation was not large enough, the IEHMOVE utility program can be used to expand the directory size. If the directory is filled, no additional members can be added to the library. Following is an example of a DD statement that might be used to create a library:

```
//DD1       DD DSNAME=FILELIB(FILE1),      X
//               DISP=(NEW,CATLG),          X
//               UNIT=2311,                 X
//               SPACE=(TRK,(40,10,3)),     X
//               VOLUME=SER=111111
```

This statement specifies that a library named FILELIB is to be created and cataloged in this job step. Its first member is named FILE1. Initial space allocated for data sets is to be 40 tracks, with additional allocation to be made, as necessary, in units of 10 tracks. In addition, space for three 256-byte records is to be allocated for the directory. The volume serial number is 111111.

A member of a partitioned data set can be replaced or deleted. The system actually accomplishes this by modifying or deleting the directory pointer to the member. The space occupied by the original member is not available for reuse. However, the MOVE or COPY control statement of the IEHMOVE utility program can be used to make available the space previously occupied by the replaced or deleted member. (For further details, see IBM System/360 Operating System: Utilities, Form C28-6586.)

KINDS OF LIBRARIES

A programmer can use libraries already provided by the system, or he can create libraries of his own. In addition, certain library names recognized by the system may be assigned to partitioned data sets provided by the system, by the programmer, or both. These libraries and their uses are discussed in the following paragraphs.

LIBRARIES PROVIDED BY THE SYSTEM

Link Library

The link library is a partitioned data set that contains load modules to be executed. Unless specified otherwise, a load module name in an EXEC statement is to be fetched from the link library. Operating system programs, such as the COBOL compiler, are usually contained in this library.

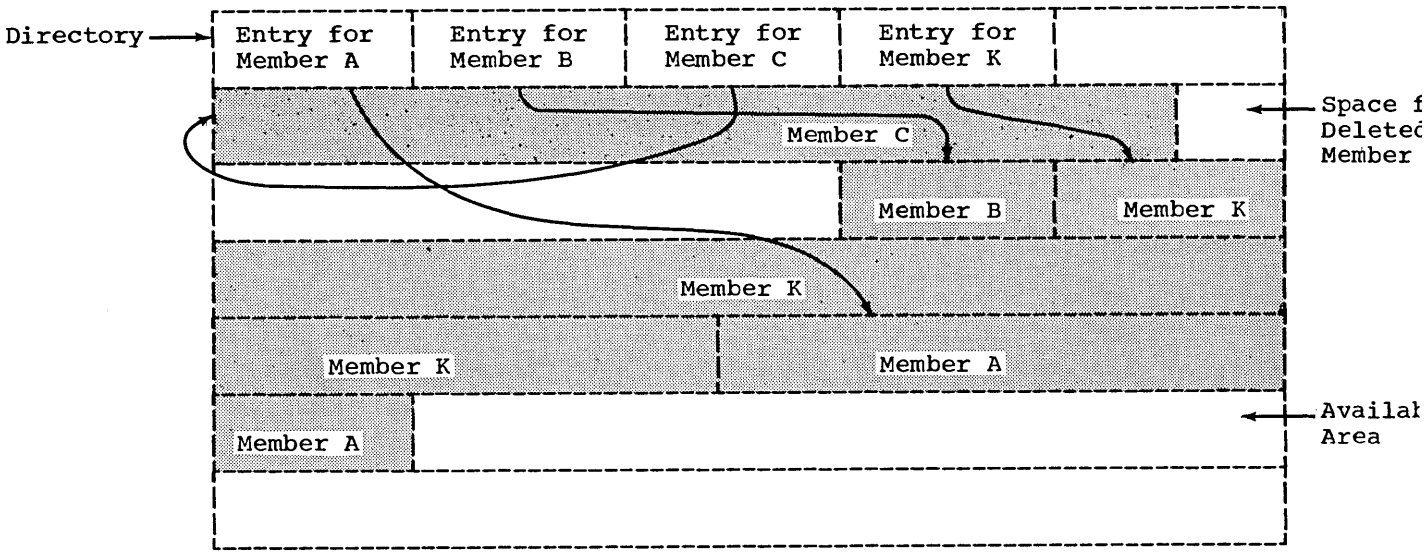


Figure 37. Format of a Library

The link library can be used by the programmer to store executable load modules at linkage editor time. The technique for doing this is described in "Linkage Editor Data Set Requirements."

The link library is identified in a job control statement as SYS1.LINKLIB.

Procedure Library

The procedure library is a partitioned data set whose members are the cataloged procedures at an installation. They include the cataloged procedures provided by IBM. Procedures written at the installation can be added to the procedure library with the IEBUPDTE utility program (see "Cataloged Procedures").

The system name for the procedure library is SYS1.PROCLIB.

Sort Library

The sort library is a partitioned data set that contains load modules from which the sort program is produced.

It is identified by the name SYS1.SORTLIB (see "Using the Sort Feature").

COBOL Subroutine Library

The COBOL subroutine library is a partitioned data set that contains the COBOL library subroutines in load module form. These subroutines are included in a COBOL load module to perform such functions as data conversion and double precision arithmetic. The COBOL programmer does not refer directly to these subroutines; calling sequences to them are generated at compile time from certain Procedure Division statements, and they are incorporated into the load module at linkage editor time. A listing of subroutine names, functions, entry points, and size is given in Appendix B.

The system name for the COBOL subroutine library is SYS1.COBLIB.

OTHER LIBRARIES RECOGNIZED BY THE SYSTEM

AUTOMATIC CALL LIBRARY

The automatic call library, defined by the SYSLIB DD statement in the linkage editor job step, contains load modules or object modules that may be used as secondary input to the linkage editor. If the library contains object modules, it may also contain control statements. External symbols that are undefined after all primary input has been processed cause the automatic library call mechanism to search the automatic call library for modules that will resolve the references. The COBOL subroutine library must be specified for the automatic call library if any of the subroutines will be needed to resolve external references. Other partitioned data sets may be concatenated as shown in the following example:

```
//SYSLIB   DD DSNAME=SYS1.COBLIB,DISP=OLD
//         DD DSNAME=MYLIB,DISP=OLD
```

In this case, both the COBOL subroutine library and the partitioned data set named MYLIB are available to the automatic library call.

Note: If the partitioned data set named in the SYSLIB DD statement contains load modules, any data set concatenated with it must also be a load module partitioned data set. If the first contains object modules, the others must also contain object modules.

The linkage editor LIBRARY control statement has the effect of concatenating any specified member names with the automatic call library.

COBOL COPY LIBRARY

The COBOL copy library is a user-created library consisting of statements or entire COBOL programs frequently used by the programmer. The programmer can include these statements or programs into a program at compile time. He calls them with the COBOL clauses COPY, INCLUDE, or BASIS.

To enter or update source statements in the copy library, a utility program must be used. IEBUPDTE is the IBM-supplied utility program used to do this. A full discussion of the statements used in this program may be found in the publication, IBM System/360 Operating System: Utilities, Form C28-6586.

Entering Source Statements

Figure 38 illustrates the method to insert source statements into a copy library member.

The utility statement ./ ADD copies CFILEA into the library called COPYLIB. CFILEA describes an FD entry. The NUMBER statement assigns a sequential numbering system to the statements in the library. The first statement is assigned number 10. and each succeeding statement is incremented by 5. The entries following the utility statements are the actual source statements to be inserted into the number. The ENDUP statement signals the end of the entries to be inserted.

This same procedure can be used to include entire source programs into a library.

Updating Source Statements

Figure 39 illustrates the method to update source statements in the copy library member inserted in the previous example.

```
//CATALOG          JOB
//                 EXEC   PGM=IEBUPDTE,PARM=(NEW)
//SYSUT2           DD     DSNAME=COPYLIB,UNIT=2311,                    X
//                            DISP=(NEW,KEEP),                        X
//                            VOLUME=SER=111111,                      X
//                            SPACE=(TRK,(15,10,2)),                  X
//                       DCB=(,RECFM=F,BLKSIZE=80)
//SYSPRINT         DD     SYSOUT=A
//SYSIN            DD     *
./                 ADD    NAME=CFILEA,LEVEL=00,SOURCE=0,LIST=ALL
./                 NUMBER NEW1=10,INCR=5
                     BLOCK CONTAINS 13 RECORDS
                     RECORD CONTAINS 120 CHARACTERS
                     LABEL RECORDS ARE STANDARD
                     DATA RECORD IS FILE-OUT.
./                 ENDUP
/*
```

• Figure 38. Entering Source Statements Into the Copy Library

```
//UPDATE          JOB
//                EXEC   PGM=IEBUPDTE,PARM=(MOD)
//SYSUT1          DD     DSNAME=COPYLIB,UNIT=2311,                        X
//                           DISP=(OLD,KEEP),                             X
//                           VOLUME=SER=111111,                           X
//                           SPACE=(TRK,(15,10,2)),                       X
//                           DCB=(,RECFM=F,BLKSIZE=80)
//SYSUT2          DD     DSNAME=COPYLIB,UNIT=2311,                        X
//                           DISP=(OLD,KEEP),                             X
//                           VOLUME=SER=111111,                           X
//                           SPACE=(TRK,(15,10,2)),                       X
//                           DCB=(,RECFM=F,BLKSIZE=80)
//SYSPRINT        DD     SYSOUT=A
//SYSIN           DD     *
./                CHANGE NAME=CFILEA,LEVEL=01,SOURCE=0,LIST=ALL
                    BLOCK CONTAINS 20 RECORDS                      00000010
./                ENDUP
/*
```

• Figure 39. Updating Source Statements in a Copy Library

SYSUT1 and SYSUT2 describe the old and the new data sets. Note that changes may be made on the same data set (identified by the DSNAME parameter). The utility statement CHANGE indicates that the new entry of CFILEA replaces the old entry. The sequence number of the altered statement must be supplied. This number, 00000010, is in dicated in columns 73-80 of the replacement source statement. Note that, although in the insert example (Figure 38, the NUMBER statement) the number was coded as 10 without leading zeros, the program assigns an 8-character field to a sequence number and pads with leading zeros if necessary. When updating a sequence number in a library, these leading zeros must be included.

At compile time, COPYLIB is identified on a SYSLIB DD statement, as follows:

```
//SYSLIB  DD   DSNAME=COPYLIB,              X
//               VOLUME=SER=111111,         X
//               DISP=OLD, UNIT=2311
```

Retrieving Source Statements

Members of the cataloged library can be retrieved using the COPY, INCLUDE, or BASIS clauses.

The COPY Clause

The COPY clause permits the programmer to include members of a library containing source statements into the Data or Environment Divisions. If the programmer wishes to retrieve the member, CFILEA, inserted in the previous examples, he writes the statement:

```
FD FILEA COPY 'CFILEA'
```

The compiler translates this instruction to read:

```
FD FILEA BLOCK CONTAINS 20 RECORDS
   RECORD CONTAINS 120 CHARACTERS
   LABEL RECORDS ARE STANDARD
   DATA RECORD IS FILE-OUT.
```

Note that CFILEA itself does not appear in the statement. CFILEA is a name identifying the entries. It acts as a header record but is not itself retrieved. The compiler source listing, however, will print out the COPY statement as the programmer wrote it.

The INCLUDE Clause

The INCLUDE clause permits the programmer to include members of a library containing source statements into the Procedure Division. It is implemented like the COPY clause.

Assume a procedure named DOWORK was inserted with the following statements:

```
./   ADD          NAME=DOWORK,LEVEL=00,
                    SOURCE=0,LIST=ALL
./   NUMBER       SEQ1=400,INCR=10
       COMPUTE QTY-ON-HAND = TOTAL-USED-
         NUMBER-ON-HAND.
       MOVE-QTY-ON-HAND TO PRINT-AREA.
./   ENDUP
```

To retrieve the member, DOWORK, the programmer writes:

```
Paragraph-name.  INCLUDE 'DOWORK'.
```

The statements included in the DOWORK procedure will immediately follow the paragraph-name, replacing the words INCLUDE 'DOWORK'.

The BASIS Clause

Frequently used source programs, such as a payroll program, can be inserted into the copy library. The BASIS clause brings in an entire source program at compile time. Calling in a cataloged program eliminates the need for the programmer to handle a program each time he wants to compile one. The programmer may, however, alter any statement in the source program by referring to its COBOL sequence number with an INSERT or DELETE statement. The INSERT statement adds new source statements after the sequence number indicated. The DELETE statement deletes the statements indicated by the sequence numbers. The programmer may delete a single statement with one sequence number, or may delete more than one statement with the first and last sequence numbers to be deleted separated by a hyphen.

Note: The COBOL sequence number is the 6-digit number which the programmer assigns in columns 1-6 of the source cards. This sequence number has no connection with the sequence numbers assigned in simulated columns 73-80 by the IEBUPDTE utility program. The sequence numbers assigned by IEBUPDTE are used to update source statements in the copy library, and are intended to be permanent changes. The COBOL sequence numbers are used to update COBOL source statements at compile time and are in effect for the one run only.

Assume that a company payroll program is kept as a source program member in from a copy library. The name of the program is PAYROLL. During the year, social security tax (FICA) is taken out at a rate of 4.40 percent each week for all personnel until earnings exceed $6600. The coding to accomplish this is shown in Figure 40.

Now, however, due to a change in the Social Security laws, FICA is to be taken out until earnings exceed $7800. Assume also that at this time the company is sponsoring a drive for contributions to the local children's aid fund. All employees have been asked to contribute on a voluntary basis. Each employee record in the input data set contains two fields, OTHER-CODE and OTHER-DOLLAR, to handle various contributions. If OTHER-CODE is coded Y, the employee has agreed to contribute the dollar amount shown in OTHER-DOLLAR. The programmer can make these changes as follows:

```
//GO.SYSIN  DD  *
BASIS 'PAYROLL'
DELETE 000730
      IF ANNUAL-PAY GREATER THAN 7800 GO TO
        PAY-WRITE.
INSERT 000770
      IF CONTRIB='Y' GO TO FUND-PAY.
INSERT 000850
   FUND-PAY.  SUBTRACT OTHER-DOLLAR FROM
                BASE-PAY.
              MOVE OTHER-DOLLAR TO
                OUTPUT-OTHER.
   EX1.  EXIT.
```

The altered program will contain the coding shown in Figure 41.

COBOL Sequence Numbers			IEBUPDTE Sequence Numbers
000730		IF ANNUAL-PAY GREATER THAN 6600 GO TO PAY-WRITE.	00000110
000740	FICA-PAYR.	COMPUTE FICA-PAY = BASE-PAY * .044	00000115
000750		MOVE FICA-PAY TO OUTPUT-FICA.	00000120
000760	PAY-WRITE.	MOVE BASE-PAY TO OUTPUT-BASE.	00000125
000770		ADD BASE-PAY TO ANNUAL-PAY.	00000130
.		.	.
.		.	.
.		.	.
000850		STOP RUN.	00000240

Figure 40. COBOL Statements to Deduct FICA Tax.

```
COBOL                                                                  IEBUPDTE
Sequence                                                               Sequence
Numbers                                                                Numbers
I                          IF ANNUAL-PAY GREATER THAN 7800 GO TO
                               PAY-WRITE.
000740        FICA-PAYR.   COMPUTE FICA-PAY = BASE-PAY * .044          00000115
000750                     MOVE FICA-PAY TO OUTPUT-FICA.               00000120
000760        PAY-WRITE.   MOVE BASE-PAY TO OUTPUT-BASE.               00000125
000770                     ADD BASE-PAY TO ANNUAL-PAY.                 00000130
I                          IF CONTRIB='Y' GO TO FUND-PAY.
   .                       .                                            .
   .                       .                                            .
   .                       .                                            .
000850                     STOP RUN.                                   00000240
I             FUND-PAY.    SUBTRACT OTHER-DOLLAR FROM BASE-PAY.
I                          MOVE OTHER-DOLLAR TO OUTPUT-OTHER.
I             EX1.         EXIT.
```

• Figure 41. Changed COBOL Statements to Update FICA Example.

Note that changes made through use of the INSERT and DELETE statements remain in effect for the one run only.

New statements brought into the program by INSERT cards are identified by the letter I in the COBOL sequence number field.

Note: If BASIS and COPY are both used, the library containing the member specified in the BASIS card must be defined first. The COPY libraries concatenated with the BASIS library may be defined and referenced in any order. See the section "A Check List for Job Control Procedures" for an example.

JOB LIBRARY

The job library consists of one or more partitioned data sets which contain load modules to be executed. It is specified by the JOBLIB DD statement which must precede the EXEC statement of the first step of a job. Partitioned data sets assigned to the job library are concatenated with the link library so that any load module is obtained automatically when its name appears in the PGM= parameter of the EXEC statement. The following statements illustrate how three partitioned data sets can be assigned to the job library:

```
//MYJOB   JOB ...
//JOBLIB  DD  DSNAME=MYLIB1,DISP=(OLD,PASS)
//        DD  DSNAME=MYLIB2,DISP=(OLD,PASS)
//        DD  DSNAME=MYLIB3,DISP=(OLD,PASS)
//STEP1   EXEC ...
          .
          .
          .
//STEP2   EXEC ...
```

These statements specify that the job library containing the data sets MYLIB1, MYLIB2, and MYLIB3 is to be concatenated with the link library. When a load module is named in an EXEC statement in any step of the job, the directories of the job library will be searched for the name. When a job library is specified for a job, the link library is searched for a named load module only if the module is not found in the job library.

Partitioned data sets used in the job library can be created by specifying the partitioned data set name and the member name in the SYSLMOD DD statement when each member is processed by the linkage editor.

LIBRARIES CREATED BY THE USER

As discussed previously, a programmer can create members of the link library, the procedure library, and the job library. He can also create partitioned data sets for use in the copy library, the automatic call library, and the job library. In addition, he can create partitioned data sets to be used as libraries for additional input to the linkage editor, and he can create libraries whose members are source program entries.

Additional Input to Linkage Editor

Libraries of object modules (with or without linkage editor control statements) and libraries of load modules can be used as additional input to the linkage editor. Members are specified by use of the INCLUDE and LIBRARY linkage editor control statements.

A library of object modules and control statements can be created by use of the IEBUPDTE utility program.

A library of load modules can be created by use of the SYSLMOD DD statement in the linkage editor job step, as discussed in "Job Library."

CREATING AND CHANGING LIBRARIES

A programmer can create or change a partitioned data set in one of three ways: (1) through the use of DD statements, (2) through the use of utility programs, and (3) through the use of certain linkage editor control statements.

The DD statement can be used to create libraries as is discussed at the beginning of this chapter. In addition, DD statements can be used to add members to existing libraries, including the link library, to retrieve members of existing libraries.

Utility programs can be used to create libraries such as those used in the copy library or as secondary input to the linkage editor. In addition, utility programs can be used to move, copy, and replace members of an existing library; to add, delete, and renumber the records within an existing library; and to assign sequence numbers to the records of a new library.

Linkage editor control statements can be used to make changes to members of a library of load modules. The name of a member can be changed or additional names can be specified. Additional entry points can be identified, existing entry points can be deleted, and portions of a load module can be deleted or replaced. For further information, see IBM System/360 Operating System: Linkage Editor, Form C28-6538.

A COBOL program can refer to and pass control to other COBOL programs, or to programs written in other languages. A program in another language can refer to and pass control to a COBOL program. A program that refers to another program is a calling program. A program that is referred to is a called program. Control is returned from a called program to the first instruction following the calling sequence in the calling program.

A called program can also be a calling program; that is, a called program can, in turn, call another program. However, a called program can not call the program that called it or an earlier calling program. Control is returned in the same order of calling; that is, a called program usually returns control to its own calling program, not to an earlier calling program. Computer generated switches, e.g., ON and ALTER, are reinitialized upon each entrance to the called program.

All called and calling programs to be executed as a single job step must be linkage edited together; they must all be included in the same load module.

This chapter describes the accepted linkage conventions for calling and called programs in both COBOL and assembler language and discusses how such programs are linkage edited. In addition, it includes a discussion of overlay design in which different called programs may, at different times, occupy the same area in main storage.

SPECIFYING LINKAGE

Whenever a program calls another program, a link must be established between the two. The calling program must state the entry point of the called program and must specify any arguments to be passed. The called program must have an entry point and must be able to accept the arguments. In addition, the called program must establish the linkage for the return of control to the calling program.

LINKAGE IN A CALLING COBOL PROGRAM

A calling COBOL program must contain the following statements at the point where another program is to be called:

```
ENTER LINKAGE.
CALL entry-name  [USING argument-list].
ENTER COBOL.
```

The entry-name is the name of the entry point in the called program to which control is to be transferred. The argument list is one or more data-names, separated by blanks, that are to be passed to the called program.

If the called program is an assembler language program, the argument list may also include file-names and procedure-names. If the argument contains a file-name, the COBOL (F) compiler passes the address of the DCB (data control block) for a queued file, or the address of the DECB (data event control block) for a basic file, as an entry of the argument list. This can be used to test bits in the DCB or DECB or to enter some options in the DCB by a called assembler language subroutine. However, when changing a field of the DCB, precautions should be taken not to contradict the information in other fields or the information in the object codes supplied by the compiler, job control language, or other sources. If no arguments are passed, the USING clause is omitted.

LINKAGE IN A CALLED COBOL PROGRAM

A called COBOL program must contain two sets of statements.

The following statements must be inserted to name the point where the program is to be entered:

```
ENTER LINKAGE.
ENTRY entry-name [USING parameter-list].
ENTER COBOL.
```

The entry-name is the name of the entry point in the called program. It is the same name that appears in the CALL statement of the program that calls this program. The parameter list is one or more data-names that correspond to the arguments of the CALL statement of the calling program. Each data-name of the parameter list must be defined in the Linkage Section of the Data Division and must have a level number of 01 or 77.

The following statements must be inserted at the point where control is to be returned to the calling program:

```
ENTER LINKAGE.
RETURN.
ENTER COBOL.
```

The RETURN statement enables restoration of necessary registers and it returns control to the point in the calling program immediately following the calling sequence.

Correspondence of Arguments and Parameters

The number of data-names in the argument list of a calling program must be the same as the number of data-names in the parameter list of the called program. There is a one-for-one correspondence; that is, the first argument is passed to the first parameter, the second argument to the second parameter and so forth.

Only the address of an argument is passed. Consequently, the data-name that is an argument and the data-name that is the corresponding parameter both refer to the same locations in main storage. The pair of names, however, need not be identical, but the data descriptions must be equivalent. For example, if an argument is a 77 level data-name of a character string of length 30, its corresponding parameter also could be a 77 level data-name of a character string of length 30, or the parameter could be a 01 level name with subordinate names representing character strings whose combined length is 30.

Although all parameters in the ENTRY statement must be described with level numbers of 01 or 77, there is no such restriction made for arguments in the CALL statement. An argument may be a qualified name or a subscripted name. When a group item with a level number other than 01 is specified as an argument, proper word-boundary alignment is required if subordinate items are described as COMPUTATIONAL, COMPUTATIONAL-1, or COMPUTATIONAL-2. If the argument corresponds to a 01-level parameter, doubleword alignment is required.

LINKAGE IN A CALLING OR CALLED ASSEMBLER-LANGUAGE PROGRAM

In a COBOL program, the expansions of the ENTER LINKAGE statement provide the save and return coding that is necessary to establish linkage between the calling and the called programs. Assembler language programs must be prepared in accordance with the basic linkage conventions of the operating system. Table 15 shows the conventions for use of general registers as linkage registers.

Conventions Used in a Calling Assembler Language Program

A calling assembler language program must reserve a save area of 18 words, beginning on a fullword boundary, to be used by the called program for saving registers. It must load the address of this area into register 13. If the program is to pass arguments, an argument list must be prepared, and the address of the argument list must be loaded into register 1. The calling program must load the address of the return point into register 14, and it must load the address of the entry point of the called program into register 15.

Table 15. Linkage Registers

Register Number	Register Use	Contents
1	Argument Register	Address of the argument list that is passed to the called program.
13	Save Area Register	Address of an area (of 18 full words) to be used by the called program to save registers.
14	Return Register	Address of the location in the calling program to which control should be returned after execution of the called program.
15	Entry Point Register[1]	Address of the entry point in the called program to which control is to be transferred.

[1]Register 15 also is used as a return code register. The return code indicates whether or not any exceptional conditions occurred during execution of the called program. A COBOL program connot generate a return code, nor can a COBOL program test a return code returned to it from an assembler language program. A return code can be used, however, if both the calling and called programs have been written using the assembler language.

```
         .
         .
         .
      (Calling Sequence)

         LA     1,ARGLST     Loads into register 1 the address of the argument list to
                               be passed.

         LA     13,AREA      Loads into register 13 the address of the save area to be
                               passed.

         CALL   CALLED       Transfers control to the entry point of the called
         .                     program. (The CALL macro-instruction generates coding
         .                     that loads a V-type address constant of CALLED into
         .                     register 15 and that places into register 14 the return
                               address, that is, the address of the first byte
                               following the macro-expansion.)

      (Save Area)

AREA     DC     18F          Reserves the save area of 18 full words to be passed to
                               the called program.

      (Argument List)

ARGLIST  DC     A(ARG1,ARG2) Creates address constants for the first two arguments of
                               the argument list (called ARG1 and ARG2).

         DC     X'80'        Sets the first bit of the next word to one.

         DC     AL3(ARG3)    Creates an address constant for this argument and stores
                               it in the last three bytes of the word. Since the
                               first bit of the first byte of the word is a one, this
                               third argument is specified to be the last argument of
                               the list.
```

Note: Since the calling program containing this coding would have previously been called by the COBOL program, it also could establish linkage between the save area it has received and the save area it passes to the called program. It would store in word three of the old save area the address of the new save area, and it would store in word two of the new save area the address of the old save area.

Figure 42. Sample Linkage Coding Used in a Calling Assembler-Language Program

The argument list is a group of contiguous fullwords, each of which is an address of a data item to be passed to the called program. The argument list must begin on a fullword boundary. The high-order bit of the last argument, by convention, is set as a flag of 1 to indicate the end of the list. Figure 42 shows a portion of an assembler language program that illustrates the conventions used in a calling program.

Conventions Used in a Called Assembler Language Program

A called assembler language program must save the registers and store other pertinent information in the save area passed to it by the calling program (the layout of the save area is shown in Figure 43). A called program must also contain a return routine that (1) loads the address of the save area back into register 13, (2) restores the contents of other registers, (3) loads the return address in register 14, and (4) optionally sets flags in the high-order eight bits of word 4 of the save area to all ones to indicate that the return occurred. It can then branch to the address in register 14 to complete the return.

Figure 44 shows a portion of an assembler language program that illustrates the conventions used in called programs.

Word No.	Area No.	Contents
Word 1	AREA	Not used by COBOL or assembler-language programs.
Word 2	AREA +4	Address (stored by the calling program) of the save area used by the calling program. This is the address of a save area that was passed to the calling program by the program that called the calling program.
Word 3	AREA +8	Address (stored by the called program) of the next save area, that is, the save area that the called program provides for a program that it calls. The called program need not reserve a save area if it does not, in turn, call another program.
Word 4	AREA +12	Return address (contents of register 14) stored by the called program.
Word 5	AREA +16	Entry point address (contents of register 15) stored by the called program.
Word 6	AREA +20	Contents of register 0 (stored by the called program).
Word 7	AREA +24	Contents of register 1 (stored by the called program); that is, the address of the argument list passed to the called program.
Word 8 through Word 18	AREA +28 AREA +68	Contents of registers 2 through 12 (stored by the called program).

Figure 43. Save Area Layout and Contents

FILE-NAME AND PROCEDURE-NAME ARGUMENTS

A calling COBOL program that calls an assembler language program can pass file-names and procedure names, in addition to data-names, as arguments. In the actual argument list that the compiler generates, the procedure-name is passed as the address of the procedure. For a queued file, the file-name is passed as the address of the DCB; for a basic file, the file-name is passed as the address of the DECB.

LINKAGE EDITING PROGRAMS

Each time an entry point is specified in a called program, an external name is defined. An external name is a name that can be referred to by another separately compiled or assembled program. Each time an entry name is specified in a calling program, an external reference is defined. An external reference is a symbol that is defined as an external name in another separately compiled or assembled program. The linkage editor resolves external names and references and combines calling and called programs into a format suitable for execution together, i.e., as a single load module.

Load modules of both calling and called programs are used as input to the linkage editor. There are two kinds of input, primary and additional. Primary input consists of a sequential data set that contains one or more separately compiled object modules and/or linkage editor control statements. The primary input can contain object modules that are either calling or called programs or both. Additional input consists of object modules or load modules that are not part of the primary input data set but are to be included in the load module. The additional input may be in the form of (1) a sequential data set consisting of one or more object modules with or without linkage editor control statements, or (2) libraries containing object modules with or without linkage editor control statements, or (3) libraries consisting of load modules. The additional input is specified by linkage editor control statements in the primary input and a DD statement for each additional input data set. Additional input may contain either calling or called programs or both.

Note: Each additional input data set may itself contain external references or names and linkage editor control statements that specify more additional input.

SPECIFYING PRIMARY INPUT

The primary input data set is specified for linkage editor processing by the SYSLIN DD statement. The linkage editor must always have a primary input data set specified by a SYSLIN DD statement whether or not there are called or calling programs and even if the primary input data set contains only linkage editor control statements. The SYSLIN DD statement that specifies the primary input is discussed in the chapter "Linkage Editor Data Set Requirements." See "Example of Linkage Editor Processing" for a discussion of how to specify a primary input data set that contains more than one object module along with linkage editor control statements.

```
        .
        .
        .
        ENTRY   CALLED      Establishes CALLED as an external name that can be
                              referred to in another program.

        (Save Routine)

CALLED  SAVE    (14,10)     Stores the contents of registers 14, 15, 0, and 1 in words
                              4, 5, 6, and 7 of the save area. These are conventional
                              linkage registers. Registers 2 through 10, which are
                              not actually used in linkage, are saved in subsequent
                              words of the save area. The implication in not saving
                              registers 11 and 12 is that they will not be used in
                              this program. Consequently there is no need to save
                              them since their contents will be unchanged when control
                              is returned. The expanded code of the SAVE macro
                              instruction uses register 13, which contains the address
                              of the save area, in effecting the storage of registers.

        LR      2,13        Loads the address of the save area into register 2, which
                              subsequently will be used to refer to the save area.

        LM      3,5,0(1)    Loads into registers 3, 4, and 5 the addresses of the
                              three arguments passed to the program. The address of
                              the argument list always is passed in register 1, which
                              here is used as the base register to get the addresses.
                              Subsequent references to the first argument will use
                              register 3 as the base register for that address.
                              References to the second and third arguments will use
                              registers 4 and 5. (If a variable length argument list
                              could be used in calling this program, each argument
                              would be tested for a one in the high order bit.)

        (Return Routine)

        LR      13,2        Loads into register 13 the address of the save area that
                              was passed to this program.

        RETURN  (14,10),T   This RETURN macro instruction restores the saved registers
                              (14, 15, and 0 through 10). The return address is
                              restored to register 14, and the expansion includes a
                              branch to that instruction. The T in the RETURN macro-
                              instruction causes the eight high-order bits of word 4
                              of the save area to be set to ones as an indication that
                              the return occurred.
```

Note: If the called program containing this coding also calls another program, it would contain the calling sequence, and it would establish linkage between the save area it had received and the save area it passes to the called program.

Figure 44. Sample Linkage Coding Used in a Called Assembler Language Program

```
//JOBX      JOB
//STEP1     EXEC     PGM=IEQCBL00,PARM=LOAD
            .
            .
            .
//SYSLIN    DD       DSNAME=&GOFILE,DISP=(MOD,PASS),UNIT=SYSSQ
//SYSIN     DD       *
            (Source module for MAIN, a calling program)
/*
//STEP2     EXEC     PGM=IEQCBL00,PARM=LOAD
            .
            .
            .
//SYSLIN    DD       DSNAME=*.STEP1.SYSLIN,DISP=(MOD,PASS)
//SYSIN     DD       *
            (Source module for ADD, a called program)
/*
//STEP3     EXEC     PGM=IEQCBL00,PARM=LOAD
            .
            .
            .
//SYSLIN    DD       DSNAME=*.STEP2.SYSLIN,DISP=(MOD,PASS)
//SYSIN     DD       *
            (Source module for SUBTRACT, a called program)
/*
//STEP4     EXEC     PGM=IEWL
            .
            .
            .
//SYSLIB    DD       DSNAME=SYS1.COBLIB,DISP=OLD
//SYSLMOD   DD       DSNAME=PROGLIB(CALC),DISP=OLD
//ADDLIB    DD       DSNAME=MYLIB,DISP=OLD
//SYSLIN    DD       DSNAME=&GOFILE,DISP=OLD
//          DD       *
            INCLUDE  ADDLIB(A)
            LIBRARY  ADDLIB (X,Y,Z)
/*
```

Figure 45. Specifying Primary and Additional Input to the Linkage Editor

SPECIFYING ADDITIONAL INPUT

Additional input data sets are specified by linkage editor control statements and a DD statement for each additional input data set.

The linkage editor control statements that specify additional input are INCLUDE and LIBRARY.[1] A primary input data set may consist entirely of such statements. INCLUDE and LIBRARY statements may be placed before, between, or after object modules or other control statements in either primary or additional input data sets. One method of using these statements is shown in Figure 45.

---

[1]The operation field in a linkage editor control statement must start after column 1. The operand field must be preceded by at least one blank.

Note: Additional input often contains members of libraries. See "Specifying Libraries As Additional Input" in the chapter "Libraries" for more information.

INCLUDE Statement

The INCLUDE statement is used to include an additional input data set that is either a member of a library or a sequential data set. Its format is:

Operation	Operand
INCLUDE	ddname[(member-name [,member-name]...)] [,ddname[(member-name [,member-name...])]]...

where ddname indicates the name of the DD statement that specifies the library or sequential data set, and member-name is the name of the library member that is to be included. Member-name is not used when the additional input data set is not a member of a partitioned data set.

Note: The linkage editor INCLUDE statement should not be confused with the COBOL language INCLUDE statement.

LIBRARY Statement

The LIBRARY statement is used to include additional input that may be required to resolve external references.

The format is:

Operation	Operand
LIBRARY	ddname(member-name [,member-name]...) [,ddname(member-name [,member-name...])]...

where ddname indicates the name of the DD statement that specifies the library, and member-name is the name of the member of the library.

The LIBRARY statement differs from the INCLUDE statement in that libraries specified in the LIBRARY statement are not searched for additional input until all other processing, except references reserved for the automatic library call, is completed by the linkage editor. Any additional module specified by an INCLUDE statement is included immediately, whenever the INCLUDE statement is encountered.

LINKAGE EDITOR PROCESSING

The linkage editor first processes the primary input and any additional input specified by INCLUDE statements. All external references that refer only to other modules in the primary and included input are resolved. If there are still unresolved references after this input is processed, the automatic call library, which includes libraries specified by the SYSLIB DD statement and by the LIBRARY statements, is searched to resolve the references. The automatic call library generally will contain the COBOL library subroutines. (External references to these subroutines are generated by the COBOL compiler when statements in the source module require certain functions to be performed, such as some data conversions.)

If the additional input contains external references and/or linkage editor control statements, the references are resolved in the same way. Data sets specified by the INCLUDE statement are included when the statement is encountered. Data sets specified by the LIBRARY statement are used only if there are unresolved references after all of the other processing is completed.

Example of Linkage Editor Processing

Figure 45 shows the control statements for a job that separately compiles three source modules (one is a calling program and two are called programs) and places them in one data set as primary input for the linkage editor. The linkage editor then links them together with additional input (called programs that are members of the specified library) to form one load module.

STEP1 compiles a source module called MAIN, STEP2 compiles a source module called ADD. STEP3 compiles a source module called SUBTRACT. The object module from each step is placed in the sequential data set called &GOFILE. (Since MOD and PASS are specified for &GOFILE in the SYSLIN DD statement in STEP1, the object modules ADD and SUBTRACT are placed in the data set behind the object module, MAIN.)

In STEP4 the linkage editor uses the &GOFILE data set as primary input, and the cataloged libraries MYLIB and SYS1.COBLIB as additional input. (The INCLUDE and LIBRARY statements become part of the primary input through the DD * statement following the SYSLIN DD statement.)

The object modules of the data set &GOFILE and the member A of MYLIB are processed first. If there are unresolved references after this input is processed, the linkage editor searches the automatic call library, which includes the COBOL subroutine library and members X, Y and Z of MYLIB, to resolve these references. MYLIB is specified in the ADDLIB DD statement.

After linkage editor processing is completed, the load module CALC is added as a member to the existing, cataloged library PROGLIB. CALC now contains MAIN, SUBTRACT, ADD, A, and, possibly, COBOL subroutines, and X, Y, and Z.

OVERLAY STRUCTURES

If the called programs needed to execute one COBOL source program do not all fit into main storage at the same time, it is still possible to use them with the overlay technique. Called programs that do not need to be in main storage at the same time can be given the same relative storage address and then loaded at different times during execution when they are needed. In this way, the same storage space can be used for more than one called program.

Considerations for Overlay

Assume a COBOL main program, called COBMAIN, exists that contains calls at one or more points in its logic to COBOL subprograms: CSUB1, CSUB2, CSUB3, CSUB4, and CSUB5. Also assume that the load module sizes for the main program and the subprograms are given as follows:

PROGRAM	MODULE SIZE (IN BYTES)
COBMAIN	20,000
CSUB1	4,000
CSUB2	5,000
CSUB3	6,000
CSUB4	3,000
CSUB5	4,000

Through the linkage mechanism, ENTER LINKAGE, CALL SUB1..., all subprograms plus COBMAIN must be linkage edited together to form one module 42,000 bytes in size. Therefore, COBMAIN would require 42,000 bytes of storage in order to be executed.

If the subprograms needed do not fit into main storage, the following two techniques of overlay are available to the COBOL programmer:

- Preplanned overlay using the linkage editor
- Dynamic overlay using macro instructions during execution

Note: The largest load module that can be processed by Fetch is 524,248 bytes. If a load module exceeds this limit, it should be divided.

Preplanned Linkage Editing with Overlay

The preplanned linkage editor facility permits the reuse of storage locations already occupied. By judiciously segmenting a program and using the preplanned linkage editor overlay facility, the programmer can accomplish the execution of a program too large to fit into storage at one time.

In using the preplanned overlay technique, the programmer specifies to the linkage editor which subprograms are to overlay each other. The subprograms specified are processed as part of the program by the linkage editor, so they can be automatically placed in main storage for execution when requested by the program. The resulting output of the linkage editor is called an overlay structure.

It is possible, at linkage edit time, to set up an overlay structure by using the COBOL source language statement ENTER LINKAGE and the linkage editor OVERLAY statement. These statements enable a user to call a subprogram that is not actually in storage. The details for setting up the linkage editor control statements for accomplishing this procedure can be found in the publication IBM System/360 Operating System: Linkage Editor, Form C28-6538.

In a linkage editor run, the programmer specifies the overlay points in a program by using OVERLAY statements. The linkage editor treats the entire input as one program, resolving all symbols and inserting tables into the program.

These tables are used by the control program to bring the overlay subprograms into storage automatically when called.

Figure 46 shows the deck arrangement for an overlay structure using preplanned linkage editor overlay. The OVERLAY statements specify to the linkage editor that the overlay structure to be established is one in which SUBPROGA, SUBPROGB, and SUBPROGC overlay each other when called during execution.

Dynamic Overlay Technique

In preparation for the dynamic overlay technique, each part of the program that is brought into storage independently should be processed separately by the linkage editor. (Hence, each part must be processed as a separate load module). To execute the entire program, the programmer must:

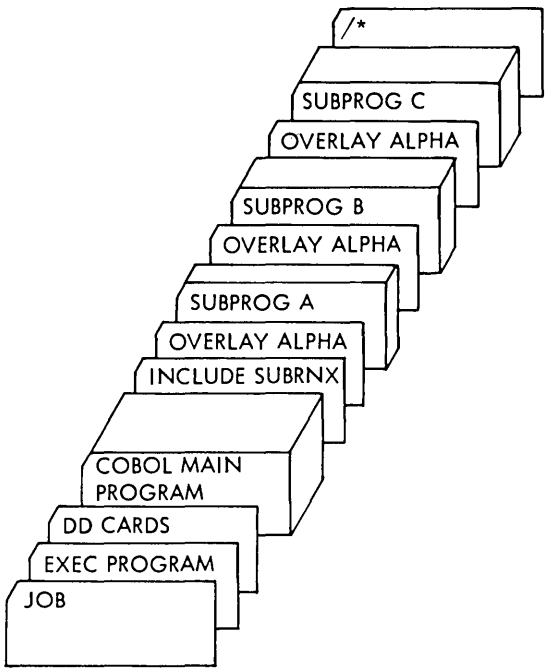


Figure 46. Sample Deck for Linkage Editor Overlay Structure

1. Specify the main program in the EXEC statement, and

2. Bring the separately processed load modules into storage when they are required, by using the appropriate supervisor linkage macro instructions. This is accomplished during execution.

The dynamic overlay technique can be used to overlay subprograms during execution. To accomplish dynamic overlay of subprograms, the programmer must write an assembler language subprogram that employs the LINK macro to call each COBOL subprogram. For a detailed description of the LINK macro instruction, refer to the publication IBM System/360 Operating System: Supervisor and Data Management Macro-Instructions, Form C28-6647.

In using the dynamic overlay technique, the main program communicates with the assembler language subprogram by using the COBOL language CALL statement. The COBOL CALL statement can be used to pass the name of the COBOL subprogram (to be linked) and the specified parameter list to the assembler language subprogram. This procedure is effected with each CALL used in the main program. Hence, each CALL results in linking with a subprogram through the assembler language subprogram.

When the COBOL subprogram is finished executing, it returns to the assembler language subprogram, which in turn returns to the main program. The process is repeated for each CALL to the assembler language subprogram.

This technique requires that a programmer have detailed knowledge of the linkage conventions, assembler language, and the LINK macro with its features and restrictions.

Beyond this, the programmer must ensure that the COBOL subprogram modules exist in a private library (PDS) and are defined by a //JOBLIB DD statement in the job control language for execution of the main program.

The following topics are discussed in this chapter: the data control block, error processing for COBOL files, and volume and data set labels.

More information about input/output processing is contained in the publication IBM System/360 Operating System: Supervisor and Data Management Services, Form C28-6646.

THE DATA CONTROL BLOCK

Each data set is described to the operating system by a data control block (DCB). A data control block consists of a group of contiguous fields that provide information about the data set to the system for scheduling and executing input/output operations. The fields describe the characteristics of the data set (e.g., data set organization) and its processing requirements (e.g., whether the data set is to be read or written). The COBOL compiler creates a skeleton DCB for each data set and inserts pertinent information specified in the Environment Division, FD entry, and input/output statements in the source program. The DCB for each file is part of the object module that is generated. Subsequently, other sources can be used to enter information into the data control block fields. The process of filling in the data control block is completed at execution time.

Additional information that completes the DCB at execution time may come from the DD statement for the data set and, in certain instances, from the data set label when the file is opened.

OVERRIDING DCB FIELDS

Once a field in the DCB is filled in by the COBOL compiler, it cannot be overridden by a DD statement or a data set label. For example, if the blocking factor for a data set is specified in the COBOL source program by the BLOCK CONTAINS clause, it cannot be overridden by a DD statement. In the same way, information from the DD statement cannot be overridden by information included in the data set label.

To use an option which is not supported by the COBOL (F) compiler, the bit for the option in the DCB may be set by calling an assembler language subroutine. For example, to use the TRACK OVERFLOW feature of the System/360 Operating System, the bit for the option must be set in the RECFM field of the DCB. However, since the compiler does not support this option and the RECFM field contains bits that are set by the compiler (including the RECORDING MODE bit), this field cannot be overridden by a DD statement. The TRACK OVERFLOW bit can be set by issuing a CALL instruction (CALL SETBIT USING file-name), where SETBIT is an assembler language subroutine which sets the bit at DCB+36. For information about the use of CALL see the chapter "Calling and Called Programs" in this publication.

IDENTIFYING DCB INFORMATION

The links between the DCB, DD statement, data set label, and input/output statements are the file-name, the external name in the ASSIGN TO clause of the SELECT statement, the ddname, and the dsname (see Figure 47).

1. The file-name specified in the SELECT statement and in the FD entry of the COBOL source program is the name associated with the DCB.

2. The external name specified in the ASSIGN TO clause of the source program is the ddname link to the DD statement. This name is placed in the DCB.

3. The dsname specified in the DD statement is the link to the physical data set.

The fields of the data control block are described in the tables in Appendix C. They identify those fields for which information must be supplied by the source program, by a DD statement, or by the data set label. For further information about the data control block, see the discussion of the DCB macro instruction for the appropriate file processing technique in the publication IBM System/360 Operating System: Supervisor and Data Management Services, Form C28-6646.

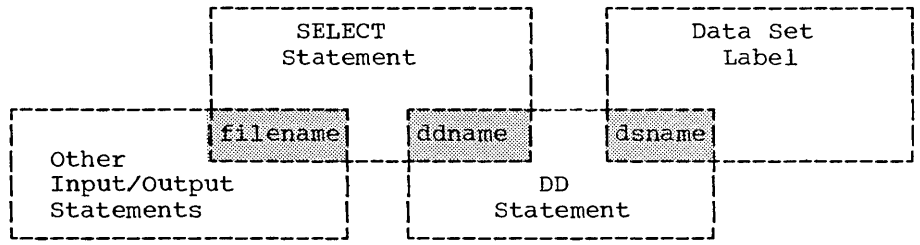


Figure 47. Links Between the Select Statement, the DD Statement, the Data Set Label, and the Input/Output Statements

ERROR PROCESSING FOR COBOL FILES

System Error Recovery

During the processing of a COBOL file, data transmission to or from an input/output device may not be successful the first time it is attempted. If it is not successful, standard error recovery routines, provided by the operating system, attempt to clear the failure and allow the program to continue uninterrupted.

If the error cannot be corrected by the system, an abnormal termination (ABEND) of the program may occur unless the programmer has specified some means of error analysis. Error processing routines initiated by the programmer are discussed in the following paragraphs.

DD Statement Option

For standard sequential files (QSAM), the programmer can specify a DD statement option which allows continued processing of the interrupted file. If the programmer has also written an error processing declarative section (USE AFTER STANDARD ERROR) in his program, the DD statement option is executed when a normal exit is taken from the declarative. For further information on this DD statement option, see the discussion of the EROPT subparameter of the DCB parameter in the chapter, "Execution Time Data Set Requirements."

INVALID KEY Option

Invalid key errors may occur for BISAM files, for QISAM files opened as output, and for BDAM files. These errors may be tested for by means of the INVALID KEY clause of the READ, REWRITE, or WRITE verb.

USE AFTER STANDARD ERROR Option

The programmer may specify the USE option in the declarative section of the Procedure Division to determine the particular cause of the input/output error. With the USE AFTER STANDARD ERROR option (Option 2 of the USE sentence), the programmer can pass control to an error processing routine to investigate the nature of the error. To help find the error condition, this error processing routine may include a call to an assembler language subroutine.

For BSAM or QSAM, using an assembler language subroutine, the programmer can investigate the contents of register 1 to determine the type of error and take appropriate action, as for example, returning an error code to the main program.

For the other file processing techniques, the assembler language subroutine can be used to distinguish between invalid key and non-invalid key errors. The filename is passed as an argument to the called assembler language subroutine. For files using the basic file processing techniques (BDAM and BISAM) the address of the data event control block (DECB) is passed. For files using the queued technique (QISAM), the address of the data control block (DCB) is passed. The DCB or DECB contains at least one byte of information describing the condition causing the error. With assembler language instructions, the program can interrogate the condition byte, distinguish between KEY and other errors, and take appropriate recovery action. Linkage to the assembler subroutines can be accomplished by using the following statements:

For BSAM or QSAM:

```
     USE AFTER ERROR ON FILE-A.

     ENTER LINKAGE.
     CALL 'ERRTNA'.
         (assembler language)
     ENTER COBOL.
```

In this example, ERRTNA is an assembler language program used to interrogate the contents of register 1.

For other processing techniques:

```
USE AFTER ERROR ON FILE-B.

ENTER LINKAGE.
CALL 'ERRTNB' USING FILE-B.
    (assembler language)
ENTER COBOL.
```

ERRTNB is the assembler language program used to investigate the error condition byte in the DCB or DECB.

Appendix G, "Input/Output Error Conditions," contains a summary of conditions causing errors for each processing technique.

Either the INVALID KEY clause or USE AFTER STANDARD ERROR declarative, but not both, may be specified for a file. If an error occurs and neither has been specified, the program may terminate abnormally or may continue executing with incorrect data. Table 16 lists the error processing facilities available for each access method. The following discussion summarizes the action taken by each facility for each method.

QSAM

Operating System: If the error is uncorrectable (read only), the program will ABEND in the absence of a DD statement option or USE AFTER STANDARD ERROR declarative. If both the DD statement option and USE section are specified, the control program will execute the USE declarative first and then the DD option.

DD Statement Option: The EROPT subparameter in the DCB parameter specifies one of three actions: accept the error block, skip the error block, or terminate the job.

USE AFTER STANDARD ERROR: The programmer may specify this option in order to display the cause of the error. Control goes to the declarative section. The programmer may specify a call to an assembler language subroutine in the section to examine the contents of register 1. The programmer can then display a message indicating the error and execute his DD statement option.

QISAM

A. WRITE (load mode):

Operating System: If the error is uncorrectable, the program will ABEND unless an error processing routine is specified.

INVALID KEY: If the error is due to an invalid key, recovery is possible. (The programmer may attempt to reconstruct the key and retry the operation, or may bypass the error record.) If the error is not due to an invalid key, the program terminates.

USE AFTER STANDARD ERROR: Control goes to the declarative section. The programmer may specify a call to an assembler language subroutine in the section to determine the nature of the

• Table 16. Input/Output Error Processing Facilities

Access Method	Error Processing Facility Available			
	Operating System	DD Statement Option	COBOL Clauses	
			INVALID KEY	USE AFTER STANDARD ERROR
QSAM	X	X	(1)	X
QISAM WRITE	X		X	X
READ	X		(1)	X
BDAM	X		X	X
BSAM	X		(1)	X
BISAM	(2)		X	X

(1) Error cannot be caused by an invalid key.
(2) The system standard error recovery facility (SYNAD) is not available. If errors occur, they are ignored and processing continues with unpredictable results, unless a programmer-specified error processing routine is included.

error by investigating the DCB. If the error is due to a key error, recovery is possible. If the error is not due to a key error, the error is uncorrectable. The program may continue executing, but processing of the file is limited to CLOSE. If the programmer closes the file, he may do so in either the declarative section or in the main body of his program.

B. READ, REWRITE (scan mode):
Operating System: Same as for WRITE.

INVALID KEY: The error cannot be caused by an invalid key. A source program coding error is implied and a compiler diagnostic message is generated. The program is not executed.

USE AFTER STANDARD ERROR: The programmer may specify this option in order to display the cause of the error. Control goes to the declarative section. The programmer may specify a call to an assembler language subroutine in the section to determine the nature of the error in the DCB. Since the error cannot be due to an invalid key, processing of the file is limited to CLOSE. If the programmer elects to close the file, he may do so in either the declarative section or in the main body of his program.

BDAM

Operating System: If the error is uncorrectable, the program will ABEND unless an error processing routine is specified.

INVALID KEY: If the error is due to an invalid key, recovery is possible. If the error is not due to an invalid key, the program terminates.

USE AFTER STANDARD ERROR: Control goes to the declarative section. The programmer may specify a call to an assembler language subroutine in the section to determine the nature of the error in the DECB. If the error is due to a key error or the "no space found" condition, recovery is possible. Any other error is uncorrectable. The program may continue executing, but processing of the file is limited to CLOSE. If the programmer closes the file, he may do so in either the declarative section or in the main body of his program.

BSAM

Operating System: If the error is uncorrectable, the program will ABEND unless an error processing routine is specified.

USE AFTER STANDARD ERROR: The programmer may specify this option in order to display the cause of the error. Control goes to the declarative section. The programmer may specify a call to an assembler language subroutine in the section to examine the contents of register 1. Since the error cannot be due to an invalid key, processing of the file is limited to CLOSE. If the programmer elects to close the file, he may do so in either the declarative section or in the main body of his program.

BISAM

INVALID KEY: If the error is due to an invalid key, recovery is possible. If the error is not due to an invalid key, the error is ignored and processing of the file continues with unpredictable results.

USE AFTER STANDARD ERROR: Control goes to the declarative section. The programmer may specify a call to an assembler language subroutine in the section to determine the nature of the error. If the error is due to a key error or the "no space found" condition, recovery is possible. Any other error is uncorrectable. In some cases, the programmer may wish to close the file or he may try to bypass the error (see Appendix G). If the programmer closes the file, he may do so in either the declarative section or in the main body of his program.

VOLUME LABELING

Various groups of labels may be used in secondary storage to identify magnetic tape and direct-access volumes, as well as the data sets they contain. The labels are used to locate the data sets and are identified and verified by label processing routines of the operating system.

There are two different kinds of volume labels, standard and non-standard. Magnetic tape volumes can have standard or nonstandard labels, or they can be unlabeled. The type(s) of label processing for tape volumes to be supported by an installation is selected during the system generation process. Direct-access volumes are supported with standard labels only.

Standard label support includes the following label groups:

A volume label group

A data set label group for each data set

Optional user label groups, i.e., user header and/or trailer labels for each data set, for physical sequential data set organizations.

Specific information about the contents of standard labels is contained in the publication IBM System/360 Operating System: Supervisor and Data Management Services, Form C28-6646.

STANDARD VOLUME LABELS

A volume label group is composed of an initial volume label and up to seven additional volume labels. The initial volume label is written by a utility program. (For a direct-access volume, this is a volume initialization program that is required to initialize the volume before it can be used for data storage.)

Additional volume labels are processed by installation-written routines.

The volume label contains a volume serial number that is used to identify the volume. The programmer can request a specific volume by specifying its serial number in the SER subparameter of the VOLUME parameter in the DD statement. The volume label is intended primarily to verify that the correct volume is mounted. It also can be used to prevent use of the volume by unauthorized programs.

DATA SET LABELS

Data set labels are written when the data set is created. The programmer specifies, in the LABEL parameter of the DD statement for the data set, the type of labels he wants for the data set. Standard data set labels contain installation data (e.g., creation date and expiration date), device-dependent information (e.g., tape density), and information about the characteristics of the data set (e.g., record format and block length). The DD statement for a data set and the program that creates the data set are sources of information for its label. When a data set is to be retrieved, its label type is identified in the LABEL parameter of the DD statement.

MAGNETIC TAPE VOLUME AND DATA SET LABELS

All standard tape labels (see Figure 48) are 80-character records. They are written in extended binary-coded-decimal interchange code (EBCDIC) for 9-track tape units and in binary-coded-decimal (BCD) for 7-track tape units. The tape label is recorded in the same density as the data on the tape.

A tape using standard tape labels is identified as such by the operating system when it reads the initial record and determines that it is an initial volume label by finding that the first four characters of the record are VOL1 (volume label 1).

Data Set Header Label Group: The data set header label group consists of two data set header labels: HDR1 and HDR2. HDR1 contains operating system data and device-dependent information; HDR2 contains data set characteristics. (Additional data set header labels are not supported.) These labels are created by the operating system in accordance with a fixed format when the data set is recorded on tape. They can then be used to locate the data set, to verify references to the data set, and to protect it from unauthorized use.

Nonstandard Labels

The number and contents of nonstandard labels can be specified by the user, except that the initial record cannot be a standard tape volume label; i.e., the first four characters of this record cannot be VOL1. When these first four characters are not VOL1, the operating system transfers control to the nonstandard label processing routines. These routines provide volume positioning that is compatible with the positioning techniques used by the system's standard label processing routines. The operating system assumes that a tape using nonstandard labels is properly positioned upon completion of a nonstandard label processing routine.

Information concerning use of unlabeled volumes, single data sets in multiple volumes, and multiple data sets in multiple volumes is contained in the publication IBM System/360 Operating System: Supervisor and Data Management Services, Form C28-6646.

All direct-access volume labels are written in extended binary-coded-decimal interchange code (EBCDIC).

The format of the direct-access volume label group is the same as the format of the tape volume label group, except for field five of the initial volume label.

The data set label of a data set in a direct-access volume consists of the data set control block (DSCB). The DSCB appears in the volume table of contents (VTOC) and contains the equivalent of the tape data set header and trailer information, in addition to space allocation and other control information.

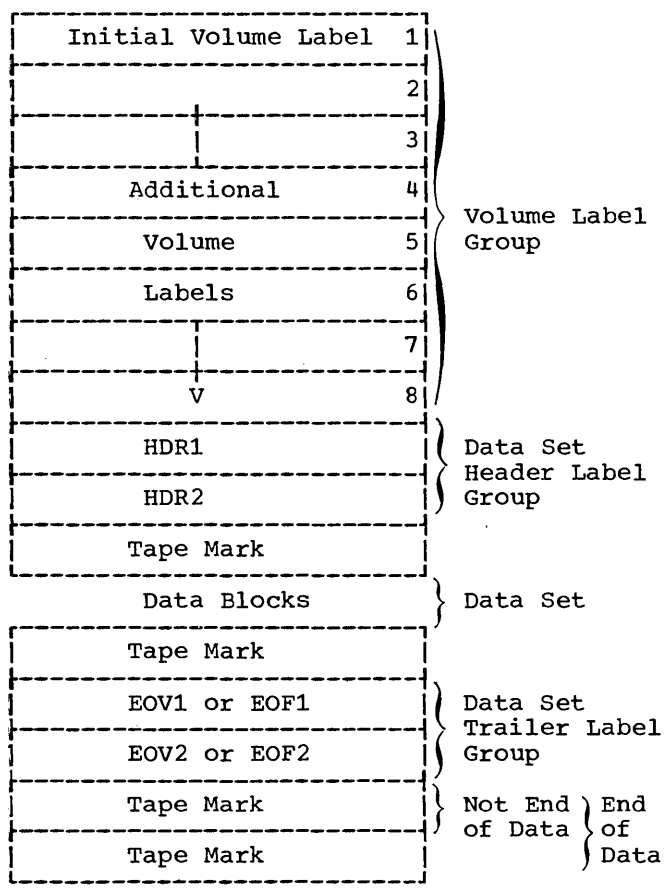


Figure 48. Standard Tape Labels

Volume Table of Contents

The volume table of contents (VTOC) contains data set control blocks that describe each data set in that volume. Each data set control block (DSCB) contains the dsname of the data set and information indicating its location in the volume and the amount of space allocated to it. The dsname of the data set should be unique in the volume. If the volume serial number is known, the data set can be located in that volume.

The VTOC also contains an indication of the amount of space still available in the volume. When space is to be allocated for a new data set (SPACE, SPLIT, etc.), the system checks to see if sufficient space is available. If it is, a new data set label is created for the data set, and the corresponding VTOC entries are adjusted to reflect the decrease in available space. If the RLSE subparameter of the SPACE parameter has been specified, all unused space assigned to the data set is made available to the volume when the data set is closed. If the disposition KEEP has been specified, the data set remains tabulated in the VTOC after the completion of the job. If the disposition catalog has been specified, the data set remains tablulated in the VTOC and, in addition, an entry is made in a system index (see "Cataloging Data Sets" in the chapter "Using the DD Statement").

When a data set is deleted (DISP=(,DELETE)) all references are deleted from the VTOC, and the space formerly occupied by it is made available for other data sets. If the data set has been accessed by using the system catalog, its name is also deleted from the catalog.

When a data set is uncataloged -- DISP=(,UNCATLG) -- however, only the entry from the system catalog is removed; the data set remains on the volume, and its identification remains in the VTOC.

This chapter contains information concerning system requirements for the COBOL (F) compiler, execution time, and the sort feature. Additional information for use in estimating the main and auxiliary storage requirements is contained in the publication IBM System/360 Operating System: Storage Estimates, Form C28-6581.

MINIMUM MACHINE REQUIREMENTS FOR THE COBOL (F) LEVEL COMPILER

The basic system requirements for use of the COBOL (F) compiler are:

- At least a System/360 Model 40, with a minimum of 80K (81,920) bytes of main storage available to the compiler, and the standard and decimal instruction sets. The floating-point instruction set is required if floating-point data items and fractional exponents are used in the program.

  At least 80K (81,920) bytes should be allocated in the SIZE option of the EXEC job control card that requests execution of the compiler. If less than this is specified, the system assumes the default value of 80K. If more storage is allocated, the compiler will run more efficiently.

- A device, such as the 1052 Printer-Keyboard, for direct operator communication.

- At least one direct-access device, such as an IBM 2311 Disk Storage Drive, for residence of the operating system and SYSUT1. Both may reside on the same volume.

- A device, such as a card reader or a tape unit, for the job input stream.

- A printer or tape unit for the system output file, unless some other device specified for direct operator communication is selected at system generation time.

- SYSUT1 on a direct-access device. SYSUT2, SYSUT3, and SYSUT4 can reside on tape or on a direct-access device. If they reside on a tape device, there must be a tape unit for each data set. If they reside on a direct-access device, there must be enough space in the device.

MULTIPROGRAMMING WITH A VARIABLE NUMBER OF TASKS (MVT)

The REGION Parameter

COMPILATION: If the compiler is being executed under the MVT control program (multiprogramming with a variable number of tasks) of the System/360 Operating System, the REGION parameter, specified as 86K bytes in the COBFC and COBFCLG cataloged procedures, becomes significant (see the section "Cataloged Procedures"). If the programmer wishes to override this value, he can specify a region size in either the JOB or the EXEC statements of the compiler. The size specified should not be less than the value of SIZE in the PARM field of the EXEC statement, rounded to the next highest 2K multiple, plus 6K.

The following examples illustrate both the default and the override cases:

1.
```
//JOB1   JOB   1234,J.SMITH
//STEP1  EXEC  COBFC
          .
          .
          .
```

In this example, the programmer accepts the default value of 86K specified in the COBFC cataloged procedure.

2.
```
//JOB2   JOB   1234,J.SMITH
//STEP1  EXEC  COBFCLG,REGION=134K,  X
//             PARM.COB='SIZE=130000'
          .
          .
          .
```

In this example, the default value is overridden. Rounding 130000 to the next highest 2K multiple, it becomes 131072, or 128K. Thus, the correct region size is 128K+6K=134K. (K=1024 bytes.)

EXECUTION: Priority schedulers require that the REGION parameter be specified for execution of object programs, unless the programmer is willing to accept default region size. The default value is 52K bytes, and is established in the input reader procedure. The region size needed for the execution of the object program is the sum of the following values:

1. The size of the object module after it has been linkage edited with all of the necessary object time subroutines.

2. The size of the input/output buffers being used, multiplied by the blocking factor (QSAM files are double buffered if no blocking factor is specified).

3. The size of the data management routines and control blocks that are used (see the publication IBM System/360 Operating System: Storage Estimates, Form C28-6551).

4. An additional 4K bytes.

Intermediate Data Sets Under MVT and MFT

SYSIN and SYSOUT data resides on intermediate, direct-access, execution-time data sets. These data sets are used by the system to temporarily hold all of the job's input and output data. The handling of SYSIN and SYSOUT data under MVT and MFT differs significantly from the handling under the primary control program (PCP). This is true because the object program must use intermediate data sets on direct-access devices instead of unit record devices.

SYSIN-SYSOUT CHARACTERISTICS: The input and output data set characteristics are determined by the system, but can be altered by the programmer if necessary. The procedure used to alter the default values depends on whether the data set is for input or output, as follows:

- For SYSIN data -- the programmer must request, at the time the job is submitted, that the operator use one of the several reader procedures available. Reader procedures are cataloged procedures that control the reader and vary according to the blocking factor specified.

- For SYSOUT data -- the programmer must use override statements as described in the section "Cataloged Procedures."

When a job is being run in an MVT or MFT environment, the SPACE parameter assumes added importance. Under the primary control program, output goes directly to the printer as it is produced; under MVT and MFT, output is placed on the SYSOUT intermediate data set as it is produced. Since nothing is written out until the completion of the job, the programmer must make sure that the SYSOUT data set is large enough to hold all of the possible output data of his program. The SPACE parameter of the DD statement is specified for SYSOUT with a default value of 50 tracks. If the programmer determines that his output will exceed the default value, he can do either or both of two things:

1. Specify blocking of his data set with the DCB parameter of an override DD statement

2. Override the compilation step of a compiled procedure by specifying the SPACE parameter. An example of a statement that can be used is:

```
//COB.SYSPRINT  DD  SPACE=(121,(500,50))
```

Note: If the TRK or CYL subparameters of the SPACE parameter are used, the programmer should be aware that requests will differ depending upon the direct-access device used (2301, 2303, 2311,...etc.). To avoid this consideration, the average-record-length subparameter can be used.

MULTIPLE OPEN AND CLOSE STATEMENTS: Under the MVT and MFT control programs, input data following the DD* or DD DATA card becomes a single data set. Once a CLOSE statement is encountered, repositioning takes place to the beginning of the data set. To avoid errors, the programmer should keep this in mind when using more than one OPEN and CLOSE statement for a data set assigned to SYSIN.

Note: Under MVT, a file must be closed before the STOP RUN statement is executed. Failure to do this will result in an abnormal termination.

EXECUTION TIME CONSIDERATIONS

The amount of main storage must be sufficient to accommodate at least:

- The selected control program

- Support for the file processing technique used

- Load module to be executed

The number of input/output devices required for the source program is determined by whatever is specified in the Environment Division.

SORT FEATURE CONSIDERATIONS

The basic requirements for use of the Sort Feature are:

- A System/360 model with sufficient main storage to accommodate the load module to be executed plus a minimum of 15,360 bytes for the sort program execution.
- At least one direct-access device (which may be the system residence device) for residence of SYS1.SORTLIB.
- At least three tape units or one direct-access device for intermediate storage.

The following is a sample COBOL program and the output listing resulting from its compilation, linkage editing, and execution. The program creates a blocked, unlabeled, standard sequential file, writes it out on tape, and then reads it back in. It also does a check on the field called NO-OF-DEPENDENTS. All data records in the file are displayed. Those with a zero in the NO-OF-DEPENDENTS field are displayed with the special character Z. The records of the file are not altered from the time of creation, despite the fact that the NO-OF-DEPENDENTS field is changed for display purposes. The individual records of the file are created using the subscripting technique. TRACE is used as a debugging aid during program execution.

The chapter "Output" describes the output formats illustrated in the listing. The parts of the listing are numbered in accordance with the numbers used in the chapter "Output."

```
    //JOB3    JOB   PROGRAMMERNAME,MSGLEVEL=1
    //STEP3   EXEC  PGM=IEQCBL00,PARM='BUF=13000,SIZE=160000,MAP'
    //SYSUT1 DD     DSNAME=&UT1,UNIT=SYSDA,SPACE=(TRK,(100,10))
    //SYSUT2 DD     DSNAME=&UT2,UNIT=SYSDA,SPACE=(TRK,(100,10))
(1) //SYSUT3 DD     DSNAME=&UT3,UNIT=SYSDA,SPACE=(TRK,(100,10))
    //SYSUT4 DD     DSNAME=&UT4,UNIT=SYSDA,SPACE=(TRK,(100,10))
    //SYSLIN DD     DSNAME=PUNCH,UNIT=SYSDA,SPACE=(TRK,(100,10)),          X
    //              DISP=(NEW,PASS)
    //SYSPRINT DD   SYSOUT=A
    //SYSIN  DD     *
    IEF236I ALLOC. FOR JOB3     STEP3
    IEF237I SYSUT1   ON 190
    IEF237I SYSUT2   ON 190
(2) IEF237I SYSUT3   ON 190
    IEF237I SYSUT4   ON 190
    IEF237I SYSLIN   ON 190
    IEF237I SYSIN    ON 00C
```

```
00001   100010 IDENTIFICATION DIVISION.
00002   100020 PROGRAM-ID. 'TESTRUN'.
00003   100030   AUTHOR. PROGRAMMER NAME.
00004   100040   INSTALLATION. NEW YORK PROGRAMMING CENTER.
00005   100050   DATE-WRITTEN. MAY 31 1966.
00006   100060 DATE-COMPILED. NOV  2,1966
00007   100070   REMARKS. THIS PROGRAM HAS BEEN WRITTEN AS A SAMPLE PROGRAM FOR
00008   100080     COBOL USERS. IT CREATES AN OUTPUT FILE AND READS IT BACK AS
00009   100090     INPUT.
00010   100100 ENVIRONMENT DIVISION.
00011   100110 CONFIGURATION SECTION.
00012   100120   SOURCE-COMPUTER. IBM-360 E50.
00013   100130   OBJECT-COMPUTER. IBM-360 E50.
00014   100140 INPUT-OUTPUT SECTION.
00015   100150 FILE-CONTROL.
00016   100160     SELECT FILE-1 ASSIGN TO 'SAMPLE' UTILITY.
00017   100170     SELECT FILE-2 ASSIGN TO 'SAMPLE' UTILITY.
00018   100180 DATA DIVISION.
00019   100190 FILE SECTION.
00020   100200 FD  FILE-1
00021   100210     LABEL RECORDS ARE OMITTED
00022   100220     BLOCK CONTAINS 100 CHARACTERS
00023   100230     RECORDING MODE IS F
00024   100240     DATA RECORD IS RECORD-1.
00025   100250   01 RECORD-1.
00026   100260     02 FIELD-A  PICTURE IS X(20).
00027   100270 FD  FILE-2
00028   100280     LABEL RECORDS ARE OMITTED
00029   100290     BLOCK CONTAINS 5 RECORDS
00030   100300     RECORD CONTAINS 20 CHARACTERS
00031   100310     RECORDING MODE IS F
00032   100320     DATA RECORD IS RECORD-2.
00033   100330   01 RECORD-2.
00034   100340     02 FIELD-A PICTURE IS X(20).
00035   100350 WORKING-STORAGE SECTION.
00036   100360     77 COUNT  PICTURE S99  COMPUTATIONAL.
00037   100370     77 ALPHABET  PICTURE X(26) VALUE IS 'ABCDEFGHIJKLMNOPQRSTUVWX
00038   100380-    'YZ'.
00039   100390     77 ALPHA REDEFINES ALPHABET PICTURE X OCCURS 26 TIMES.
00040   100400     77 NUMBER  PICTURE S99 COMPUTATIONAL.
00041   100410     77 DEPENDENTS  PICTURE X(26)  VALUE IS '01234012340123401234O'
00042   100420-    '12340'.
00043   100430     77 DEPEND REDEFINES DEPENDENTS PICTURE X OCCURS 26 TIMES.
00044   100440   01 WORK-RECORD.
00045   100450     02 NAME-FIELD  PICTURE X.
00046   100460     02 FILLER  PICTURE X VALUE SPACE.
00047   100470     02 RECORD-NO  PICTURE 9999.
00048   100480     02 FILLER  PICTURE X  VALUE SPACE.
00049   100490     02 LOCATION  PICTURE AAA  VALUE 'NYC'.
00050   100500     02 FILLER  PICTURE X  VALUE SPACE.
00051   100510     02 NO-OF-DEPENDENTS  PICTURE XX.
00052   100520     02 FILLER  PICTURE X(7)  VALUE SPACES.
00053   100530 PROCEDURE DIVISION.
00054   100540 BEGIN. READY TRACE.
00055   100550     NOTE THAT THE FOLLOWING OPENS THE OUTPUT FILE TO BE CREATED
00056   100560     AND INITIALIZES COUNTERS.
00057   100570 STEP-1. OPEN OUTPUT FILE-1. MOVE ZERO TO COUNT, NUMBER.
00058   100580     NOTE THAT THE FOLLOWING CREATES INTERNALLY THE RECORDS TO BE
00059   100590     CONTAINED IN THE FILE, WRITES THEM ON TAPE, AND DISPLAYS
00060   100600     THEM ON THE CONSOLE.
00061   100610 STEP-2. ADD 1 TO COUNT, ADD 1 TO NUMBER, MOVE ALPHA (COUNT) TO
00062   100620         NAME-FIELD.
00063   100630         MOVE DEPEND (COUNT) TO NO-OF-DEPENDENTS.
00064   100640         MOVE NUMBER TO RECORD-NO.
00065   100650 STEP-3. DISPLAY WORK-RECORD UPON CONSOLE. WRITE RECORD-1 FROM
00066   100660         WORK-RECORD.
00067   100670 STEP-4. PERFORM STEP-2 THRU STEP-3 UNTIL COUNT IS EQUAL TO 26.
00068   100680     NOTE THAT THE FOLLOWING CLOSES OUTPUT AND REOPENS IT AS
00069   100690     INPUT.
00070   100700 STEP-5. CLOSE FILE-1. OPEN INPUT FILE-2.
00071   100710     NOTE THAT THE FOLLOWING READS BACK THE FILE AND SINGLES OUT
00072   100720     EMPLOYEES WITH NO DEPENDENTS.
00073   100730 STEP-6. READ FILE-2 RECORD INTO WORK-RECORD AT END GO TO STEP-8.
00074   100740 STEP-7. IF NO-OF-DEPENDENTS IS EQUAL TO '0' MOVE 'Z' TO
00075   100750          NO-OF-DEPENDENTS. EXHIBIT NAMED WORK-RECORD. GO TO
00076   100760          STEP-6.
00077   100770 STEP-8. CLOSE FILE-2 .
00078   100780          STOP RUN.
```

(4)

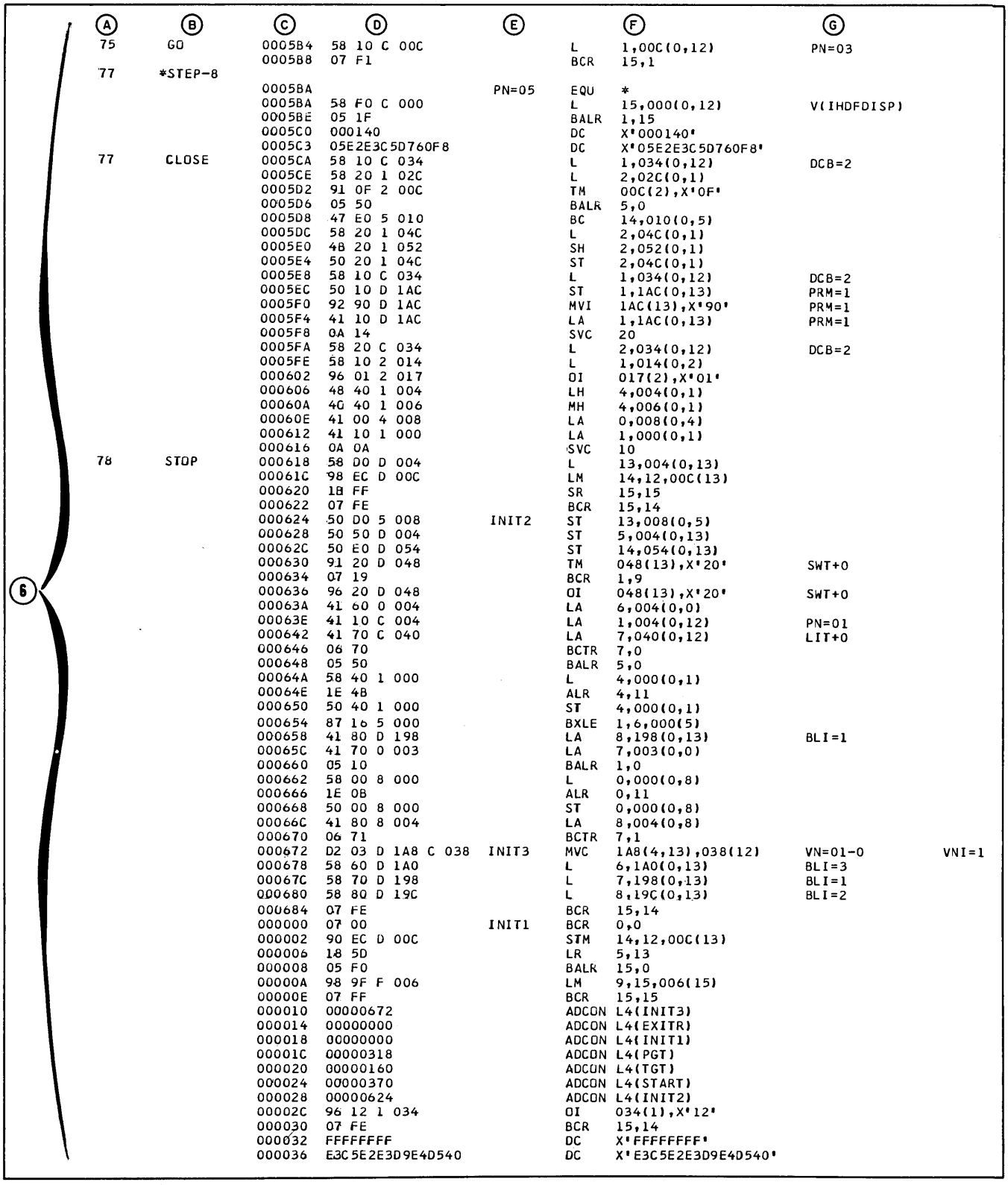
(A) INTRNL NAME	(B) LVL	(C) SOURCE NAME	(D) BASE	(E) DISPL	(F) INTRNL NAME	(G) DEFINITION	(H) USAGE	(I) R	O	Q
DNM=1-146	FD	FILE-1	DCB=01		DNM=1-146		QSAM			
DNM=1-164	01	RECORD-1	BLI=1	000	DNM=1-164	DS OCL20	GROUP			
DNM=1-184	02	FIELD-A	BLI=1	000	DNM=1-184	DS 20C	DISP			
DNM=1-200	FD	FILE-2	DCB=02		DNM=1-200		QSAM			
DNM=1-218	01	RECORD-2	BLI=2	000	DNM=1-218	DS OCL20	GROUP			
DNM=1-238	02	FIELD-A	BLI=2	000	DNM=1-238	DS 20C	DISP			
DNM=1-257	77	COUNT	BLI=3	000	DNM=1-257	DS 1H	COMP			
DNM=1-271	77	ALPHABET	BLI=3	002	DNM=1-271	DS 26C	DISP			
DNM=1-288	77	ALPHA	BLI=3	002	DNM=1-288	DS 1C	DISP	*	*	
DNM=1-305	77	NUMBER	BLI=3	01C	DNM=1-305	DS 1H	COMP			
DNM=1-320	77	DEPENDENTS	BLI=3	01E	DNM=1-320	DS 26C	DISP			
DNM=1-339	77	DEPEND	BLI=3	01E	DNM=1-339	DS 1C	DISP	*	*	
DNM=1-354	01	WORK-RECORD	BLI=3	038	DNM=1-354	DS OCL20	GROUP			
DNM=1-377	02	NAME-FIELD	BLI=3	038	DNM=1-377	DS 1C	DISP			
DNM=1-396	02	FILLER	BLI=3	039	DNM=1-396	DS 1C	DISP			
DNM=1-411	02	RECORD-NO	BLI=3	03A	DNM=1-411	DS 4C	DISP-NM			
DNM=1-429	02	FILLER	BLI=3	03E	DNM=1-429	DS 1C	DISP			
DNM=1-447	02	LOCATION	BLI=3	03F	DNM=1-447	DS 3C	DISP			
DNM=1-464	02	FILLER	BLI=3	042	DNM=1-464	DS 1C	DISP			
DNM=1-482	02	NO-OF-DEPENDENTS	BLI=3	043	DNM=1-482	DS 2C	DISP			
DNM=2-000	02	FILLER	BLI=3	045	DNM=2-000	DS 7C	DISP			

```
              MEMORY MAP
        (A)
        TGT                         00160

        SAVE AREA                   00160
        SWITCH CELL                 001A8
        TALLY CELL                  001AC
        UNUSED                      001B0
        ENTRY-SAVE                  001B8
        UNUSED                      001BC
        WORKING CELLS               001C0
        OVERFLOW CELLS              002F0
  (5)   TEMPORARY STORAGE CELLS     002F0
        TEMPORARY STORAGE-2 CELLS   002F8
        BLL CELLS                   002F8
        SBL-I CELLS                 002F8
        BL-I CELLS                  002F8
        SUBADR-I CELLS              00304
        ONCTL-I CELLS               00304
        PFMCTL-I CELLS              00304
        PFMSAV-I CELLS              00304
        VN-I CELLS                  00308
        DECBADR-I CELLS             0030C
        SAVE AREA =2                0030C
        SAVE AREA =3                0030C
        XSASW-I CELLS               0030C
        XSA-I CELLS                 0030C
        PARAM CELLS                 0030C
  (B)   RPTSAV AREA                 00314

LITERAL POOL (HEX)

00358 (LIT+0)      00000001  001AF0E9  C0000000

     DISPLAY LITERALS (BCD)

00364 (LTL+12)    'WORK-RECORD'
                    (C)
                    PGT                     00318

        OVERFLOW CELLS              00318
        VIRTUAL CELLS               00318
  (5)   PROCEDURE NAME CELLS        0031C
        GENERATED NAME CELLS        00330
        DCB ADDRESS CELLS           00348
        VNI CELLS                   00350
        LITERALS                    00358
        DISPLAY LITERALS            00364

 REGISTER ASSIGNMENT

  REG 6   BLI=3
  REG 7   BLI=1
```

```
    REG 8   BLI=2
   (A)      (B)         (C)         (D)                  (E)      (F)                              (G)
   54      *BEGIN
                        000370                           START    EQU    *
                        000370   58 F0 C 000                      L      15,000(0,12)              V(IHDFDISP)
                        000374   05 1F                            BALR   1,15
                        000376   000140                           DC     X'000140'
                        000379   04C2C5C7C9D5                     DC     X'04C2C5C7C9D5'
   54       READY       000380   96 40 D 048                      OI     048(13),X'40'             SWT+0
   57      *STEP-1
                        000384   58 F0 C 000                      L      15,000(0,12)              V(IHDFDISP)
                        000388   05 1F                            BALR   1,15
                        00038A   000140                           DC     X'000140'
                        00038D   05E2E3C5D760F1                   DC     X'05E2E3C5D760F1'
   57       OPEN        000394   58 10 C 030                      L      1,030(0,12)               DCB=1
                        000398   50 10 D 1AC                      ST     1,1AC(0,13)               PRM=1
                        00039C   92 8F D 1AC                      MVI    1AC(13),X'8F'             PRM=1
                        0003A0   41 10 D 1AC                      LA     1,1AC(0,13)               PRM=1
                        0003A4   0A 13                            SVC    19
                        0003A6   58 10 C 030                      L      1,030(0,12)               DCB=1
                        0003AA   58 F0 1 030                      L      15,030(0,1)
                        0003AE   05 EF                            BALR   14,15
                        0003B0   50 10 D 198                      ST     1,198(0,13)               BLI=1
                        0003B4   58 70 D 198                      L      7,198(0,13)               BLI=1
   57       MOVE        0003B8   D2 01 6 000 C 040                MVC    000(2,6),040(12)          DNM=1-257      LIT+0
                        0003BE   D2 01 6 01C C 040                MVC    01C(2,6),040(12)          DNM=1-305      LIT+0
   61      *STEP-2
                        0003C4                           PN=01    EQU    *
                        0003C4   58 F0 C 000                      L      15,000(0,12)              V(IHDFDISP)
(6)                     0003C8   05 1F                            BALR   1,15
                        0003CA   000140                           DC     X'000140'
                        0003CD   05E2E3C5D760F2                   DC     X'05E2E3C5D760F2'
   61       ADD         0003D4   48 30 C 042                      LH     3,042(0,12)               LIT+2
                        0003D8   4A 30 6 000                      AH     3,000(0,6)                DNM=1-257
                        0003DC   40 30 6 000                      STH    3,000(0,6)                DNM=1-257
   61       ADD         0003E0   48 30 C 042                      LH     3,042(0,12)               LIT+2
                        0003E4   4A 30 6 01C                      AH     3,01C(0,6)                DNM=1-305
                        0003E8   40 30 6 01C                      STH    3,01C(0,6)                DNM=1-305
   61       MOVE        0003EC   41 40 6 002                      LA     4,002(0,6)                DNM=1-288
                        0003F0   48 20 6 000                      LH     2,000(0,6)                DNM=1-257
                        0003F4   4C 20 C 042                      MH     2,042(0,12)               LIT+2
                        0003F8   1A 42                            AR     4,2
                        0003FA   5B 40 C 040                      S      4,040(0,12)               LIT+0
                        0003FE   D2 00 6 038 4 000                MVC    038(1,6),000(4)           DNM=1-377
   63       MOVE        000404   41 20 6 01E                      LA     2,01E(0,6)                DNM=1-339
                        000408   48 10 6 000                      LH     1,000(0,6)                DNM=1-257
                        00040C   4C 10 C 042                      MH     1,042(0,12)               LIT+2
                        000410   1A 21                            AR     2,1
                        000412   5B 20 C 040                      S      2,040(0,12)               LIT+0
                        000416   D2 00 6 043 2 000                MVC    043(1,6),000(2)           DNM=1-482
                        00041C   92 40 6 044                      MVI    044(6),X'40'              DNM=1-482+1
   64       MOVE        000420   48 30 6 01C                      LH     3,01C(0,6)                DNM=1-305
                        000424   4E 30 D 190                      CVD    3,190(0,13)               TS=01
                        000428   F3 31 6 03A D 196                UNPK   03A(4,6),196(2,13)        DNM=1-411      TS=07
                        00042E   96 F0 6 03D                      OI     03D(6),X'F0'              DNM=1-411+3
   65      *STEP-3
                        000432   58 F0 C 000                      L      15,000(0,12)              V(IHDFDISP)
                        000436   05 1F                            BALR   1,15
                        000438   000140                           DC     X'000140'
                        00043B   05E2E3C5D760F3                   DC     X'05E2E3C5D760F3'
```

```
 (A)       (B)        (C)       (D)                  (E)        (F)                           (G)
 65     DISPLAY    000442   58 F0 C 000                      L      15,000(0,12)          V(IHDFDISP)
                   000446   05 1F                            BALR   1,15
                   000448   0002                             DC     X'0002'
                   00044A   00                               DC     X'00'
                   00044B   000014                           DC     X'000014'
                   00044E   0D0001A0                         DC     X'0D0001A0'           BLI=3
                   000452   0038                             DC     X'0038'
                   000454   FFFF                             DC     X'FFFF'
 65     WRITE      000456   D2 13 7 000 6 038                MVC    000(20,7),038(6)      DNM=1-164       DNM=1-354
                   00045C   58 10 C 030                      L      1,030(0,12)           DCB=1
                   000460   58 F0 1 030                      L      15,030(0,1)
                   000464   05 EF                            BALR   14,15
                   000466   50 10 D 198                      ST     1,198(0,13)           BLI=1
                   00046A   58 70 D 198                      L      7,198(0,13)           BLI=1
                   00046E                         GN=01      EQU    *
                   00046E   58 10 D 1A8                      L      1,1A8(0,13)           VN=01
                   000472   07 F1                            BCR    15,1
 67    *STEP-4
                   000474                         PN=02      EQU    *
                   000474   58 F0 C 000                      L      15,000(0,12)          V(IHDFDISP)
                   000478   05 1F                            BALR   1,15
                   00047A   000140                           DC     X'000140'
                   00047D   05E2E3C5D760F4                   DC     X'05E2E3C5D760F4'
 67     PERFORM    000484   58 00 D 1A8                      L      0,1A8(0,13)           VN=01
                   000488   50 00 D 1A4                      ST     0,1A4(0,13)           PSV=1
                   00048C   58 00 C 01C                      L      0,01C(0,12)           GN=02
                   000490   50 00 D 1A8                      ST     0,1A8(0,13)           VN=01
                   000494                         GN=02      EQU    *
                   000494   48 30 6 000                      LH     3,000(0,6)            DNM=1-257
                   000498   49 30 C 044                      CH     3,044(0,12)           LIT+4
                   00049C   58 F0 C 020                      L      15,020(0,12)          GN=03
                   0004A0   07 8F                            BCR    8,15
                   0004A2   58 10 C 004                      L      1,004(0,12)           PN=01
                   0004A6   07 F1                            BCR    15,1
                   0004A8                         GN=03      EQU    *
                   0004A8   58 00 D 1A4                      L      0,1A4(0,13)           PSV=1
                   0004AC   50 00 D 1A8                      ST     0,1A8(0,13)           VN=01
 70    *STEP-5
                   0004B0   58 F0 C 000                      L      15,000(0,12)          V(IHDFDISP)
                   0004B4   05 1F                            BALR   1,15
                   0004B6   000140                           DC     X'000140'
                   0004B9   05E2E3C5D760F5                   DC     X'05E2E3C5D760F5'
 70     CLOSE      0004C0   58 10 C 030                      L      1,030(0,12)           DCB=1
                   0004C4   58 20 1 02C                      L      2,02C(0,1)
                   0004C8   91 0F 2 00C                      TM     00C(2),X'0F'
                   0004CC   05 50                            BALR   5,0
                   0004CE   47 E0 5 010                      BC     14,010(0,5)
                   0004D2   58 20 1 04C                      L      2,04C(0,1)
                   0004D6   4B 20 1 052                      SH     2,052(0,1)
                   0004DA   50 20 1 04C                      ST     2,04C(0,1)
                   0004DE   58 10 C 030                      L      1,030(0,12)           DCB=1
                   0004E2   50 10 D 1AC                      ST     1,1AC(0,13)           PRM=1
                   0004E6   92 90 D 1AC                      MVI    1AC(13),X'90'         PRM=1
                   0004EA   41 10 D 1AC                      LA     1,1AC(0,13)           PRM=1
                   0004EE   0A 14                            SVC    20
                   0004F0   58 20 C 030                      L      2,030(0,12)           DCB=1
                   0004F4   58 10 2 014                      L      1,014(0,2)
                   0004F8   96 01 2 017                      OI     017(2),X'01'
                   0004FC   48 40 1 004                      LH     4,004(0,1)
                   000500   4C 40 1 006                      MH     4,006(0,1)
```

```
   (A)     (B)        (C)      (D)                       (E)       (F)                              (G)
                      000504   41 00 4 008                         LA    0,008(0,4)
                      000508   41 10 1 000                         LA    1,000(0,1)
                      00050C   0A 0A                               SVC   10
   70      OPEN       00050E   58 10 C 034                         L     1,034(0,12)          DCB=2
                      000512   50 10 D 1AC                         ST    1,1AC(0,13)          PRM=1
                      000516   92 80 D 1AC                         MVI   1AC(13),X'80'        PRM=1
                      00051A   41 10 D 1AC                         LA    1,1AC(0,13)          PRM=1
                      00051E   0A 13                               SVC   19
   73      *STEP-6
                      000520                             PN=03     EQU   *
                      000520   58 F0 C 000                         L     15,000(0,12)         V(IHDFDISP)
                      000524   05 1F                               BALR  1,15
                      000526   000140                              DC    X'000140'
                      000529   05E2E3C5D760F6                      DC    X'05E2E3C5D760F6'
   73      READ       000530   58 10 C 034                         L     1,034(0,12)          DCB=2
                      000534   D2 02 1 021 C 025                   MVC   021(3,1),025(12)                     GN=04+1
                      00053A   58 F0 1 030                         L     15,030(0,1)
                      00053E   05 EF                               BALR  14,15
                      000540   50 10 D 19C                         ST    1,19C(0,13)          BLI=2
                      000544   58 80 D 19C                         L     8,19C(0,13)          BLI=2
                      000548   D2 13 6 038 8 000                   MVC   038(20,6),000(8)     DNM=1-354       DNM=1-218
                      00054E   58 10 C 010                         L     1,010(0,12)          PN=04
                      000552   07 F1                               BCR   15,1
   73      GO         000554                             GN=04     EQU   *
                      000554   58 10 C 014                         L     1,014(0,12)          PN=05
                      000558   07 F1                               BCR   15,1
   74      *STEP-7
                      00055A                             PN=04     EQU   *
(6)                   00055A   58 F0 C 000                         L     15,000(0,12)         V(IHDFDISP)
                      00055E   05 1F                               BALR  1,15
                      000560   000140                              DC    X'000140'
                      000563   05E2E3C5D760F7                      DC    X'05E2E3C5D760F7'
   74      IF         00056A   58 10 C 02C                         L     1,02C(0,12)          GN=06
                      00056E   58 20 C 028                         L     2,028(0,12)          GN=05
                      000572   D5 00 C 046 6 043                   CLC   046(1,12),043(6)     LIT+6           DNM=1-482
                      000578   07 72                               BCR   7,2
                      00057A   95 40 6 044                         CLI   044(6),X'40'         DNM=1-482+1
                      00057E   07 72                               BCR   7,2
   74      MOVE       000580                             GN=06     EQU   *
                      000580   D2 00 6 043 C 047                   MVC   043(1,6),047(12)     DNM=1-482       LIT+7
                      000586   92 40 6 044                         MVI   044(6),X'40'         DNM=1-482+1
   75      EXHIBIT    00058A                             GN=05     EQU   *
                      00058A   58 10 C 048                         L     1,048(0,12)          LIT+8
                      00058E   50 10 D 1AC                         ST    1,1AC(0,13)          PRM=1
                      000592   41 20 D 1AC                         LA    2,1AC(0,13)          PRM=1
                      000596   58 F0 C 000                         L     15,000(0,12)         V(IHDFDISP)
                      00059A   05 1F                               BALR  1,15
                      00059C   8001                                DC    X'8001'
                      00059E   10                                  DC    X'10'
                      00059F   00000B                              DC    X'00000B'
                      0005A2   0C00004C                            DC    X'0C00004C'          LIT+12
                      0005A6   0000                                DC    X'0000'
                      0005A8   00                                  DC    X'00'
                      0005A9   000014                              DC    X'000014'
                      0005AC   0D0001A0                            DC    X'0D0001A0'          BLI=3
                      0005B0   0038                                DC    X'0038'
                      0005B2   FFFF                                DC    X'FFFF'
```


A
B
C
D
E
F
G
6
75 GO 0005B4 58 10 C 00C L 1,00C(0,12) PN=03
0005B8 07 F1 BCR 15,1
77 *STEP-8
0005BA PN=05 EQU *
0005BA 58 F0 C 000 L 15,000(0,12) V(IHDFDISP)
0005BE 05 1F BALR 1,15
0005C0 000140 DC X'000140'
0005C3 05E2E3C5D760F8 DC X'05E2E3C5D760F8'
77 CLOSE 0005CA 58 10 C 034 L 1,034(0,12) DCB=2
0005CE 58 20 1 02C L 2,02C(0,1)
0005D2 91 0F 2 00C TM 00C(2),X'0F'
0005D6 05 50 BALR 5,0
0005D8 47 E0 5 010 BC 14,010(0,5)
0005DC 58 20 1 04C L 2,04C(0,1)
0005E0 4B 20 1 052 SH 2,052(0,1)
0005E4 50 20 1 04C ST 2,04C(0,1)
0005E8 58 10 C 034 L 1,034(0,12) DCB=2
0005EC 50 10 D 1AC ST 1,1AC(0,13) PRM=1
0005F0 92 90 D 1AC MVI 1AC(13),X'90' PRM=1
0005F4 41 10 D 1AC LA 1,1AC(0,13) PRM=1
0005F8 0A 14 SVC 20
0005FA 58 20 C 034 L 2,034(0,12) DCB=2
0005FE 58 10 2 014 L 1,014(0,2)
000602 96 01 2 017 OI 017(2),X'01'
000606 48 40 1 004 LH 4,004(0,1)
00060A 4C 40 1 006 MH 4,006(0,1)
00060E 41 00 4 008 LA 0,008(0,4)
000612 41 10 1 000 LA 1,000(0,1)
000616 0A 0A SVC 10
78 STOP 000618 58 D0 D 004 L 13,004(0,13)
00061C 98 EC D 00C LM 14,12,00C(13)
000620 18 FF SR 15,15
000622 07 FE BCR 15,14
000624 50 D0 5 008 INIT2 ST 13,008(0,5)
000628 50 50 D 004 ST 5,004(0,13)
00062C 50 E0 D 054 ST 14,054(0,13)
000630 91 20 D 048 TM 048(13),X'20' SWT+0
000634 07 19 BCR 1,9
000636 96 20 D 048 OI 048(13),X'20' SWT+0
00063A 41 60 0 004 LA 6,004(0,0)
00063E 41 10 C 004 LA 1,004(0,12) PN=01
000642 41 70 C 040 LA 7,040(0,12) LIT+0
000646 06 70 BCTR 7,0
000648 05 50 BALR 5,0
00064A 58 40 1 000 L 4,000(0,1)
00064E 1E 4B ALR 4,11
000650 50 40 1 000 ST 4,000(0,1)
000654 87 16 5 000 BXLE 1,6,000(5)
000658 41 80 D 198 LA 8,198(0,13) BLI=1
00065C 41 70 0 003 LA 7,003(0,0)
000660 05 10 BALR 1,0
000662 58 00 8 000 L 0,000(0,8)
000666 1E 0B ALR 0,11
000668 50 00 8 000 ST 0,000(0,8)
00066C 41 80 8 004 LA 8,004(0,8)
000670 06 71 BCTR 7,1
000672 D2 03 D 1A8 C 038 INIT3 MVC 1A8(4,13),038(12) VN=01-0 VNI=1
000678 58 60 D 1A0 L 6,1A0(0,13) BLI=3
00067C 58 70 D 198 L 7,198(0,13) BLI=1
000680 58 80 D 19C L 8,19C(0,13) BLI=2
000684 07 FE BCR 15,14
000000 07 00 INIT1 BCR 0,0
000002 90 EC D 00C STM 14,12,00C(13)
000006 18 5D LR 5,13
000008 05 F0 BALR 15,0
00000A 98 9F F 006 LM 9,15,006(15)
00000E 07 FF BCR 15,15
000010 00000672 ADCON L4(INIT3)
000014 00000000 ADCON L4(EXITR)
000018 00000000 ADCON L4(INIT1)
00001C 00000318 ADCON L4(PGT)
000020 00000160 ADCON L4(TGT)
000024 00000370 ADCON L4(START)
000028 00000624 ADCON L4(INIT2)
00002C 96 12 1 034 OI 034(1),X'12'
000030 07 FE BCR 15,14
000032 FFFFFFFF DC X'FFFFFFFF'
000036 E3C5E2E3D9E4D540 DC X'E3C5E2E3D9E4D540'

```
      (A)    (B)    (C)      (D)
(7)  CARD   ERROR MESSAGE
     77     IEQ1080I-W     PERIOD PRECEDED BY SPACE. ASSUME END OF SENTENCE.
```

```
(8)  IEF285I   UT1.JOB3                                          DELETED
     IEF285I   VOL SER NOS= 111111.
     IEF285I   UT2.JOB3                                          DELETED
     IEF285I   VOL SER NOS= 111111.
     IEF285I   UT3.JOB3                                          DELETED
     IEF285I   VOL SER NOS= 111111.
     IEF285I   UT4.JOB3                                          DELETED
     IEF285I   VOL SER NOS= 111111.
     IEF285I   PUNCH                                             PASSED
     IEF285I   VOL SER NOS= 111111.
     IEF285I   SYSOUT                                            SYSOUT
     IEF285I   VOL SER NOS= SCRTCH.
     //STEP4  EXEC  PGM=IEWL,PARM='XREF'
     //SYSUT1 DD    DSNAME=&UT1,UNIT=SYSDA,SPACE=(TRK,(100,10))
     //SYSLMOD DD   DSNAME=GOJOB(GO),UNIT=SYSDA,SPACE=(TRK,(100,10,1)),     X
(1)  //             DISP=(NEW,PASS)
     //SYSLIB DD    DSNAME=SYS1.COBLIB,DISP=OLD
     //SYSPRINT DD  SYSOUT=A
     //SYSLIN DD    DSNAME=PUNCH,UNIT=SYSDA,SPACE=(TRK,(100,10)),          X
     //             DISP=(OLD,DELETE)
     IEF236I ALLOC. FOR JOB3     STEP4
     IEF237I SYSUT1   ON 190
(2)  IEF237I SYSLMOD  ON 190
     IEF237I SYSLIB   ON 190
     IEF237I SYSLIN   ON 190
```

```
(3)  E-LEVEL LINKAGE EDITOR OPTIONS SPECIFIED--XREF
(4) IEW0000   GO        NOW ADDED TO DATA SET


                                        ----  CROSS REFERENCE TABLE  ----
          (A)                                (B)
   CONTROL SECTION                         ENTRY
(5)
     NAME     ORIGIN  LENGTH                NAME   LOCATION     NAME   LOCATION     NAME   LOCATION     NAME   LOCATION
   TESTRUN       00     6AC
   IHDFDISP*    6B0     58C

       (A)          (B)              (C)
(6)  LOCATION  REFERS TO SYMBOL  IN CONTROL SECTION

        330           IHDFDISP       IHDFDISP
(5) ENTRY ADDRESS      00
    TOTAL LENGTH      C3C
```

```
   IEF285I   UT1.JOB3                                        DELETED
   IEF285I   VOL SER NOS= 111111.
   IEF285I   GOJOB                                           PASSED
   IEF285I   VOL SER NOS= 111111.
⑦  IEF285I   SYS1.COBLIB                                     KEPT
   IEF285I   VOL SER NOS= 111111.
   IEF285I   SYSOUT                                          SYSOUT
   IEF285I   VOL SER NOS= SCRTCH.
   IEF285I   PUNCH                                           DELETED
   IEF285I   VOL SER NOS= 111111.
   //STEP5  EXEC  PGM=*.STEP4.SYSLMOD
①  //SYSOUT DD    SYSOUT=A
   //SYSABEND DD  SYSOUT=A
   //SAMPLE DD    UNIT=2400,LABEL=(,NL)
   IEF236I ALLOC. FOR JOB3     STEP5
②  IEF237I PGM=*.DD ON 190
   IEF237I SAMPLE   ON 282
   STEP-1
   STEP-2
   STEP-3
   STEP-4
   STEP-2
   STEP-3
   STEP-2
   STEP-3
   STEP-2
   STEP-3
   STEP-2
   STEP-3
   STEP-2
   STEP-3
   STEP-2
   STEP-3
   STEP-2
   STEP-3
   STEP-2
   STEP-3
   STEP-2
   STEP-3
   STEP-2
   STEP-3
   STEP-2
   STEP-3
   STEP-2
   STEP-3
   STEP-2
   STEP-3
   STEP-2
   STEP-3
   STEP-2
   STEP-3
   STEP-2
   STEP-3
   STEP-2
   STEP-3
   STEP-2
   STEP-3
   STEP-2
   STEP-3
   STEP-2
   STEP-3
   STEP-2
   STEP-3
   STEP-2
   STEP-3
   STEP-5
   STEP-6
   STEP-7
   WORK-RECORD = A 0001 NYC Z
```

```
STEP-6
STEP-7
WORK-RECORD = B 0002 NYC 1
STEP-6
STEP-7
WORK-RECORD = C 0003 NYC 2
STEP-6
STEP-7
WORK-RECORD = D 0004 NYC 3
STEP-6
STEP-7
WORK-RECORD = E 0005 NYC 4
STEP-6
STEP-7
WORK-RECORD = F 0006 NYC Z
STEP-6
STEP-7
WORK-RECORD = G 0007 NYC 1
STEP-6
STEP-7
WORK-RECORD = H 0008 NYC 2
STEP-6
STEP-7
WORK-RECORD = I 0009 NYC 3
STEP-6
STEP-7
WORK-RECORD = J 0010 NYC 4
STEP-6
STEP-7
WORK-RECORD = K 0011 NYC Z
STEP-6
STEP-7
WORK-RECORD = L 0012 NYC 1
STEP-6
STEP-7
WORK-RECORD = M 0013 NYC 2
STEP-6
STEP-7
WORK-RECORD = N 0014 NYC 3
STEP-6
STEP-7
WORK-RECORD = O 0015 NYC 4
STEP-6
STEP-7
WORK-RECORD = P 0016 NYC Z
STEP-6
STEP-7
WORK-RECORD = Q 0017 NYC 1
STEP-6
STEP-7
WORK-RECORD = R 0018 NYC 2
STEP-6
STEP-7
WORK-RECORD = S 0019 NYC 3
STEP-6
STEP-7
WORK-RECORD = T 0020 NYC 4
STEP-6
STEP-7
WORK-RECORD = U 0021 NYC Z
STEP-6
STEP-7
WORK-RECORD = V 0022 NYC 1
STEP-6
STEP-7
WORK-RECORD = W 0023 NYC 2
STEP-6
STEP-7
WORK-RECORD = X 0024 NYC 3
STEP-6
STEP-7
WORK-RECORD = Y 0025 NYC 4
STEP-6
STEP-7
WORK-RECORD = Z 0026 NYC Z
STEP-6
STEP-8
```

```
IEF285I   GOJOB                                                  PASSED
IEF285I   VOL SER NOS= 111111.
IEF285I   SYSOUT                                                 SYSOUT
IEF285I   VOL SER NOS=       .
IEF285I   SYSOUT                                                 SYSOUT
IEF285I   VOL SER NOS=       .
IEF285I   AAAAAAAA.AAAAAAAA.AAAAAAAA.AAAAAAAA.00000008 DELETED
IEF285I   VOL SER NOS= LGL001.
IEF285I   GOJOB                                                  DELETED
IEF285I   VOL SER NOS= 111111.
```

③

```
START RDR,00C
START
IEC101A M 283,
A 0001 NYC 0
B 0002 NYC 1
C 0003 NYC 2
D 0004 NYC 3
E 0005 NYC 4
F 0006 NYC 0
G 0007 NYC 1
H 0008 NYC 2
I 0009 NYC 3
J 0010 NYC 4
K 0011 NYC 0
L 0012 NYC 1
M 0013 NYC 2
N 0014 NYC 3
O 0015 NYC 4
P 0016 NYC 0
Q 0017 NYC 1
R 0018 NYC 2
S 0019 NYC 3
T 0020 NYC 4
U 0021 NYC 0
V 0022 NYC 1
W 0023 NYC 2
X 0024 NYC 3
Y 0025 NYC 4
Z 0026 NYC 0
IEF101I RDR CLOSED
IEE007A READY.
```

COBOL library subroutines perform operations requiring such extensive codes that it would be inefficient to place them in the object module each time they are needed.

COBOL library subroutines are stored in the COBOL (F) library (SYS1.COBLIB). The required subroutines are inserted in load modules by the linkage editor.

There are five major categories of COBOL library subroutines:

- Conversion routines
- Arithmetic verb routines
- Input/output verb routines
- Sort feature interface routines
- Other verb routines

In addition, Q routines, which are not classified as COBOL library subroutines, are used to calculate the length of variable-length fields and the location of variably-located fields resulting from an OCCURS DEPENDING ON clause. They are stored in the Task or Subtask Global Tables in the object module.

COBOL LIBRARY CONVERSION SUBROUTINES

Eight numeric data formats are permitted in COBOL; five external (for input and output) and three internal (for internal processing).

The five external formats are: (1) external or zoned decimal, (2) external floating point, (3) sterling display, (4) report, and (5) sterling report. The three internal formats are: (1) internal or packed decimal, (2) binary, and (3) internal floating point.

The conversions from internal decimal to external decimal, from external decimal to internal decimal, and from internal decimal to report are done in-line. The other conversions are performed by the COBOL library subroutines shown in Table 17.

COBOL LIBRARY ARITHMETIC SUBROUTINES

Most arithmetic operations are performed in-line. However, involved calculations, such as exponentiation, and calculations with very large numbers, such as decimal multiplication of two 30-digit numbers, are performed by COBOL library subroutines. These subroutine names and their functions are shown in Table 18.

COBOL LIBRARY INPUT/OUTPUT SUBROUTINES

The input/output subroutines are for the verbs DISPLAY (TRACE and EXHIBIT), ACCEPT, and BSAM WRITE and CLOSE.

DISPLAY, TRACE, and EXHIBIT Subroutine (IHDFDISP)

The IHDFDISP subroutine is used to print, punch, or type data, usually in limited amounts, on an output unit. TRACE and EXHIBIT are kinds of DISPLAY.

The acceptable forms of data for this subroutine are:

1. Display
2. External decimal
3. Internal decimal (converted by the subroutine to external decimal)
4. Binary (converted by the subroutine to external decimal)
5. External floating point

Internal floating-point numbers must be converted to external floating point before the subroutine is called.

ACCEPT Subroutine (IHDFACPT)

The IHDFACPT subroutine is called to read from SYSIN or from the operator's console at execution time. For SYSIN, a maximum of 80 characters is accepted and moved to the specified operand. For the console, a maximum of 255 characters are accepted and either 255 characters or the length of the operand, whichever is smaller, is moved to the operand.

Table 17. Functions of COBOL Library Conversion Subroutine (Part 1 of 2)

Subroutine Name and Entry Points	Conversion	
	From	To
IHDFEFID	External Floating Point	Internal Decimal
IHDFEFBI	External Floating Point	Binary
IHDFEFIF	External Floating Point	Internal Floating Point
IHDFBIID[1] IHDFBID2[1] IHDFBID4[1]	Binary	Internal Decimal
IHDFBIED[1] IHDFBIX2[1] IHDFBIEX[1]	Binary	External Decimal
IHDFBIIF[2] IHDFBIFD[2] IHDFBIIL[3]	Binary	Internal Floating Point
IHDFTEFP[2] IHDFBIFL[2]	Binary	External Floating Point
IHDFIDEF	Internal Decimal	External Floating Point
IHDFIFEF	Internal Floating Point	External Floating Point
IHDFIDBI	Internal Decimal	Binary
IHDFEDBI	External Decimal	Binary
IHDFIDIF	Internal Decimal	Internal Floating Point
IHDFDCIF	External Decimal	Internal Floating Point
IHDFIFID	Internal Floating Point	Internal Decimal
IHDFIFEX	Internal Floating Point	External Decimal

[1]The entry points used depend on whether the double-precision number is in registers 0 and 1, 2 and 3, or 4 and 5, respectively.
[2]The entry points are for single-precision binary and double-precision binary, respectively.
[3]This entry point is used for calls from other COBOL library subroutines.

Table 17. Functions of COBOL Library Conversion Subroutines (Part 2 of 2)

Subroutine Name and Entry Points	Conversion	
	From	To
IHDFIFBI	Internal Floating Point	Binary integer and a power of 10 exponent
IHDFIFBX[3] IHDFIFBD	Internal Floating Point	Binary
IHDFIDSR	Internal Decimal	Sterling Report
IHDFIDST	Internal Decimal	Sterling Non-Report
IHDFSTID	Sterling Non-Report	Internal Decimal

[1]The entry points used depend on whether the double-precision number is in registers 0 and 1, 2 and 3, or 4 and 5, respectively.
[2]The entry points are for single-precision binary and double-precision binary, respectively.
[3]This entry point is used for calls from other COBOL library subroutines.

Table 18. Functions of COBOL Library Arithmetic Subroutines

Subroutine Name	Function
IHDFXMUL	Internal Decimal Multiplication (30 digits * 30 digits = 60 digits)
IHDFXDIV	Internal Decimal Division (60 digits/30 digits = 30 digits)
IHDFXPWR	Exponentiation of an Internal Decimal Base by a Binary exponent
IHDFFPWR	Floating-Point Exponentiation
IHDFGPWR[1]	Floating-Point Exponentiation

[1]The IHDFGPWR entry point is used if the exponent has a picture specifying an integer. The IHDFFPWR entry point is used in all other cases.

BSAM WRITE and CLOSE Subroutine (IHDFBSAM)

The IHDFBSAM routine is used only for the WRITE and CLOSE (output) statements for BSAM files.

SORT FEATURE SUBROUTINE (IHDFSORT)

The IHDFSORT routine acts as an interface between the COBOL calling program and the SORT/MERGE program.

COBOL LIBRARY SUBROUTINES FOR OTHER VERBS

There are also COBOL library subroutines for the comparisons, the verbs MOVE and TRANSFORM and the RERUN clause.

COMPARE Subroutine (IHDFVCOM)

The IHDFVCOM subroutine compares two operands, one or both of which is variable in length. They may exceed 256 bytes.

MOVE Subroutine (IHDFVMOV and IHDFVMVJ)

The MOVE subroutine is used when one or both operands is variable in length. They may exceed 256 bytes. The subroutine has two entry-points, depending on the type of move: IHDFVMOV (left-justified) and IHDFVMVJ (right-justified).

TRANSFORM Subroutine (IHDFVTRN)

The IHDFVTRN subroutine translates variable-length items.

Class Test Subroutine (IHDFCLAS)

The IHDFCLAS subroutine is used to perform class tests for variable-length items and those fixed length items over 256 bytes long.

Note: The following tables are placed in the library for use by the in-line coding generated and the subroutines called for by both class test and TRANSFORM:

IHDFATBL	IHDFITBL
IHDFETBL	IHDFTRAN

RERUN Subroutine (IHDCKPT)

The IHDCKPT subroutine is used to take checkpoints on a file specified on the RERUN clause. It saves registers, issues the checkpoint macro-instruction (CHKPT), restores the calling registers and returns control to the program.

When a file named in a RERUN clause is opened, a checkpoint counter for that file is set to a value of 1 in order to take a checkpoint before the first record is processed. Each time a READ or WRITE occurs, the compiler generates code to test the checkpoint counter. If the counter equals 1, the IHDCKPT subroutine is called, a checkpoint is taken, and the counter is reset to the integer value specified in the RERUN clause. If the counter is not equal to 1, it is decremented by 1 and no checkpoint is taken.

In this appendix, each field of the data control block is listed by the name of the operand of the assembler language macro instruction that can specify a value for that field. Tables 19 through 24 illustrate the data control blocks for the following file processing techniques: QSAM, BSAM, QISAM, BISAM, and BDAM. Some of the data control block fields, can be referred to with the DCB parameter of the DD statement. However, any field filled in by the COBOL compiler cannot be overridden.

Values for fields for which no entry appears in the column headed "COBOL Source" may be supplied by the DD statement or by the data set label.

For information concerning the specification of values for data control block fields, see the DCB macro-instruction for the different file processing techniques in the publication IBM System/360 Operating System: Supervisor and Data Management Macro-Instructions, Form C28-6647.

Note: The DCB subparameters are discussed under "User Defined Files" in the chapter "Execution Time Data Set Requirements."

• Table 19. Data Control Block for QSAM

Data Control Block Field	Explanation of Field	COBOL Source	Applicable DD Statement DCB Subparameters
DSORG	Data set organization	ORGANIZATION omitted ACCESS IS SEQUENTIAL or omitted	
MACRF	Type of macro instruction	OPEN INPUT OPEN OUTPUT OPEN I-O	
DDNAME	Name of DD statement	ASSIGN TO external-name	
DEVD	Device(s) on which the data set may reside and device-dependent information		[TRTCH=,DEN=, MODE=,STACK=])
OPTCD	Optional service provided by the control program		(OPTCD=[W|C|WC])
RECFM	Characteristics of the records in the data set	RECORDING MODE BLOCK CONTAINS WRITE...AFTER ADVANCING	
LRECL	Logical record length	RECORD CONTAINS	
BLKSIZE	Maximum length of block	BLOCK CONTAINS integer RECORDS	BLKSIZE=xxx (if integer=0 in BLOCK CONTAINS)
BFTEK	Type of buffering (S,E)	(COBOL specifies S)	
BUFNO	Number of buffers assigned to DCB	SAME AREA RESERVE	BUFNO=xxx
BFALN	Buffer boundary alignment	(COBOL compiler specifies double-word boundary)	
BUFCB	Address of buffer pool	SAME AREA	
EODAD[1]	Address of user's end-of-data-set exit routine for input data set	READ. . .AT END	
SYNAD	Address of error exit routine	USE AFTER ERROR ON file-name	
EROPT	Error option		(EROPT=[ACC|SKP|ABE])

[1]This field is filled in when the file is read.

• Table 20. Data Control Block for BSAM

Data Control Block Field	Explanation of Field	COBOL Source	Applicable DD Statement DCB Subparameters
DSORG	Data set organization	ORGANIZATION IS DIRECT RELATIVE ACCESS IS SEQUENTIAL	DSORG=DA (must be included for output files)
MACRF	Type of macro instruction	OPEN INPUT OPEN OUTPUT	
DDNAME	Name of DD statement	ASSIGN TO external-name	
DEVD	Device(s) on which data set may reside	ACTUAL KEY	
OPTCD	Optional service to be provided by the control program		(OPTCD=[option])
RECFM	Characteristics of the records in the data set	RECORDING MODE	
LRECL	Logical record length	RECORD CONTAINS	
BLKSIZE	Maximum block length	RECORD CONTAINS	
NCP	Maximum number of read or write macro-instructions issued before a check macro-instruction	Option 1 of the APPLY clause	
BUFNO	Number of buffers assigned to DCB	one	
EODAD[1]	Address of end-of-data-set exit (input)	READ...AT END	
SYNAD	Address of error exit	USE AFTER ERROR ON file-name	

[1]This field is filled in when the file is read.

• Table 21. Data Control Block for QISAM (load mode[1])

Data Control Block Field	Explanation of Field	COBOL Source	Applicable DD Statement DCB Subparameters
DSORG	Data set organization	ORGANIZATION INDEXED ACCESS IS SEQUENTIAL or omitted	DSORG=IS (required)
MACRF	Type of macro instruction	OPEN OUTPUT	
DDNAME	Name of DD statement	ASSIGN TO external-name	
OPTCD	Optional service provided by the control program[2]		OPTCD= [W|M|Y|I|R] (more than one code can be specified)
RECFM	Characteristics of the records in the data set	RECORDING MODE BLOCK CONTAINS	
LRECL	Logical record length	RECORD CONTAINS	
BLKSIZE	Maximum length of block	BLOCK CONTAINS integer RECORDS	BLKSIZE=xxx (IF integer=0 in BLOCK CONTAINS)
RKP	Relative position of record key in logical record	RECORD KEY clause	
NTM	Maximum number of cylinder index tracks		NTM=xx
KEYLEN	Length of key for each physical record	RECORD KEY	
CYLOFL	Number of overflow tracks for each cylinder		CYLOFL=xx
BUFNO	Number of buffers assigned to DCB	RESERVE	BUFNO=xxx
BFALN	Buffer boundary alignment	(COBOL compiler specifies double-word boundary)	
BUFCB	Address of buffer pool	SAME AREA	
SYNAD	Address of error exit routine	USE AFTER ERROR ON file-name or INVALID KEY statement	

[1]The load mode is used to create an indexed sequential file.
[2]At OPEN time, this field is updated to include the delete code option, in addition to any options specified in the DD statement.

Table 22. Data Control Block for QISAM (scan mode[1])

Data Control Block Field	Explanation of Field	COBOL Source	Applicable DD Statement DCB Subparameters
DSORG	Data set organization	ORGANIZATION INDEXED ACCESS IS SEQUENTIAL or omitted	DSORG=IS (required)
MACRF	Type of macro instruction	OPEN INPUT OPEN I-O	
DDNAME	Name of DD statement	ASSIGN TO external-name	
RECFM[2]	Characteristics of the records in the data set	RECORDING MODE BLOCK CONTAINS	
LRECL[2]	Logical record length	RECORD CONTAINS	
BLKSIZE[2]	Maximum length of block	BLOCK CONTAINS integer RECORDS	
BFTEK[2]	Type of buffering (S,E)	(COBOL specifies S)	
BUFNO	Number of buffers assigned to DCB	RESERVE	BUFNO=xxx
BFALN	Buffer boundary alignment	(COBOL compiler specifies double-word boundary)	
BUFCB	Address of buffer pool	SAME AREA	
EODAD	Address of user's end-of-file exit routine for files opened as input	READ...AT END	
SYNAD	Address of error exit routine	USE AFTER ERROR ON file-name	

[1]The scan mode is used to retrieve and update records in an indexed sequential file.
[2]The information for these fields is obtained from the data set label. It is stored there when the data set is created. However, the COBOL user must still specify the COBOL clauses in his source program.

Table 23. Data Control Block for BISAM

Data Control Block Field	Explanation of Field	COBOL Source	Applicable DD Statement DCB Subparameters
DSORG	Data set organization	ORGANIZATION IS INDEXED ACCESS IS RANDOM	DSORG=IS
MACRF	Type of macro instruction	OPEN I-O, READ, WRITE OPEN I-O, READ, REWRITE, WRITE OPEN INPUT, READ	
DDNAME	Name of DD statement	ASSIGN TO external name	
NCP	Number of channel programs to be established for this DCB	Option 1 of the APPLY clause	
MSWA	Address of main storage work area reserved for the control program. Required when new variable-length records are being added.	TRACK-AREA IS data-name TRACK-AREA IS integer CHARACTERS	
SMSW	Number of bytes reserved for main storage work area.	TRACK-AREA IS data-name TRACK-AREA IS integer CHARACTERS	

Table 24. Data Control Block for BDAM

Data Control Block Field	Explanation of Field	COBOL Source	Applicable DD Statement DCB Subparameters
DSORG	Data set organization	ORGANIZATION IS DIRECT RELATIVE ACCESS IS RANDOM	
MACRF	Type of macro instruction	OPEN INPUT, READ OPEN I-O, READ, REWRITE OPEN INPUT, READ, ACTUAL KEY OPEN I-O, READ, REWRITE, OPEN I-O, READ, REWRITE, WRITE	
DDNAME	Name of DD statement	ASSIGN TO external-name	
OPTCD	Optional service to be provided by the control program	APPLY RESTRICTED SEARCH	
RECFM	Characteristics of the records in the data set	RECORDING MODE IS U RECORDING MODE IS V RECORDING MODE IS F	
KEYLEN	Length of key for each physical record	SYMBOLIC KEY	
LIMCT	Maximum number of tracks or blocks to be searched (extended search)	APPLY RESTRICTED SEARCH ON integer TRACKS no APPLY RESTRICTED SEARCH	

In general, compilation is faster if:

1. Options in the EXEC statement are specified
   a. making more main storage available (the SIZE option),
   b. optimizing the space available for buffers (the BUF option),
   c. suppressing output (the NOSOURCE, NODECK, NOLOAD, and the NOMAP options).
   d. suppressing object code listing and object and linkage edit decks (the SUPMAP option only if E level messages are present).
   e. specifying the option CLIST, which will result in a condensed listing.
2. The maximum block size for a compiler data set is specified.
3. A disk configuration and separate channels for utility data sets are used.
4. Separate devices (i.e., not the same direct-access unit) on the same channel are used.

Compilation time is also affected by the speed of the devices allocated to the data sets. For example, a tape device is faster than a printer for printed output. The blocking information that follows applies to PCP, MFT, and MVT.

BLOCK SIZE FOR COMPILER DATA SETS

The blocking factor specified for compiler data sets other than utility data sets must be permissible for the device the data set is on. In addition, for the SYSLIN data set, it must be permissible for the linkage editor used. (Any block size specified for a utility data set in a DD statement is overridden by the compiler.) If a block size other than the default option is needed, it can be requested by specifying the BLKSIZE subparameter of the DCB parameter in the DD statement for the data sets. The format of the subparameter is:

```
DCB=(,BLKSIZE=nnn)
```

where nnn is equal to N times the logical record size in bytes, and $1 \leq N \leq M$. M is equal to the blocking factor permissible for the device, and, in the case of SYSLIN, to the blocking factor permissible for the linkage editor used.

If blocking is desired, the record format for SYSPRINT [DCB=(,RECFM=nnn)] should be specified as FBA. The record format for SYSIN, SYSLIN, SYSPUNCH, and SYSLIB should be specified as FB.

The logical record size for SYSPRINT is 121 bytes. The logical record size for SYSIN, SYSLIN, SYSPUNCH, and SYSLIB is 80 bytes.

Note: For compile, linkage edit, and execute cases when labeled volumes are used, RECFM and BLKSIZE must be given for SYSLIN in the compile step only. If BLKSIZE is specified for SYSPUNCH, LRECL must also be specified. The 44K version of the linkage editor supports input data sets with a blocking factor of up to 40 specified.

HOW BUFFER SPACE IS ALLOCATED TO BUFFERS

Once the amount of space available for a compilation is determined, the compiler subtracts the amount required for itself. From the space remaining, it then computes the space available for utility and input/output data set buffers. If space still remains, it is used for the compiler's internal processing.

Once the compiler determines the amount of space available for buffers, it calculates how this space is to be divided. First, it computes the amount of space required for the buffers of the input/output data sets. From the space remaining, it determines the maximum buffer size, and hence block size, possible for a utility data set. The utility data sets all have the same block size. Hence, the block size of a utility data set is dependent on the amount of space available for buffers. If a block size has been specified on a DD statement for a utility data set, it is overridden.

A larger buffer size for a utility data set allows for faster processing. However,

if the program that is being compiled takes up a large amount of the available storage, a smaller space for buffers enables the compiler to use more main storage for internal processing.

The following describes how the space available for buffers is determined and how it is allocated for buffers.

Let A represent the space that can be allocated to these buffers. It is determined as follows:

1. If neither the BUF nor the SIZE option of the PARM parameter of the EXEC statement is specified, A equals the default value for buffer space. This value is specified at system generation time. The minimum value is 2762 bytes.

2. If the SIZE option is specified, but BUF is not, A equals (SIZE-80K)/4 plus the default value for buffer space.

3. If BUF is specified (whether or not SIZE is specified), A equals the value specified for BUF.

4. If BUF is smaller than 2762 bytes (the minimum value), a warning message is printed and the minimum value is assumed. If BUF is too large to allow minimum table space for compilation, a warning message is printed and the default value (or the minimum value, if the default value is also too large) is assumed.

The programmer must make sure that the amount of buffer space allocated by the system is sufficient, taking into consideration the block sizes specified for the compiler data sets. The allocated buffer space is divided as follows:

1. Let B represent the amount of buffer space to be allocated for input/output data sets. B is computed as either equal to:

   2 times the block size of SYSPRINT + SYSIN + SYSLIB

   or

   2 times the block size of SYSPRINT + SYSPUNCH + SYSLIN

whichever is larger. The maximum allowable value of B is A-1280 bytes. If the computed value is greater than the maximum allowable value, a diagnostic message is printed and compilation is abandoned.

If the block sizes are not specified in the DD statements, the following default values are assumed:

SYSIN, SYSLIN, SYSPUNCH, SYSLIB: 80 bytes each

SYSPRINT: 121 bytes

2. Let C represent the amount of buffer space to be allocated for each utility data set. Therefore, C equals the block size of data sets, SYSUT1, SYSUT2, SYSUT3 and SYSUT4, respectively.

   If $A \leq 6B$, then $C = \frac{A-B}{5}$

   If $A > 6B$, then $C = \frac{A}{6}$

If C > maximum block size permitted for any device a utility data set is on, then the maximum block size is the value chosen for C. The minimum block size for a utility data set is 255 bytes.

The COBOL (F) compiler can be invoked by a problem program at execution time through the use of the ATTACH or LINK macro instruction, i.e., dynamic invocation. Dynamic invocation of COBOL (F) compiled programs can be accomplished through the use of the LINK or LOAD macro instruction.

INVOKING THE COBOL (F) COMPILER

The problem program must supply the following information to the COBOL (F) compiler:

- The options to be specified for the compilation
- The ddnames of the data sets to be used during processing by the COBOL (F) compiler
- The header to appear on each page of the listing

```
┌--------┬---------┬----------------------┐
|Name    |Operation|Operand               |
├--------┼---------┼----------------------┤
|[symbol]|LINK     |EP=IEQCBL00           |
|        |ATTACH   |PARAM=(optionlist     |
|        |         |  [,ddnamelist],      |
|        |         |  [,headerlist]),VL=1 |
└--------┴---------┴----------------------┘
```

EP
: specifies the symbolic name of the COBOL (F) compiler. The entry point at which execution is to begin is determined by the control program (from the library directory entry).

PARAM
: specifies, as a sublist, address parameters to be passed from the problem program to the COBOL (F) compiler. The first fullword in the address parameter list contains the address of the COBOL option list. The second fullword contains the address of ddname list. If standard ddnames are to be used and no header list specified, this list may be omitted. If standard ddnames are to be used and a header list is specified, this entry should contain the address of a word of binary zeros, aligned on a halfword. The last fullword contains the address of the header list. This list may be omitted.

option list
: specifies the address of a variable length list containing the COBOL options which are specified for compilation. See the description of the EXEC statement "Job Control Procedures" for more details. This address must be written even though no list is provided.

  The option list must begin on a halfword boundary. The two high-order bytes contain a count of the number of bytes in the remainder of the list. If no options are specified, the count must be zero. The option list is free form with each field separated by a comma. No blanks or zeros should appear in the list.

ddname list
: specifies the address of a variable length list containing alternative ddnames for the data sets used during COBOL (F) compiler processing. If standard ddnames are used, this operand may be omitted.

  The ddname list must begin on a halfword boundary. The two high-order bytes contain a count of the number of bytes in the remainder of the list. Each name of less than eight bytes must be left justified and padded with blanks. If an alternate ddname is omitted from the list, the standard name will be assumed. If the name is omitted within the list, the 8-byte entry must contain binary zeros. Names can be omitted from the end by merely shortening the list.

The sequence of the 8-byte entries in the ddname list is as follows:

Entry	Alternate Name For:
1	SYSLIN
2	not applicable
3	not applicable
4	SYSLIB
5	SYSIN
6	SYSPRINT
7	SYSPUNCH
8	SYSUT1
9	SYSUT2
10	SYSUT3
11	SYSUT4

header list
specifies the address of a variable length list containing information to be included in the heading on each page of the listing. The list must begin on a halfword boundary. The two high-order bytes should contain a count of the number of bytes in the new heading information; the next four bytes of the list should contain the page number at which the heading is to start, in EBCDIC format.

VL=1
specifies that the sign bit is to be set to 1 in the last fullword of the address parameter list.

When the COBOL compiler completes processing, a return code is returned in register 15. See the discussion of the COND parameter in "Job Control Procedures" for further details.

Linkage Editor control cards should be specified as follows:

1. For the PROGRAM-ID external name, a NAME card.

2. For each ENTRY external name, an ALIAS card. (All aliases must be specified on one card, however.) Since no more than five aliases may be specified for one load module, only six entries (or five entries and the PROGRAM-ID) should be specified in a COBOL program which is to be dynamically invoked.

Limitations on the size of a COBOL source program should be considered in relation to the capacities of both the COBOL (F) compiler and the various linkage editors. This appendix contains information to aid the programmer in determining how his source program affects space usage at compilation time and linkage editing time.

COMPILER CAPACITY

The capacity of the COBOL (F) compiler is limited by two general conditions: the total contiguous space available must be sufficient for compilation, and an individual table may not have a length greater than 32,767 bytes. If either of these conditions are not met during compilation, one of the following error messages will be issued:

IEQ000II - SIZE PARAMETER TOO SMALL FOR THIS PROGRAM.

IEQ00101 - A TABLE HAS EXCEEDED THE MAXIMUM PERMISSIBLE SIZE.

In either case, compilation is terminated. However, in the first case, the program may be recompiled with a larger size parameter.

The following information can be used in planning or adjusting a COBOL source program in order to permit compilation:

1. Procedure-name definitions and references: In any one section of the Procedure Division, the number of unresolved procedure-name definitions and procedure-name references may not exceed 900. A procedure-name that is defined in a section of the program is unresolved until the section is reached. A procedure-name reference is unresolved until the procedure-name itself is defined and resolved. Keeping forward referencing to a minimum, then, would increase the amount of space available.

2. Qualification of names: Name qualification requires a considerable amount of space, including increased table lengths, and should be avoided when possible. Some of the areas where qualification uses table space are:

   a. RECORD KEY, SYMBOLIC KEY, and ACTUAL KEY clauses.

   b. Procedure-name processing. No more than 900 procedure-names may be qualified in a program.

   c. Report Writer. Information concerning each Report Description entry (generated labels of applicable routines, controls, sums, headings, footings, etc.) is maintained in tables during compilation.

      Some of this information is released from the tables at the end of the report group, some at the end of the Report Description entry, and some at the end of the Report Section. Name qualification and subscripting make the information tables still larger. No more than 30 reports can be processed by the compiler without approaching the limits on the size of these tables.

3. Error table: In order to avoid rewinding or rereading a file, a table is maintained during processing containing notation about errors encountered during compilation. No more than 3,000 entries can be made in this table.

4. Address constant table: In order to reduce the amount of space used by text records that are out of sequence, the COBOL (F) compiler stores items that will be used as address constants in an address constant table. The maximum number of entries in this table is 4644. The number of entries made in this table will be governed mostly by the number of each of the following items in the program:

   a. Files (BSAM,BDAM,ISAM)

   b. Files for which a SAME AREA clause is specified

   c. Referenced procedure-names

   d. Generated procedure-names (from IF, ONSIZE ERROR, AT END)

   e. Varied-names. Two for each ALTER or PERFORM procedure-name or for the first procedure-name of PERFORM THRU.

   f. USE sentence

5. Miscellaneous: The appearance of the following items in a program causes entries to be made into tables, which can greatly reduce the amount of total space available for compilation:

   a. Procedure-names (no more than 16,380 may be specified); note, however, further restrictions on number of procedure-names that can be referred to.

   b. Literals (no more than 16,380 may be specified; total length of literals may not exceed 32,760 bytes)

   c. OCCURS...DEPENDING ON clauses

   d. Subscripting

   e. Intermediate arithmetic results

   f. Complex arithmetic expressions

   g. Complex logical expressions

LINKAGE EDITOR CAPACITY

Some COBOL program and linkage editor considerations are listed below as a further guide in preparing a source program. Consult the publication IBM System/360 Operating System: Linkage Editor, Form C28-6538, for additional information on linkage editor capacities and processing.

1. A COBOL (F) object program consists of a single CSECT (control section). The size of the object module may be determined by looking at the location of the last instruction in INIT3 in the object code listing (see the section entitled "Output") or from the END card.

2. The size of the object module is greatly increased by any of the following:

   a. The blocking factor and alternate area reservation of basic files

   b. The specification of the SAME AREA clause for queued files

3. RLD (Relocation List Dictionary) cards are part of the load module, and are used by the linkage editor to compute the address constants for the load module. The number of RLDs produced by the compiler can be determined by the following formula:

   No. of RLDs = (no. of unique subprograms called) + (no. of COBOL (F) library routines called) + (no. of ENTRY statements) + 7.

4. The output text of the compiler is written out in a sequence that differs from the order indicated by the location counters contained in each output item. This sequence difference may result in a strain on the facilities of the linkage editor. The output items are written out in the following order:

   a. Constant global table items, including TALLY, random processing information, virtuals, literals

   b. Procedure Division

   c. INIT1

   d. DCBs and DECBs

   e. Working-Storage value clauses

   f. Address constants in DCBs, DECBs, buffer control blocks, etc.

   g. Global table adcons

   The location counter for each item indicates the following order:

   a. 4-c

   b. 4-d and 4-f intermixed, with overlays

   c. 4-e

   d. 4-a and 4-g intermixed, with no overlays

   e. 4-b

5. Value clauses in the Working-Storage Section may result in many discontinuous text records.

6. The object module produced by the COBOL (F) compiler may not be sorted prior to the linkage editor step.

This appendix contains a brief summary of I/O error conditions for each of the file processing techniques. More detailed information on error conditions can be found by consulting the publications, IBM System/360 Operating System: Supervisor and Data Management Macro-Instructions, Form C28-6647, and IBM System/360 Operating System: System Control Blocks, Form C28-6628.

QSAM/BSAM File Processing Techniques

Register 1 contains error bits detailing the exact cause of an error. Conditions causing input/output errors and suggested user responses are as follows:

I/O Error Condition	Suggested User Response
1. Input Error 2. Output Error 3. Invalid Request (BSAM only)	For BSAM, display the error message. Processing of the file is limited to CLOSE. For QSAM, display the error message and then execute the EROPT option in the DD statement. (The EROPT option gives the user three choices: ACC-accept the error block and continue processing SKP-skip to the next block. ABE-terminate the job.)

BDAM File Processing Technique

The DECB contains two error condition bytes at location DECB + 4. Conditions causing input/output errors and suggested user responses are as follows:

I/O Error Condition	Suggested User Response
1. Record Not Found 2. Invalid Request a. Requested block outside data set. b. Attempted to add a fixed-length record with key beginning with HEX FF.	Condition caused by an invalid key. Processing of the file may be continued.
1. Uncorrectable I/O Error. 2. Uncorrectable Error, Not I/O	Processing of the file is limited to CLOSE.
1. Space Not Found	Processing of the file may be continued. If option 1 of the APPLY clause is specified for the file, the record may be written by changing the actual key. CLOSE, READ or REWRITE may be executed for the file.

QISAM File Processing Technique

The DCB contains two error condition bytes named EXCD1 and EXCD2, at location DCB + 80. Conditions causing Input/Output errors and suggested user responses are as follows:

I/O Error Condition	Suggested User Response
1. Sequence Check. 2. Duplicate Record.	Condition caused by an invalid key. Processing of the file may be continued.
1. Space Not Found. 2. Uncorrectable Output Error. 3. Unreachable Block (Input). 4. Unreachable Block (Update).	Processing of the file is limited to CLOSE.
1. Uncorrectable Input Error.	The user may attempt to bypass the block containing the error. If, in reading the next block, the error does not recur, he may continue processing without closing the file. If the error persists, processing of the file is limited to CLOSE.

BISAM File Processing Technique

The DECB contains an error condition byte at location DECB + 24. Conditions causing input/output errors and suggested user responses are as follows:

I/O Error Condition	Suggested User Response
1. Record Not Found. 2. Duplicate Record.	Condition caused by an invalid key. Processing of the file may be continued.
1. Invalid Request.	Processing of the file is limited to CLOSE.
1. Uncorrectable I/O Error. 2. Unreachable Block--Index cannot be read. 3. Record Length Check.	The user can try to execute the instruction again. If the error persists, he should close the file.
1. Space Not Found.	Processing of the file may be continued. The record may be written after changing the keys and executing a write if a cylinder overflow area is available for the new value of the keys. CLOSE or READ may be executed for the file.

Figure 49 contains an error processing summary for the Indexed Sequential file processing techniques.

File Processing Technique	Type of Error	Error Processing Routine Specified		
		USE AFTER STANDARD ERROR	INVALID KEY	None
QISAM WRITE (load mode)	Uncorrectable I/O Error	User must terminate processing on the file and reload it.	COBOL executes a STOP RUN with a return code of 16. User must reload the file.	Abnormal Termination
	Space Not Found			
	Duplicate Record	User must drop the invalid record if he wishes to continue.	User must drop the invalid record if he wishes to continue.	
	Sequence Check			
QISAM READ REWRITE (scan mode)	Uncorrectable I/O Error	User must terminate processing on the file and reload it. The user may attempt to bypass the block in error and continue processing. However, if the error persists he should reload his program.	Not applicable.	Abnormal Termination
	Unreachable Block	Processing may continue but user must not process the file in error.		
BISAM	Uncorrectable I/O Error	Processing may continue but user must not process the record in error.	COBOL executes a STOP RUN with a return code of 16. User must reload the file.	Errors are ignored and processing continues with unpredictable results.
	Unreachable Block			
	Record Length Check			
	Space Not Found	Processing may continue.		
	Record Not Found	User may change the key and attempt the same operation. However, if the error persists he should drop the invalid record in order to continue.	User may change the key and attempt the same operation. However, if the error persists he should drop the invalid record in order to continue.	
	Duplicate Record			

• Figure 49. Error Processing Summary for ISAM Files

This section describes diagnostic messages generated by the system. Compiler messages and object time messages are discussed.

COMPILER DIAGNOSTIC MESSAGES

Using one of the messages as an example, COBOL (F) compiler messages are of the following format:

```
005  IEQ4046I-E  ILLEGAL TO REWRITE*****,A
                 BSAM FILE. STATEMENT
                 DELETED.
```

The code 005 is the card number of the statement where the error has occurred. IEQ identifies this as a COBOL (F) compiler message. 4046 is the identifying number of the message. The symbol I means that this is a message to the programmer for his action. E is a level of severity in the error code that is, as follows:

- W Warning -- This level indicates that an error was made in the source program. However, it is not serious enough to hinder the execution of the program.
- C Conditional -- This level indicates that an error was made but the compiler usually makes a corrective assumption. The statement containing the error is retained. Execution can be attempted for the debugging value.
- E Error -- This level indicates that a serious error was made. Usually the compiler makes no corrective assumption. The statement containing the error is dropped. Execution of program should not be attempted.
- D Disaster -- This level indicates that a serious error was made. Compilation is not completed. Results are unpredictable.

The severity level of the messages is correlated with the return code issued by the compiler at the end of compilation (see below). This code can be tested by the programmer with the COND parameter in the EXEC statement.

Severity Level	Return Code
(No messages)	0
W	4
C	8
E	12
D	16

The return code indicated is the most severe level occurring in the messages.

The message text is composed of two sentences. The first sentence describes the error; the second describes what the compiler has done as a result of the error. Most of the messages are thus self-explanatory, except in two situations:

1. When no compiler action is given. These messages are numbered in the 3000 series. They appear in combination with other messages that do have the compiler action described.
2. When messages describe errors that require an explanation too long to include in a message. These explanations appear in text under the messages.

Words in a message that must vary according to the program being compiled are denoted by ***** in the messages printed below. The symbol *** appearing in a message on a listing means that the compiler has encountered unrecognizable information.

IEQ0001I- SIZE PARAMETER TOO SMALL FOR THIS PROGRAM

Explanation: Compiler was unable to allocate sufficient table space for the source program.

User Response: Specify a larger amount of main storage in the SIZE option of the PARM parameter in the EXEC statement, or divide the program into smaller segments and utilize the overlay feature of the linkage editor.

IEQ0002I- INVALID COPY/BASIS LIBRARY NAME

Explanation: Library name given in BASIS, COPY, or INCLUDE statement does not exist.

User Response: Check that a SYSLIB DD statement is present that defines a library which contains the member whose name is speci-

fied on the BASIS, COPY, or INCLUDE statement.

IEQ0003I- PERMANENT I/O ERROR ON *****. COMPILATION ABANDONED.

Explanation: ***** is the name of the DD statement associated with the data set on which the input/output error occurred.

Note: The above message appears on the console listing only.

IEQ0004I- ILLEGAL PARM OPTION ON EXEC STATEMENT.

IEQ0005I- ***** INVALID BLOCKSIZE. DEFAULT USED.

Explanation: Blocksize specified on DD card is not an integral multiple of record length, or blocking factor is too large.

User Response: Change blocksize parameter on DD card or accept default option.

IEQ0006I- ***** DATASET NOT USABLE. JOB STEP CANNOT EXECUTE.

Explanation: Data set required by the compiler cannot be opened. Program cannot be compiled.

User Response: Check DD card.

IEQ0007I- SYSLIN NOT USABLE. LOAD OPTION CANCELLED.

Explanation: SYSLIN cannot be opened.

User Response: Check DD card.

IEQ0008I- SYSPUNCH NOT USABLE. DECK OPTION CANCELLED.

Explanation: SYSPUNCH cannot be opened.

User Response: Check DD card.

IEQ0009I- SIZE BUF PARAMETER IGNORED. DEFAULT USED.

Explanation: Value of parameters is inadequate to calculate table and dictionary space for compilation purposes, or BUF parameter is less than minimum required.

User Response: If the default value is unsuitable, increase the value for the SIZE parameter or decrease the BUF parameter.

IEQ0010I- A TABLE HAS EXCEEDED THE MAXIMUM PERMISSIBLE SIZE.

IEQ0015I- BUF SUBPARAMETER IN PARM FIELD TOO SMALL FOR DD CARD BLKSIZE SUBPARAMETERS. COMPILATION ABANDONED.

Explanation: The size of the buffer used for SYSPUNCH, SYSLIN, SYSIN, SYSPRINT, and SYSLIB exceeds BUF-1280 bytes.

User Response: Increase the BUF parameter in the EXEC card or decrease the BLKSIZE values specified in the DD cards for SYSPRINT, SYSIN, SYSLIB, SYSPUNCH, or SYSIN.

IEQ0020I- LOGIC ERROR OR MACHINE ERROR IN TABLE HANDLER. COMPILATION ABANDONED.

IEQ6001I- ERROR FOUND PROCESSING F4 TEXT. 00 CODE IN LISTING TEXT.

IEQ6002I- ERROR FOUND PROCESSING F4 TEXT. END OF LISTING TEXT REACHED.

IEQ6003I- ERROR FOUND PROCESSING F4 TEXT. UNKNOWN DATA A-TEXT CODE.

IEQ6004I- ERROR FOUND PROCESSING F4 TEXT. EOF REACHED WHILE PROCESSING LISTING TEXT.

IEQ6005I- ERROR FOUND PROCESSING F3 TEXT. COMPILATION ABANDONED.

IEQ6006I- MAP SUPPRESS SPECIFIED AND E LEVEL DIAGNOSTIC HAS OCCURRED. ONLY DIAGNOSTICS WILL BE PRODUCED.

Note: The preceding messages may be interspersed in the compiler output listing. The following messages are grouped in the compiler output listing.

IEQ1001I- NUMERIC LITERAL NOT RECOGNIZED AS LEVEL NUMBER BECAUSE ***** ILLEGAL AS USED. SKIPPING TO NEXT LEVEL, SECTION OR DIVISION.

IEQ1002I- ***** SECTION HEADER MISSING. ASSUMED PRESENT.

IEQ1003I- ***** PARAGRAPH NAME MISSING. ASSUMED PRESENT.

IEQ1004I- INVALID WORD *****. SKIPPING TO NEXT RECOGNIZABLE WORD.

IEQ1005I- INVALID ORDER IN ENVIRONMENT DIVISION. SKIPPING TO NEXT DIVISION.

IEQ1006I- DECLARATIVES SECTION WITHOUT USE SENTENCE. CONTINUING.

IEQ1007I- ***** NOT PRECEDED BY A SPACE. ASSUME SPACE.

IEQ1008I- RIGHT PAREN SHOULD NOT BE PRECEDED BY SPACE.

IEQ1009I- 'INCLUDE' SHOULD BE PRECEDED BY PROCEDURE-NAME. IGNORED.

IEQ1010I- LEFT PAREN SHOULD NOT BE FOLLOWED BY SPACE.

IEQ1011I- RECORDING MODE SPECIFICATION IS INVALID. ASSUMED VARIABLE.

IEQ1012I- FILE-NAME NOT UNIQUE. USING FIRST DEFINITION.

IEQ1013I- CHARACTER LENGTH IN SPECIAL-NAMES SHOULD BE ONE.

IEQ1014I- DEVICE SPECIFICATION NOT FOUND IN ASSIGN CLAUSE. CONTINUING.

IEQ1015I- ***** INVALID AS EXTERNAL-NAME. IGNORED.

IEQ1016I- MORE THAN ONE ***** CLAUSE. SKIPPING TO NEXT CLAUSE.

IEQ1017I- ***** INVALID IN ***** CLAUSE. SKIPPING TO NEXT CLAUSE.

IEQ1018I- COPY CLAUSE INVALID IN A COPY LIBRARY. IGNORED.

IEQ1019I- NO LIBRARY NAME. COPY CLAUSE IGNORED.

IEQ1020I- ***** SHOULD BE PROCEDURE-NAME FOLLOWING *DEBUG. *****.

IEQ1021I- ***** DOES NOT BELONG ON *DEBUG CARD. SKIPPING TO NEXT CARD.

IEQ1022I- PERIOD DOES NOT BELONG ON *DEBUG CARD. DELETED.

IEQ1023I- INVALID FILE-NAME. USE IGNORED.

IEQ1024I- UNDEFINED FILE-NAME. USE IGNORED.

IEQ1025I- REDEFINES CLAUSE NOT FIRST CLAUSE FOLLOWING DATA-NAME. ASSUMED FIRST.

IEQ1026I- SEARCH ASSUMED FOLLOWING RESTRICTED.

IEQ1027I- RESTRICTED ASSUMED TO PRECEDE SEARCH.

IEQ1028I- ***** SENTENCE IMPROPERLY WRITTEN. SENTENCE IGNORED.

IEQ1029I- ***** IN ***** SENTENCE NOT DEFINED AS FILE-NAME. NAME IGNORED.

IEQ1030I- ***** IN ***** SENTENCE IS INVALID. WORD IGNORED.

IEQ1031I- USE SENTENCE NOT PRECEDED BY SECTION-NAME. SECTION-NAME ASSUMED.

IEQ1032I- ***** INCORRECTLY USED IN USE SENTENCE. SENTENCE IGNORED.

IEQ1033I- ***** FILE-NAME PREVIOUSLY ASSIGNED SAME AREA. USING FIRST ONE.

IEQ1034I- ***** CLAUSE ILLEGAL IN ***** LEVEL. SKIPPING TO NEXT VALID CLAUSE.

IEQ1035I- DETAIL NAME NOT UNIQUE. SKIPPING TO NEXT 01.

IEQ1036I- QUALIFIED NAME INVALID AFTER LEVEL NUMBER. USING LOWEST NAME.

IEQ1037I- ***** INVALID IN DATA DESCRIPTION. SKIPPING TO NEXT CLAUSE.

IEQ1038I- ***** INVALID AFTER LEVEL NUMBER. SKIPPING TO NEXT LEVEL.

IEQ1039I- DATA-NAME IN ***** CLAUSE NEED NOT BE QUALIFIED. USING LOWEST NAME.

IEQ1040I- IMPROPER LEVEL NUMBER FOR FILE-SECTION.

IEQ1041I- ***** INVALID AS USED IN ***** SECTION. SKIPPING TO NEXT LEVEL, SECTION OR DIVISION.

IEQ1042I- ASSIGN CLAUSE MISSING IN SELECT. CONTINUING.

IEQ1043I- END OF SENTENCE SHOULD PRECEDE *****. ASSUMED PRESENT.

IEQ1044I- MISSING DATA RECORDS CLAUSE IN FD.

IEQ1045I- INVALID ORDER IN ***** SECTION.

IEQ1046I- MEMBER NOT FOUND IN LIBRARY. IGNORING COPY.

IEQ1047I- LIBRARY NOT FOUND ON SYSTEM. IGNORING COPY.

IEQ1048I- LIBRARY MEMBER HAS BAD TRACK. IGNORING REST OF COPY.

IEQ1049I- ***** PREVIOUSLY ASSIGNED OVERFLOW-NAME. THIS ONE DISCARDED.

IEQ1050I- ***** FILE ALREADY ASSIGNED THIS APPLY OPTION. FILE-NAME IGNORED.

IEQ1051I- NO DATA-NAME IN USE SENTENCE. SENTENCE IGNORED.

IEQ1052I- ***** ILLEGALLY USED IN USE SENTENCE. END SENTENCE, RESCANNING AT NEXT RECOGNIZABLE WORD.

IEQ1053I- ***** CLAUSE INCOMPLETE. CLAUSE IGNORED.

IEQ1054I- SELECT CLAUSE FOR ***** IGNORED. NOT LEGAL FOR SD.

IEQ1055I- VALID FILE-NAME NOT PRESENT. DESCRIPTION IGNORED.

IEQ1056I- FILE-NAME NOT DEFINED IN A SELECT. DESCRIPTION IGNORED.

IEQ1057I- FIRST WORD IN REPORT SECTION NOT RD. IGNORED.

IEQ1058I- NO REPORTS CLAUSE IN FILE SECTION. REPORT SECTION IGNORED.

IEQ1059I- NO REPORT CLAUSE FOR RD. RD IGNORED.

IEQ1060I- INVALID WORD IN RW VERB CLAUSE. IGNORED.

IEQ1061I- DUPLICATE CLAUSE. DROPPED.

IEQ1062I- WORD INVALID AS BCD NAME.

IEQ1063I DUPLICATE ENTRY IN PAGE CLAUSE. DROPPED.

IEQ1064I- NO TYPE CLAUSE SPECIFIED. SKIPPING TO NEXT 01.

IEQ1065I- INTEGER MISSING IN PAGE CLAUSE. ENTRY IGNORED.

IEQ1066I- INVALID WORD IN PAGE CLAUSE. SKIPPING TO NEXT RECOGNIZABLE WORD.

IEQ1067I- INVALID HEADER. SKIPPING TO NEXT RECOGNIZABLE WORD.

IEQ1068I- OPERAND FOR GENERATE NOT FOUND. CLAUSE DROPPED.

IEQ1069I- INVALID TYPE CLAUSE. SKIPPING TO NEXT 01.

IEQ1070I- FLT-PT LIT MANTISSA EXCEEDS 16 DIGITS. TRUNCATED TO 16.

IEQ1071I- FLT-PT LIT EXPONENT EXCEEDS 2 DIGITS. TRUNCATED TO 2. RESCANNING.

IEQ1072I- FLT-PT LIT EXPONENT FOLLOWED BY NON-BLANK. RESCANNING AT NON-BLANK.

IEQ1073I- FLT-PT LIT E FOLLOWED BY INVALID CHARACTER. RESCANNING AT E.

IEQ1074I- FLT-PT LIT SIGN FOLLOWED BY INVALID CHARACTER. RESCANNING AT E.

IEQ1075I- FLT-PT LIT EXCEEDS LIMIT. ASSUME MAX OR MIN PER SIGN OF EXPONENT.

IEQ1076I- ALPHANUMERIC LIT EXCEEDS 120 CHARACTERS. TRUNCATED TO 120.

IEQ1077I- ALPHANUMERIC LIT CONTINUES IN A-MARGIN. ASSUME B-MARGIN.

IEQ1078I- ALPHA-LITERAL CONTINUED WITH MISSING HYPHEN OR QUOTE. ASSUMED.

IEQ1079I- ALPHANUMERIC LIT HAS ZERO LENGTH. ASSUME ONE SPACE.

IEQ1080I- PERIOD PRECEDED BY SPACE. ASSUME END OF SENTENCE.

IEQ1081I- PERIOD NOT FOLLOWED BY BLANK. ASSUME END OF SENTENCE.

IEQ1082I- NUMERIC LIT EXCEEDS 18 DIGITS. TRUNCATED TO 18.

IEQ1083I- ILLEGAL CHARACTER. SCAN RESUMED AT NEXT VALID CHARACTER.

IEQ1084I- COMMA SHOULD NOT BE PRECEDED BY BLANK. ASSUMED OK.

IEQ1085I- WORD OR PICTURE EXCEEDS 30 CHARACTERS. TRUNCATED TO 30 CHARACTERS.

IEQ1086I- ***** SHOULD BEGIN A-MARGIN.

IEQ1087I- '*****' SHOULD NOT BEGIN A-MARGIN.

IEQ1088I- MISSING FIRST INSERT OR DELETE CARD. PASS CARDS UNTIL FOUND. *****.

IEQ1089I- INSERT OR DELETE NUMBER OUT OF SEQUENCE. SKIPPING TO NEXT INSERT OR DELETE NUMBER. *****.

IEQ1090I- DELETE THRU NUMBER OUT OF SEQUENCE. PASS CARDS UNTIL NEXT INSERT OR DELETE. *****.

IEQ1091I- ***** IN A-MARGIN NOT VALID AS PROC-NM. ASSUME B-MARGIN.

IEQ1092I- DECLARATIVES DO NOT FOLLOW PROCEDURE DIVISION. IGNORED.

IEQ1093I- NO DECLARATIVES SECTION. END DECLARATIVES IGNORED.

IEQ1094I- USE STATEMENT INVALID. IGNORED.

IEQ1095I- WORD 'SECTION' OR 'DIVISION' MISSING. ASSUMED PRESENT.

IEQ1097I- PROGRAM-ID MISSING OR MISPLACED. IF PROGRAM-ID DOES NOT IMMEDIATELY FOLLOW IDENTIFICATION DIVISION, IT WILL BE IGNORED.

IEQ1098I- ALPHA LITERAL NOT CONTINUED WITH HYPHEN AND QUOTE. END LITERAL ON LAST CARD.

IEQ1099I- ***** IS INVALID AS USED.

IEQ1100I- ***** SEQUENCE ERRORS IN SOURCE PROGRAM.

IEQ1101I- NEXT PAGE NOT IN FIRST LINE CLAUSE. IGNORED.

IEQ1102I- INCOMPLETE 02 LEVEL. ASSUME VALUE SPACES.

IEQ1103I- GROUP TYPE ALLOWED ONCE FOR RD. IGNORED.

IEQ1104I- CONTROL NAME NOT SPECIFIED IN RD. SKIPPING TO NEXT 01.

IEQ1105I- LEVEL 02 EXPECTED. ASSUMED.

IEQ11061- OPERAND FOR TERMINATE NOT FOUND OR ILLEGAL. OPERAND DROPPED.

IEQ1107I- ERROR IN PAGE CLAUSE SPECIFICATIONS. ASSUMING STANDARD VALUES.

IEQ1108I- ***** IS NOT A POSITIVE INTEGRAL NUMBER. ASSUMED ONE.

IEQ1109I- DUPLICATE USE OF CONTROL NAME. SKIPPING TO NEXT 01.

IEQ1110I- INVALID USE OF SUM CLAUSE. CONTINUING.

IEQ1111I- ELEMENTARY LEVEL WITHOUT COLUMN OR SUM CLAUSE. WARNING.

IEQ1112I- INTEGER OUTSIDE LEGAL LIMITS. CONTINUING.

IEQ1113I- EXPECTING 6-DIGIT SEQUENCE NUMBER. SKIPPING TO NEXT INSERT OR DELETE NUMBER. *****.

IEQ1114I- EXTRANEOUS COMMA OR HYPHEN ON DELETE CARD. IGNORED.

IEQ1115I- NO BLANK, COMMA OR HYPHEN FOLLOWING SEQUENCE NUMBER. ASSUME BLANK. *****.

IEQ1116I- EXPECTING 6-DIGIT SEQUENCE NUMBER AFTER HYPHEN. IGNORING DELETE FROM THRU NUMBER. *****.

IEQ1117I- DELETE NUMBER GREATER THAN LAST SEQUENCE NUMBER. STOP INSERT AND DELETE. *****.

IEQ1118I- INSERT NUMBER GREATER THAN LAST SEQUENCE NUMBER. STOP INSERT AND DELETE. *****.

IEQ1119I- ***** FEATURE NOT YET IMPLEMENTED.

IEQ1120I- COMMA NOT FOLLOWED BY BLANK. ASSUME SPACE.

IEQ1121I- PERIOD OR COMMA INVALID AS USED IN PICTURE CLAUSE.

IEQ1124I- EXTERNAL-NAME USED OUTSIDE OF RERUN CLAUSE. SENTENCE IGNORED.

IEQ1125I- NUMBER IS ZERO OR NEGATIVE. SENTENCE IGNORED.

IEQ1126I- NUMBER TOO LARGE FOR RERUN. SENTENCE IGNORED.

IEQ1127I- *** FILE-NAME USED IN PREVIOUS RERUN. USING FIRST ONE.

IEQ1128I- *** FILE-NAME PREVIOUSLY SPECIFIED IN RERUN. RERUN IGNORED.

IEQ2001I- OPEN OPTION IS ILLEGAL FOR THIS ACCESS METHOD. WARNING.

IEQ2002I- DEVICE MUST BE DIRECT-ACCESS FOR OPEN I-O. WARNING.

IEQ2003I- DEVICE MUST BE DIRECT-ACCESS FOR THIS ACCESS METHOD. WARNING.

IEQ2004I- RECORDING MODE OPTION ILLEGAL FOR THIS ACCESS METHOD. F MODE ASSUMED.

IEQ2005I- RESERVE CLAUSE ILLEGAL FOR THIS ACCESS METHOD OR NUMBER OF AREAS IS GREATER THAN 254. CLAUSE IGNORED.

IEQ2006I- BLOCK CONTAINS CLAUSE WITH RECORDING MODE 'U' ILLEGAL FOR QSAM. CLAUSE IGNORED.

IEQ2007I- LABEL RECORDS CLAUSE MISSING OR OPTION NOT 'STANDARD'. WARNING.

IEQ2008I- APPLY WRITE-ONLY ILLEGAL FOR THIS ACCESS METHOD. IGNORED.

IEQ2009I- LABEL RECORDS OPTION INCOMPATIBLE WITH DEVICE TYPE. WARNING.

IEQ2010I- RERUN CLAUSE ILLEGAL FOR THIS ACCESS METHOD. CLAUSE IGNORED.

IEQ2011I- RERUN CLAUSE ILLEGAL WITH OPEN I-O. CLAUSE IGNORED.

IEQ2013I- FILE-LIMIT CLAUSE ILLEGAL FOR THIS ACCESS METHOD. CLAUSE IGNORED.

IEQ2014I- FILE-LIMIT SPEC'D AND FILE NOT OPENED AS OUTPUT. CLAUSE IGNORED.

IEQ2015I- APPLY RESTRICTED SEARCH ILLEGAL FOR THIS ACCESS METHOD. CLAUSE IGNORED.

IEQ2019I- BLOCK CONTAINS CLAUSE ILLEGAL FOR THIS ACCESS METHOD. CLAUSE IGNORED.

IEQ2020I- TRACK-AREA CLAUSE ILLEGAL FOR THIS ACCESS METHOD. CLAUSE IGNORED.

IEQ2022I- APPLY OVERFLOW NAME SPEC'D AND DEVICE IS NOT A PRINTER. WARNING.

IEQ2023I- A ***** KEY WAS NOT SPECIFIED FOR THIS FILE. WARNING.

IEQ2024I- ***** KEY IS ILLEGAL FOR THIS ACCESS METHOD. CLAUSE IGNORED.

IEQ2025I- WRITE AFTER ADVANCING WAS SPEC'D AND FILE WAS NOT OPENED OUTPUT OR ACCESS METHOD NOT QSAM. WRITES WILL NOT HAVE AFTER ADVANCING OPTION FOR THIS FILE.

IEQ2026I- CLOSE OPTION CAN ONLY BE SPEC'D WHEN ORGANIZATION IS STANDARD SEQUENTIAL. WARNING.

IEQ2027I- TWO DIFFERENT CLOSE OPTIONS WERE SPEC'D FOR THIS FILE. WARNING.

IEQ2028I- 'SAME' AREA CLAUSE SPECIFIED FOR BSAM RELATIVE TRACK. CLAUSE IGNORED.

IEQ2029I- FIRST NON 77, 88 ITEM IN SECTION IS NOT AN 01. THIS ITEM WAS CHANGED TO 01.

IEQ2030I- 77 ITEM PRECEDED BY AN 01-49 ITEM OR 77 IN FILE SECTION. 77 CHANGED TO 01.

IEQ2031I- 88 ITEM PRECEDES 01-49, 77. 88 CHANGED TO 01.

IEQ2032I- 88 ITEM CONTAINED A CLAUSE OTHER THAN VALUE CLAUSE. CLAUSE DELETED.

IEQ2033I- ITEM'S USAGE INCOMPATIBLE WITH USAGE OF GROUP IT BELONGS TO. USAGE CHANGED TO GROUP'S USAGE.

IEQ2034I- GROUP ITEM HAS PICTURE CLAUSE. CLAUSE DELETED.

IEQ2035I- GROUP ITEM HAS BLANK WHEN ZERO CLAUSE. CLAUSE DELETED.

IEQ2036I- GROUP ITEM HAS JUSTIFIED CLAUSE. CLAUSE DELETED.

IEQ2037I- BLANK WHEN ZERO CLAUSE USED INCORRECTLY. CLAUSE IGNORED.

IEQ2038I- NUMERIC ITEM HAS BLANK WHEN ZERO CLAUSE. ITEM IS CHANGED TO REPORT.

IEQ2039I- PICTURE CONFIGURATION ILLEGAL. PICTURE CHANGED TO 9 UNLESS USAGE IS 'DISPLAY-ST', THEN L(6)BDZ9BDZ9.

IEQ2040I- JUSTIFIED CLAUSE SPEC'D FOR NON-ALPHABETIC OR NON-ALPHANUMERIC ITEM. CLAUSE DELETED.

IEQ2041I- CONDITIONAL VARIABLE IS A GROUP ITEM. CHANGED TO ELEMENTARY ITEM.

IEQ2042I- THIS ITEM CAUSES OVER 3 LEVELS OF SUBSCRIPTING. OCCURS CLAUSE DROPPED FOR THIS ITEM.

IEQ2043I- 01 LEVEL HAS AN OCCURS CLAUSE. CLAUSE DELETED.

IEQ2044I- FILLER IS USED AS A CONDITIONAL VARIABLE. CONDITION NAMES UNDER IT ARE DELETED.

IEQ2045I- REPORT CONTROL NAME UNDEFINED.

IEQ2046I- REPORT CONTROL NAME NOT FIXED LENGTH.

IEQ2049I- NO OPEN CLAUSE FOUND FOR FILE. WARNING.

IEQ2050I- BLOCK SIZE GREATER THAN 32760. WARNING - CANNOT WRITE BLOCK OF THIS SIZE.

IEQ2051I- TRACK AREA DATA-NAME IS NON-UNIQUE OR UNDEFINED. CLAUSE IGNORED.

IEQ2053I- KEY IS NON-UNIQUE OR UNDEFINED. CLAUSE IGNORED.

IEQ2054I- SYMBOLIC KEY PICTURE OR USAGE NOT LEGAL.

IEQ2055I- STERLING NON-REPORT PICTURE - SIGN IN POUND FIELD MUST BE ON HI OR LO ORDER DIGIT. PICTURE REPLACED BY 9D8D7.

IEQ2056I- STERLING NON-REPORT PICTURE - 9 IN ILLEGAL POSITION. PICTURE REPLACED BY 9D8D7.

IEQ2057I- STERLING NON-REPORT PICTURE - SIGN IN SHILLING FIELD ILLEGAL. PICTURE REPLACED BY 9D8D7.

IEQ2058I- STERLING NON-REPORT PICTURE - 8 IN ILLEGAL POSITION. PICTURE REPLACED BY 9D8D7.

IEQ2059I- STERLING NON-REPORT PICTURE - SIGN IN PENCE FIELD ILLEGAL. PICTURE REPLACED BY 9D8D7.

IEQ2060I- STERLING NON-REPORT PICTURE - 6 OR 7 IN ILLEGAL POSITION. PICTURE REPLACED BY 9D8D7.

IEQ2061I- STERLING NON-REPORT PICTURE - USAGE NOT DISPLAY-ST. PICTURE REPLACED BY 9(1).

IEQ2062I- STERLING NON-REPORT PICTURE - V IN ILLEGAL POSITION. PICTURE REPLACED BY 9D8D7.

IEQ2063I- STERLING NON-REPORT PICTURE - S IN ILLEGAL POSITION. PICTURE REPLACED BY 9D8D7.

IEQ2064I- STERLING NON-REPORT PICTURE - DIGIT LENGTH GT 18. PICTURE REPLACED BY 9D8D7.

IEQ2065I- STERLING NON-REPORT PICTURE - SHILLING FIELD GT 2. PICTURE REPLACED BY 9D8D7.

IEQ2066I- STERLING NON-REPORT PICTURE - PENCE FIELD GT 2. PICTURE REPLACED BY 9D8D7.

IEQ2067I- STERLING NON-REPORT PICTURE - NO POUND SEPARATOR. PICTURE REPLACED BY 9D8D7.

IEQ2068I- STERLING NON-REPORT PICTURE - NO SHILLING SEPARATOR. PICTURE REPLACED BY 9D8D7.

IEQ2069I- NUMERIC PICTURE - SIGN IN ILLEGAL POSITION. PICTURE REPLACED BY 9(1).

IEQ2070I- NUMERIC PICTURE - P IN ILLEGAL POSITION. PICTURE REPLACED BY 9(1).

IEQ2071I- NUMERIC PICTURE - V IN ILLEGAL POSITION. PICTURE REPLACED BY 9(1).

IEQ2072I- NUMERIC PICTURE - NO 9 IN PICTURE. PICTURE REPLACED BY 9(1).

IEQ2073I- NUMERIC PICTURE - P ENCLOSED BY 9'S. PICTURE REPLACED BY 9(1).

IEQ2074I- NUMERIC PICTURE - NO SIGN ON BINARY ITEM. ASSUMED SIGN PRESENT.

IEQ2075I- NUMERIC PICTURE - DIGIT LENGTH GT 18. PICTURE REPLACED BY 9(1).

IEQ2076I- NUMERIC PICTURE - DIGIT LENGTH + SCALE GT 18. PICTURE REPLACED BY 9(1).

IEQ2077I- EXTERNAL FLOATING-POINT PICTURE - USAGE NOT DISPLAY. PICTURE CHANGED TO 9.

IEQ2078I- EXTERNAL FLOATING-POINT PICTURE - SIGN COUNT NOT ONE. CHANGED TO 1.

IEQ2079I- EXTERNAL FLOATING-POINT PICTURE - SIGN IN ILLEGAL POSITION. PICTURE CHANGED TO +9.E+99.

IEQ2080I- EXTERNAL FLOATING-POINT PICTURE - SIGN MISSING. ASSUME MINUS SIGN.

IEQ2081I- EXTERNAL FLOATING-POINT PICTURE - REQUIRED CHARACTER BEFORE EXPONENT MISSING. PICTURE CHANGED TO +9.E+99.

IEQ2082I- EXTERNAL FLOATING-POINT PICTURE - NO DECIMAL-POINT IN MANTISSA. ASSUME IMPLIED V.

IEQ2083I- EXTERNAL FLOATING-POINT PICTURE - MANTISSA LENGTH GT 16. PICTURE CHANGED TO +9.E+99.

IEQ2084I- EXTERNAL FLOATING-POINT PICTURE - TOTAL LENGTH GT 21. PICTURE CHANGED TO +9.E+99.

IEQ2085I- EXTERNAL FLOATING-POINT PICTURE - EXPONENT LENGTH NOT 2 DIGITS. ASSUME 2 DIGITS.

IEQ2086I- REPORT PICTURE - TWO FIXED DOLLAR SIGNS, +, - OR FIXED AND FLOATING DOLLAR SIGN. PICTURE REPLACED BY 9(1).

IEQ2087I- REPORT PICTURE - SIGN PRECEDES FIXED DOLLAR SIGN. PICTURE REPLACED BY 9(1).

IEQ2088I- REPORT PICTURE - FIXED SIGN PRECEDES FLOATING STRING. PICTURE REPLACED BY 9(1).

IEQ2089I- REPORT PICTURE - 9, Z OR * PRECEDES FLOATING STRING. PICTURE REPLACED BY 9(1).

IEQ2090I- REPORT PICTURE - P IN ILLEGAL POSITION. PICTURE REPLACED BY 9(1).

IEQ2091I- REPORT PICTURE - TWO DIFFERENT FLOATING STRING CHARACTERS. PICTURE REPLACED BY 9(1).

IEQ2092I- REPORT PICTURE - Z AND * IN PICTURE. PICTURE REPLACED BY 9(1).

IEQ2093I- REPORT PICTURE - 9 PRECEDES * OR Z. PICTURE REPLACED BY 9(1).

IEQ2094I- REPORT PICTURE - FLOATING STRING PRECEDES * OR Z. PICTURE REPLACED BY 9(1).

IEQ2095I- NUMBER OF CHARACTERS IN BLOCK CONTAINS CLAUSE ON FORMAT F FILE NOT MULTIPLE OF RECORD LENGTH. ASSUMING BLOCK SIZE TO BE CLOSEST MULTIPLE.

IEQ2096I- REPORT PICTURE - TWO DECIMAL POINTS. PICTURE REPLACED BY 9(1).

IEQ2097I- REPORT PICTURE - DECIMAL POINT OR V CONTRADICTORY TO P. PICTURE REPLACED BY 9(1).

IEQ2098I- REPORT PICTURE - CR AND DB BOTH USED. PICTURE REPLACED BY 9(1).

IEQ2099I- REPORT PICTURE - CR OR DB AND SIGN BOTH USED. PICTURE REPLACED BY 9(1).

IEQ2100I- REPORT PICTURE - CR OR DB NOT LAST TWO CHARACTERS IN PICTURE. PICTURE REPLACED BY 9(1).

IEQ2101I- REPORT PICTURE - SIGN IS NOT FIRST OR LAST CHARACTER IN PICTURE. PICTURE REPLACED BY 9(1).

IEQ2102I- REPORT PICTURE - NUMERIC CHARACTERS AFTER DECIMAL POINT ARE NOT THE SAME. PICTURE REPLACED BY 9(1).

IEQ2103I- REPORT PICTURE - TOTAL LENGTH GT 127. PICTURE REPLACED BY 9(1).

IEQ2104I- REPORT PICTURE - NUMERIC LENGTH GT 18. PICTURE REPLACED BY 9(1).

IEQ2105I- REPORT PICTURE - NO NUMERIC CHARACTERS IN PICTURE. PICTURE REPLACED BY 9(1).

IEQ2106I- REPORT PICTURE - BLANK WHEN ZERO CLAUSE SPEC'D FOR ALL *'S. PICTURE REPLACED BY 9(1).

IEQ2107I- REPORT PICTURE - USAGE NOT DISPLAY. PICTURE CHANGED TO 9.

IEQ2109I- TRACK-AREA INTEGER NOT MULTIPLE OF 8. ROUNDED DOWN TO MULTIPLE OF 8.

IEQ2110I- APPLY WRITE-ONLY CLAUSE ILLEGALLY USED. IGNORED.

IEQ2111I- RECORDING MODE V NOT IMPLEMENTED FOR INDEXED FILES. SUBSTITUTING F.

IEQ2113I- ITEM WITH USAGE OF COMPUTATIONAL-1 OR COMPUTATIONAL-2 HAS PICTURE CLAUSE. CLAUSE IGNORED.

IEQ2115I- GROUP ITEM SIZE IS GT 32K. WARNING.

IEQ2116I- FIXED LENGTH GROUP ITEM IN WORKING-STORAGE SECTION IS GT 131K. WARNING.

IEQ2117I- THE OBJECT OF REDEFINES CLAUSE IS A VARIABLE LENGTH ITEM. WARNING.

IEQ2118I- LENGTH OF REDEFINES SUBJECT GREATER THAN LENGTH OF REDEFINES OBJECT. SUBJECT LENGTH USED - WARNING.

IEQ2119I- VALUE CLAUSE SPECIFIED FOR AN ITEM IN A REDEFINES GROUP. CLAUSE IGNORED.

IEQ2120I- OBJECT OF REDEFINES CLAUSE UNDEFINED OR ILLEGAL. CLAUSE IGNORED.

IEQ2121I- SUBJECT OF REDEFINES IS VARIABLE LENGTH. WARNING.

IEQ2122I- REDEFINES SUBJECT LEVEL NUMBER NOT EQUAL TO REDEFINES OBJECT LEVEL NUMBER. CLAUSE IGNORED.

IEQ2123I- OBJECT OF REDEFINES IS SUBSCRIPTED.

IEQ2124I- OBJECT OF REDEFINES IS VARIABLE LENGTH GROUP ITEM. WARNING.

IEQ2125I- VALUE CLAUSE LEGAL ONLY FOR 88 ITEMS IN FILE SECTION AND LINKAGE SECTION. CLAUSE IGNORED.

IEQ2126I- VALUE CLAUSE LITERAL TOO LONG. TRUNCATED TO PICTURE SIZE.

IEQ2127I- VALUE CLAUSE SPECIFIED FOR GROUP ITEM. CLAUSE IGNORED.

IEQ2128I- VALUE CLAUSE LITERAL DOES NOT CONFORM TO PICTURE. CHANGED TO BLANKS.

IEQ2129I- VALUE CLAUSE LITERAL DOES NOT CONFORM TO PICTURE. CHANGED TO ZERO.

IEQ2130I- ITEM CANNOT HAVE VALUE CLAUSE. CLAUSE IGNORED.

IEQ2131I- RECORD KEY UNDEFINED OR NON-UNIQUE. KEY IGNORED.

IEQ2132I- RECORD KEY LENGTH GREATER THAN 255 BYTES. USING FIRST 255 BYTES.

IEQ2133I- LABEL RECORDS CLAUSE MISSING.

IEQ2134I- VALUE FOR SCALING CHARACTER SHOULD BE ZERO. CHANGED TO ZERO.

IEQ2135I- OVERFLOW-NAME ON FILE WITH ALTERNATE AREAS SPECIFIED OR RESERVE CLAUSE OMITTED.

IEQ2136I- SYMBOLIC KEY LENGTH GREATER THAN 255 BYTES. KEY IGNORED.

IEQ2137I- ACTUAL KEY NOT S9(5) COMPUTATIONAL. KEY IGNORED.

IEQ2138I- ITEM LENGTH GREATER THAN 32K. TRUNCATED TO 32K.

IEQ2141I- QISAM FILE OPENED OUTPUT MAY NOT ALSO BE OPENED INPUT OR I-O. PROGRAM INTERRUPT WILL OCCUR.

IEQ2142I- ALPHABETIC OR ALPHANUMERIC ITEM HAS ILLEGAL USAGE. PICTURE CHANGED TO 9.

IEQ2143I- STERLING NON-REPORT PICTURE - COUNT OF V OR S NOT ONE. ASSUMED ONE.

IEQ2144I- NUMERIC PICTURE - COUNT OF V OR S NOT ONE. ASSUMED ONE.

IEQ2145I- ALPHABETIC OR ALPHANUMERIC ITEM LENGTH GREATER THAN 32767. TRUNCATED TO 32767.

IEQ2146I- RECORD SIZE IN RECORD-CONTAINS CLAUSE DISAGREES WITH COMPUTED RECORD SIZE. USING MAXIMUM COMPUTED SIZE.

IEQ2147I- 'RECORD CONTAINS INTEGER-1' IS NOT MINIMUM.

IEQ2148I- ON AN 01(77) COPY LIBRARY-NAME CLAUSE, LIBRARY DID NOT HAVE AN 01(77) AS FIRST CARD. WARNING.

IEQ2149I- VALUE CLAUSE SPECIFIED FOR ITEM WITH OCCURS OR FOR ITEM SUBORDINATE TO AN ITEM WITH OCCURS. CLAUSE IGNORED.

IEQ2150I- VALUE CLAUSE SPECIFIED FOR ITEM IN VARIABLE LENGTH PORTION OF A WORKING-STORAGE RECORD. CLAUSE IGNORED.

IEQ2151I- ELEMENTARY ITEMS NOT INTERNAL FLOATING-POINT MUST HAVE PICTURE. PICTURE ASSUMED 9.

IEQ2152I- COMPILER ERROR - PHASE 2 INPUT UNRECOGNIZABLE. SKIPPING TO NEXT PHASE.

IEQ2153I- DATA-NAME OF TRACK-AREA CLAUSE IS NOT AN ITEM OF LEVEL 01 OR 77 IN WORKING STORAGE. CLAUSE IGNORED.

IEQ2154I- THE AREA BEING REDEFINED IS NOT IMMEDIATELY PRECEDING THE ENTRY WHICH REDEFINES IT OR THE LEVEL NUMBERS OF THE SUBJECT AND OBJECT OF THE REDEFINES ARE NOT THE SAME. THE OBJECT OF THE REDEFINES IS ASSUMED TO BE THE LAST ENTRY WITH SAME LEVEL NUMBER AS SUBJECT OF REDEFINES.

IEQ2155I- ILLEGAL STERLING NON-REPORT PICTURE CHARACTER. PICTURE REPLACED BY 9D8D7.

IEQ2156I- PICTURE DOES NOT CONTAIN A SIGN. SIGN DROPPED FROM VALUE CLAUSE LITERAL.

IEQ2157I- SYMBOLIC KEY FOR A RELATIVE ORGANIZATIONAL FILE MUST BE S9(8) COMPUTATIONAL. WARNING.

IEQ2158I- OCCURS DEPENDING ON VARIABLE IS IN VARIABLE PORTION OF A RECORD. PROGRAM INTERRUPT WILL OCCUR.

IEQ2159I- A SEQUENTIAL FILE NAMED UNDER A SAME AREA CLAUSE MUST HAVE A RESERVE CLAUSE ASSOCIATED WITH IT. CLAUSE IGNORED.

IEQ2160I- 'RECORD CONTAINS INTEGER-2' IS USED FOR RECORD LENGTH SINCE THERE WERE TWO OR MORE VARIABLE LENGTH ITEMS IN THE RECORD. WARNING.

IEQ2161I- PICTURE INVALID. ADJACENT C DELIMITERS. ASSUMED PICTURE L(6)9BDZ9BDZ9.

IEQ2162I- PICTURE INVALID. ADJACENT D DELIMITERS. ASSUMED PICTURE L(6)9BDZ9BDZ9.

IEQ2163I- PICTURE INVALID. MORE THAN 2 DELIMITERS. ASSUMED PICTURE L(6)9BDZ9BDZ9.

IEQ2164I- PICTURE INVALID. NO STERLING DELIMITERS. ASSUMED PICTURE L(6)9BDZ9BDZ9.

IEQ2165I- PICTURE INVALID. ONLY 1 STERLING DELIMITER. ASSUME PICTURE L(6)9BDZ9BDZ9.

IEQ2166I- PICTURE INVALID. ERROR IN SHILLING FIELD. ASSUMED SHILLING PICTURE Z9B.

IEQ2167I- PICTURE INVALID. NUMBER OF POUND DIGITS EXCEEDS 15. ASSUMED PICTURE L(6)9BD.

IEQ2168I- PICTURE INVALID. ERROR IN WHOLE PENCE FIELD. ASSUMED PENCE PICTURE Z9.

IEQ2169I- PICTURE INVALID. ERROR IN DECIMAL PENCE FIELD. DECIMAL FIELD TRUNCATED.

IEQ2170I- PICTURE INVALID. ERROR IN POUND FIELD. ASSUMED POUND PICTURE L(6)9B.

IEQ2171I- PICTURE INVALID. NUMBER OF POUND DIGITS PLUS NUMBER OF PENCE DECIMAL EXCEEDS 15. DECIMAL PENCE DROPPED.

IEQ2172I- PICTURE INVALID. SIZE OF REPORT FIELD EXCEEDS 127 BYTES. ASSUMED PICTURE L(6)9BDZ9BDZ9.

IEQ2173I- PICTURE INVALID. CR OR DB NOT VALID WITH LEADING SIGN. DECIMAL FIELD TRUNCATED.

IEQ2174I- PICTURE INVALID. SIGN IN DECIMAL PENCE FIELD NOT VALID WITH LEADING SIGN. DECIMAL FIELD TRUNCATED.

IEQ2175I- TRACK-AREA EXCEEDS AND IS REDUCED TO 32,760 BYTES.

IEQ2176I- DATA-NAME OF TRACK-AREA CLAUSE EXCEEDS 32,767 BYTES IN LENGTH. CLAUSE IGNORED.

IEQ2177I- DATA-NAME OF TRACK-AREA CLAUSE IS NOT FIXED-LENGTH. CLAUSE IGNORED.

IEQ2178I- RECORD-KEY IS NOT WITHIN FILE-RECORD.

IEQ2179I- RECORD-KEY IS NOT FIXED-LENGTH.

IEQ2180I- RECORD-KEY FOR UNBLOCKED FILE INCLUDES FIRST BYTE OF RECORD.

IEQ2181I- SYMBOLIC OR ACTUAL KEY IS DEFINED WITHIN THE FILE.

IEQ2182I- APPLY WRITE ONLY IS MEANINGLESS WHEN RECORDING MODE IS F. CLAUSE IGNORED.

IEQ2183I- NO LEVEL 01 FOR FD.

IEQ2184I- VALUE CLAUSE LITERAL DOES NOT CONFORM TO PICTURE. CLAUSE IGNORED.

IEQ2186I- PICTURE DUPLICATION FACTOR IS ZERO. ASSUMING ONE OCCURRENCE OF PICTURE CHARACTER.

IEQ2188I- BLOCK CONTAINS 0 RECORDS. BLOCK-SIZE MUST BE SPECIFIED IN DD CARD.

IEQ2189I- BLOCK CONTAINS 0 RECORDS INCOMPATIBLE WITH SAME AREA CLAUSE OR FOR BISAM. CLAUSE IGNORED.

IEQ3001I- ***** NOT DEFINED.

Explanation: This message always appears in conjunction with another message.

IEQ3002I- ***** NOT UNIQUE.

Explanation: This message always appears in conjunction with another message.

IEQ3003I- HIGHEST LEVEL QUALIFIER ***** NOT DEFINED.

Explanation: This message always appears in conjunction with another message.

IEQ3004I- HIGHEST LEVEL QUALIFIER ***** NOT UNIQUE.

Explanation: This message always appears in conjunction with another message.

IEQ3005I- ***** NOT A VALID QUALIFIER.

Explanation: This message always appears in conjunction with another message.

IEQ3006I- ***** NOT DEFINED AS PART OF *****.

Explanation: This message always appears in conjunction with another message.

IEQ3007I- ***** NOT UNIQUELY QUALIFIED BY *****.

Explanation: This message always appears in conjunction with another message.

IEQ3008I- ***** NOT VALID AS DATA-NAME-1 IN ***** CORRESPONDING STATEMENT.

IEQ3009I- ***** NOT VALID AS DATA-NAME-2 IN ***** CORRESPONDING STATEMENT.

IEQ3010I- SUPERFLUOUS 'TO' IGNORED IN ***** CORRESPONDING STATEMENT.

IEQ3011I- NO CORRESPONDENCE FOUND BETWEEN DATA-NAME-1 AND *****.

IEQ3012I- NO MATCHED DCB IN QFILE TABLE FOR FILE *****.

IEQ3013I- DICT PTR LESS THAN QVAR ENTRY FOR ELEMENTARY ITEM.

IEQ3014I- NO MATCH FOUND IN QVAR FOR ***** ELEMENTARY ITEM.

IEQ3015I- Q ON, DELIMITER GROUP NAME ***** NOT FOUND.

IEQ3016I- IMPOSSIBLE *****. COMPILER ERROR.

IEQ3017I- COMPILER ERROR. ***** MINOR CODE ILLEGAL.

IEQ3019I- ILLEGAL LEVEL FOR *****.

IEQ3020I- REPORT NAME ILLEGAL AS USED. DISCARDED.

IEQ4001I- OUTCOME OF A PRECEDING CONDITION LEADS TO NON-EXISTENT 'NEXT SENTENCE'. 'STOP RUN' INSERTED.

IEQ4002I- ***** STATEMENT INCOMPLETE. STATEMENT DISCARDED.

IEQ4003I- EXPECTING NEW STATEMENT. FOUND *****. DELETING TILL NEXT VERB OR PROCEDURE NAME.

IEQ4004I- *****/***** IS ILLEGALLY USED IN ***** STATEMENT. DISCARDED.

IEQ4005I- ***** AND ***** VIOLATE RULE ABOUT LENGTH OF TRANSFORM OPERANDS. STATEMENT DISCARDED.

IEQ4006I- ***** STATEMENT CONTAINS UNPAIRED LEFT PARENTHESES. OUTERMOST IGNORED.

IEQ4007I- ***** MISSING OR MISPLACED IN ***** STATEMENT. ASSUMED IN REQUIRED POSITION.

IEQ4008I- SUPERFLUOUS ***** FOUND IN ***** STATEMENT. IGNORED.

IEQ4009I- EXAMINE STATEMENT REQUIRES FIGURATIVE CONSTANT OR SINGLE ALPHANUMERIC CHARACTER. FOUND *****. STATEMENT DISCARDED.

IEQ4010I- ***** STATEMENT CONTAINS UNPAIRED RIGHT PARENTHESES. OUTERMOST IGNORED.

IEQ4011I- ***** IS NOT AN ALLOWABLE CHARACTER FOR *****. STATEMENT DISCARDED.

IEQ4012I- COMPARISON BETWEEN TWO LITERALS IS ILLEGAL. TEST DISCARDED.

IEQ4013I- RELATIONAL MISSING IN IF STATEMENT. 'EQUAL' ASSUMED.

IEQ4014I- EXAMINE STATEMENT REQUIRES DATA-NAME WHOSE USAGE IS DISPLAY. FOUND *****/*****. STATEMENT DISCARDED.

IEQ4015I- 'GO TO .' IS ILLEGAL UNLESS ALTERED SOMEWHERE. STATEMENT DISCARDED.

IEQ4016I- OPERAND OF ***** APPEARS IN WRONG SEGMENT OF PROGRAM. STATEMENT ACCEPTED AS WRITTEN.

IEQ4017I- ELSE UNMATCHED BY CONDITION IS DISCARDED.

IEQ4018I- 'ALL' MUST BE FOLLOWED BY SINGLE ALPHANUMERIC CHARACTER. FOUND *****. 'ALL' DISCARDED.

IEQ4019I- *****/***** MAY NOT BE USED AS ARITHMETIC OPERAND IN *****

STATEMENT. ARBITRARILY SUBSTITUTING *****.

IEQ4020I- SIGN BEFORE ***** IS DISCARDED.

IEQ4021I- MINUS SIGN FOLLOWED BY SPACE ACCEPTED AS REVERSING SIGN OF FOLLOWING LITERAL.

IEQ4022I- EXIT MUST BE SINGLE-WORD PARAGRAPH PRECEDED BY A PROCEDURE-NAME. STATEMENT DISCARDED.

IEQ4023I- SYNTAX REQUIRES STORE-FIELD TO BE A REPORT ITEM OR NUMERIC DATA-NAME. FOUND *****/*****. STATEMENT DISCARDED.

IEQ4024I- TWO OPERANDS ARE REQUIRED BEFORE 'GIVING'. STATEMENT DISCARDED.

IEQ4026I- *****/***** IS ILLEGALLY USED IN ***** TEST. TEST DISCARDED.

IEQ4027I- RIGHT TERM OF A CONDITION MAY NOT BE NEGATED. NEGATION IS APPLIED TO THE RELATIONAL.

IEQ4028I- TWO 'NOT'S' IN SUCCESSION ILLEGAL. ACCEPTED AS CANCELLING EACH OTHER.

IEQ4029I- *****/***** MAY NOT BE COMPARED WITH *****/*****. TEST DISCARDED.

IEQ4030I- SYNTAX REQUIRES 'OR', 'AND', OR VERB AFTER CONDITION. FOUND *****. DELETING TILL ONE OF THESE IS FOUND.

IEQ4031I- PROCEDURE-NAME NOT THAT OF A SINGLE GO PARAGRAPH MAY NOT BE ALTERED. STATEMENT DISCARDED.

IEQ4032I- NO ACTION INDICATED IF PRECEDING CONDITION IS TRUE. STATEMENT ACCEPTED WITH TRUE AND FALSE OUTCOMES IDENTICAL.

IEQ4033I- PROCEDURE-NAME WHICH IS THE END-OF-RANGE OF A PERFORM STATEMENT MAY NOT BE ALTERED. STATEMENT DISCARDED.

IEQ4034I- GO DEPENDING ON MUST BE FOLLOWED BY INTEGRAL DATA-NAME LESS THAN 4 DIGITS IN LENGTH. FOUND *****. STATEMENT DISCARDED.

IEQ4035I- COBOL LANGUAGE FORBIDS VARYING MORE THAN 3 DATA-NAMES. STATEMENT ACCEPTED AS WRITTEN.

Explanation: This compiler can normally handle a program varying more than three data-names, but the practice is invalid under standard COBOL language rules and is not recommended.

IEQ4036I- PERFORM RANGE IS FROM ***** TO ***, WHICH PRECEDES IT. STATEMENT ACCEPTED AS WRITTEN.

Explanation: This compiler can normally handle the perform range indicated, but the practice is not recommended.

IEQ4037I- SYNTAX REQUIRES PROCEDURE-NAME TO FOLLOW 'THRU'. FOUND *****. ***** OPTION DISCARDED.

IEQ4038I- VARYING OPTION REQUIRES NUMERIC DATA-NAME. FOUND LITERAL. ARBITRARILY SUBSTITUTING *****

IEQ4039I- *****/***** IN VARYING OPTION IS NOT NUMERIC. ARBITRARILY SUBSTITUTING *****.

IEQ4040I- ***** FILE ***** MAY NOT BE OPENED ***** AND IS DISCARDED.

IEQ4041I- SYNTAX REQUIRES 'INPUT', 'OUTPUT', OR 'I-O' AFTER OPEN. FOUND *****. DELETING TILL ONE OF THESE IS FOUND.

IEQ4042I- SYNTAX REQUIRES FILE-NAME IN ***** STATEMENT. FOUND *****. DELETING TILL LEGAL ELEMENT FOUND.

IEQ4043I- ***** IS AN ILLEGAL OPERAND FOR ***** IN THIS PORTION OF PROGRAM. DISCARDED.

IEQ4044I- INVALID KEY OPTION ILLEGAL FOR *****, FOR WHICH AN ERROR DECLARATIVE EXISTS. DISCARDED.

IEQ4045I- 'AFTER ADVANCING' REQUIRES 1-CHARACTER ALPHANUMERIC DATA-NAME OR INTEGER LESS THAN 3. FOUND *****. SUBSTITUTING *****.

IEQ4046I- ILLEGAL TO *****/***** FILE *****. STATEMENT DISCARDED.

IEQ4047I- ***** STATEMENT MAY NOT APPEAR IN THIS DECLARATIVE SECTION. STATEMENT DISCARDED.

IEQ4048I- USE VERB MAY NOT APPEAR EXCEPT IN DECLARATIVES SECTION. STATEMENT DISCARDED.

IEQ4049I- INAPPROPRIATE OPTIONAL COBOL WORDS PRECEDING ***** IGNORED.

IEQ4050I- SYNTAX REQUIRES *****. FOUND *****. STATEMENT DISCARDED.

IEQ4052I- *****/***** MAY NOT BE TARGET FIELD FOR *****/***** IN ***** STATEMENT, AND IS DISCARDED.

IEQ4054I- SYNTAX REQUIRES SORT-FILE NAME. FOUND *****. STATEMENT DISCARDED.

IEQ4055I- SORT SEQUENCE NOT SPECIFIED. ASCENDING ASSUMED.

IEQ4056I- SYNTAX REQUIRES *****. FOUND *****. DISCARDED.

IEQ4057I- SORT-KEYS LIMITED TO 12 OR TO 256 BYTES. ***** DISCARDED.

IEQ4058I- SYNTAX REQUIRES 'USING' ('GIVING') TO BE FOLLOWED BY FILE-NAME DEFINED UNDER AN FD. FOUND *****. STATEMENT DISCARDED.

IEQ4059I- SORT-KEY MUST BE NON-SUBSCRIPTED FIXED-LENGTH DATA-NAME DEFINED UNDER AN SD. FOUND *****. DISCARDED.

IEQ4060I- ***** IS NOT A POSITIVE NUMERIC INTEGRAL LITERAL OF REQUIRED LENGTH. ***** OPTION DISCARDED.

IEQ4061I- NEITHER NAMED NOR CHANGED SPECIFIED. STATEMENT ACCEPTED. WILL BE TREATED AS FORMATTED DISPLAY.

IEQ4062I- 'NAMED CHANGED' ACCEPTED AS 'CHANGED NAMED'.

IEQ4063I- PREVIOUS DEBUG PACKET REFERS TO SAME PROCEDURE-NAME. CARD DELETED AND FOLLOWING STATEMENTS ATTACHED TO IMMEDIATELY PRECEDING PACKET.

IEQ4064I- ***** IS NOT A POSITIVE NUMERIC INTEGRAL LITERAL OF REQUIRED LENGTH. SUBSTITUTING *****.

IEQ4065I- ENTER LINKAGE DOES NOT PRECEDE ***** STATEMENT. ASSUMED.

IEQ4066I- SYNTAX REQUIRES 01 LEVEL SD DATA-NAME IN RELEASE STATEMENT. FOUND *****. STATEMENT DISCARDED.

IEQ4067I- SYNTAX REQUIRES 'COBOL' OR 'LINKAGE' AFTER ENTER VERB. OMISSION DOES NOT AFFECT PROCESSING.

IEQ4068I- COMPILER ERROR. PHASE 4 TRYING TO GET DATA ATTRIBUTES FOR *****.

IEQ4069I- SYNTAX REQUIRES DEVICE-NAME. FOUND ***** IN ***** STATEMENT. SYSTEM UNIT ASSUMED.

IEQ4070I- AT-END CLAUSE MISSING. AT-END-NEXT-VERB ASSUMED.

IEQ4071I- ***** EXCEEDS LEGAL LENGTH. DISCARDED.

IEQ4072I- EXIT FROM ***** PROCEDURE ASSUMED BEFORE *****.

IEQ4073I- ***** SHOULD NOT APPEAR IN DECLARATIVE SECTION. STATEMENT ACCEPTED AS WRITTEN.

Explanation: The statement will be compiled, but its use is illegal under standard COBOL rules and is not recommended.

IEQ4074I- CONDITION-NAME INAPPROPRIATELY USED. SUBSTITUTING *****.

IEQ4075I- 'NEXT SENTENCE' ILLEGAL AND DISCARDED. BOTH ***** AND NOT ***** WILL CAUSE EXECUTION OF NEXT VERB.

IEQ4076I- ***** REQUIRES ***** SUBSCRIPT(S). SUBSTITUTING FIRST OCCURRENCE OF *****.

IEQ4077I- ***** MAY NOT BE USED AS A SUBSCRIPT SINCE IT REQUIRES SUBSCRIPTING ITSELF. SUBSTITUTING FIRST OCCURRENCE OF *****.

IEQ4078I- SUBSCRIPT MUST BE INTEGRAL DATA-NAME OR LITERAL. FOUND NON-INTEGER *****. SUBSTITUTING FIRST OCCURRENCE OF *****.

IEQ4079I- ***** FOUND AMONG SUBSCRIPTS. SUBSTITUTING FIRST OCCURRENCE OF *****.

IEQ4080I- *DEBUG CARD MAY NOT REFER TO A PROCEDURE NAME WHICH ITSELF IS IN A DEBUG PACKET. CARD DELETED AND FOLLOWING STATEMENTS ATTACHED TO IMMEDIATELY PRECEDING PACKET.

IEQ4081I- ***** EXCEEDS ***** CHARACTERS. UP TO 255 ACCEPTED.

IEQ4082I- ***** IS NOT DEFINED AS SUBSCRIPTED. SUBSCRIPTS DISCARDED.

IEQ4083I- OCCURS-DEPENDING-ON VARIABLE MUST BE INTEGRAL NON-SUBSCRIPTED DATA-NAME. FOUND *****. ARBITRARILY SUBSTITUTING *****.

IEQ4084I- ILLOGICAL USE OF PARENTHESES ACCEPTED WITH DOUBTS AS TO MEANING.

IEQ4085I- RECORD DESCRIPTION FOR FILE ***** MISSING OR ILLEGAL. STATEMENT DISCARDED.

IEQ4086I- ***** CONDITION USED WHERE ONLY IMPERATIVE STATEMENTS ARE LEGAL MAY CAUSE ERRORS IN PROCESSING.

IEQ4087I- 'END DECLARATIVES' MISSING OR MISPLACED. PROGRAM CANNOT BE EXECUTED.

IEQ4088I- COMPILER ERROR. I-C TEXT COUNT FIELD 0. SKIPPING TO PHASE 5.

IEQ4089I- *****/***** SHOULD NOT BE TARGET FIELD FOR *****/***** IN ***** STATEMENT. STATEMENT ACCEPTED AS WRITTEN.

IEQ4090I- SORT-KEY MUST BE IN FIXED POSITION NOT MORE THAN 4092 BYTES FROM START OF RECORD. ***** DISCARDED.

IEQ4091I- SYNTAX REQUIRES OPERAND. FOUND *****. TEST DISCARDED.

IEQ4092I- EXTERNAL DECIMAL NAME USED IN TRANSFORM STATEMENT. STATEMENT ACCEPTED AS WRITTEN.

IEQ4093I- ALL WRITE STATEMENTS FOR ***** SHOULD USE 'AFTER ADVANCING' OPTION. STATEMENT ACCEPTED AS WRITTEN.

IEQ4094I- ***** IS IN A RECORD OF AN APPLY-WRITE-ONLY FILE, AND REFERRING TO IT MAY CAUSE ERRORS IF FILE IS OPENED AS OUTPUT WHEN ***** STATEMENT IS EXECUTED.

IEQ4095I- WRITE FROM DATA-NAME REQUIRED FOR *****, TO WHICH WRITE-ONLY IS APPLIED. STATEMENT DISCARDED.

IEQ4096I- ***** STATEMENT WILL NEVER BE EXECUTED.

Explanation: The logic of the COBOL source program prevents the computer from executing the statement noted. The compiler, however, accepts the statement as written.

IEQ4097I- UNIT(REEL) OPTION ILLEGAL FOR *****. DISCARDED.

IEQ4098I- LABEL DECLARATIVE ILLEGAL FOR *****, NON-QSAM FILE.

IEQ4099I- NO EXIT SPECIFIED BEFORE END OF THIS DECLARATIVE SECTION. CONTROL WILL FALL THROUGH TO NEXT SECTION.

Explanation: Falling through will occur only under a specific set of circumstances. At other times the proper exit from the declarative procedure will be assumed. Check the generated code to determine action for execution.

IEQ5001I- COMPILER LOGIC ERROR OR MACHINE ERROR. AN UNRECOVERABLE ERROR OCCURRED WHILE TRYING TO ASSIGN A DOUBLE REGISTER. COMPILATION ABANDONED.

IEQ5002I- COMPILER LOGIC ERROR OR MACHINE ERROR. AN UNRECOVERABLE ERROR OCCURRED WHILE PROCESSING A SUBSCRIPTED DATA-NAME. COMPILATION ABANDONED.

IEQ5003I- A DIVISOR IS A ZERO CONSTANT. RESULT OF DIVIDE WILL BE SET TO ALL 9'S.

IEQ5004I- MORE THAN 255 SUBSCRIPT ADDRESS CELLS USED. PROGRAM CANNOT EXECUTE CORRECTLY.

User Response: From the map, determine the point at which the subscript address cell number (SBS) exceeded 255. Insert a dummy paragraph-name before the offending statement in the source program and recompile.

IEQ5005I- COMPILER LOGIC ERROR OR MACHINE ERROR. AN UNRECOVERABLE ERROR OCCURRED WHILE PROCESSING A MOVE. COMPILATION ABANDONED.

IEQ5006I- COMPILER LOGIC ERROR OR MACHINE ERROR. UNEXPECTED INPUT TO THE MOVE OR STORE PROCESSOR. COMPILATION ABANDONED.

IEQ5007I- COMPILER LOGIC ERROR OR MACHINE ERROR. UNEXPECTED INPUT TO THE ARITHMETIC CODE GENERATOR. COMPILATION ABANDONED.

IEQ5008I- COMPILER LOGIC ERROR OR MACHINE ERROR. UNEXPECTED INPUT TO THE FLOATING-POINT ARITHMETIC ROUTINE 'FPCVBH'. COMPILATION ABANDONED.

IEQ5009I- COMPILER LOGIC ERROR OR MACHINE ERROR. LOST SUBSCRIPT-ID IN TABLE 'XSSNT'. COMPILATION ABANDONED.

IEQ5010I- A CONSTANT INTERMEDIATE RESULT HAD TO HAVE ITS HIGH ORDER DIGIT POSITION TRUNCATED.

IEQ5011I- AN INTERMEDIATE RESULT OR A SENDING FIELD MIGHT HAVE ITS HIGH ORDER DIGIT POSITION TRUNCATED.

IEQ5012I- COMPILER LOGIC ERROR OR MACHINE ERROR. LOST INTERMEDIATE RESULT ATTRIBUTES IN 'XINTR' TABLE. COMPILATION ABANDONED.

IEQ5013I- ILLEGAL COMPARISON OF TWO NUMERIC LITERALS. STATEMENT DISCARDED.

OBJECT TIME MESSAGES

IEQ000A- xxx

Explanation: This message is issued by an object program that was originally written in COBOL F language. The message text is supplied by the object program and may indicate alternative action to be taken.

System Action: The object program enters wait state.

Operator Response: Follow the instructions given by the programmer when he submitted the program for execution. If the job step is to be resumed, enter REPLY xx, 'y', where y is any single character.

xx IEQ990D 'AWAITING REPLY'

Explanation: This message is issued by the object program when operator intervention is required.

Operator Response: Issue a REPLY command. (The contents of the text field should be supplied by the programmer on the job request form.)

IEQ999I- NO DD CARD FOR DDNAME.

Explanation: The operating system could not find a corresponding ddname on a DD statement for a file assigned in a SELECT clause.

Operator Response: User should have indicated either to continue processing or to cancel the job. Note that if the user elects to continue processing, any READ or WRITE encountered for the file will result in an ABEND.

*, in job control language 28,30,74,75
  restrictions with UNIT parameter 32
/* Statement 13,40
  under MFT and MVT 84

A, as a device class 13,17,39
ABDUMP (see dump)
abnormal termination 106,110,157,158
  incomplete 117
  requesting a dump 48
  size errors causing 193
abnormal termination dump
  definition 110
  example of 114-117
  using 112
ABSTR 33
  in QISAM 61
ACCEPT
  Statement 80
    relationship to SYSIN DD Statement 48
  subroutine 177
ACCESS clause 49,50
  in file processing techniques 182-187
Accounting information
  in EXEC Statement 18
  in JOB Statement 16
ACCT parameter 19,27
ACTUAL KEY clause
  in BDAM 59
  in BSAM 54,57
  causing errors 112
  in file processing techniques 183
address constant table 193
AFF parameter 28,31
Allocation messages 99,103-105
  compiler 97
  sample 172
ALX 33
argument list 147
arguments
  compared to parameters 146
arithmetic subroutines 177,179
assembler language 145-149
  sample 168-171
  using EXEC Statement 21
ASSIGN TO clause
  in BDAM 59
  in BSAM 57
  in QSAM 52
  in Sort Feature 131
  relationship to DD Statement 48,155
ATTACH macro instruction 191
automatic call library 46,138,139

B, as a device class 13,39
base and displacement 100
BASIS option in job control language 22,27
BASIS Statement 20,78,139-42
  in a debug packet 110
  error messages involving 199
BCD 159
BDAM
  data control block 187
  DD Statement parameters 58,59
  defining a data set in 50
  definition of 9
  error processing for 156-158,196
  permissible COBOL Clauses 59
  restrictions with job control language 58
beginning address of word 34
BFALN 182,184,185
BFTEK 182,185
binary (see also COMPUTATIONAL)
  data items 122
  intermediate results 127
  subroutines 178,179
BISAM (see also QISAM) 58-67
  considerations when using 66,67
  data control block 186
  defining a data set in 50
  definition of 9
  error conditions 197,198
  error processing for 156-158
  processing with 66,67
BLKSIZE 41,47,53,189
  for file processing techniques 182-185
  in QISAM 62
  in Sort feature 132,133
BLOCK CONTAINS clause 32
  in file processing techniques 182,184,185
  in QSAM 52
block length (see BLKSIZE)
block size
  causing errors 112
  for utility data sets 189,190
blocking words 47,50
BSAM 53-57
  data control block 183
  DD Statement parameters 57
  defining a data set in 50
  definition of 9
  error processing for 156-158,195
  permissible COBOL clauses 57
  uses of DUMMY parameter 30
  WRITE and CLOSE subroutine 179
BUF option 21,27,190
BUFCB subparameter 182,184,185
buffers (see also BUFCB, BUFNO) 182-185
  allocating space to 189,190
  determining number of 61
  specifying number of 53
BUFNO subparameter 53,61,182-185

CALL Statement 145
catalog, system 9
cataloged data sets 63,75
  creating 71
  retrieving 73,74

Cataloged procedures
bypassing steps within 19
calling 31,81,82
COBFC 84,85
COBFCLG 84,85
COBFLG 84,85
data sets produced by 81,82
definition of 13
limiting execution time of 23
modifying 86-89
programmer-written 83
restrictions 83
relationship to SYS1.PROCLIB 81
required device class names for 32
restarting programs within 20
system 82,83
using the DD Statement 87-89
using the EXEC Statement 86,87
CATLG subparameter 39,75
character delimiters
as text 15
in job control language 15
checkid 17,93
checkpoint data set 16
Checkpoint/Restart 91-95
methods 91
parameters in JOB Statement 16
parameters in EXEC Statement 20
RD parameter 16,20,93
RESTART parameter 17,94
subroutine 180
SYSCHK DD Statement 94
Checkpoints
considerations 93
how taken 16
CHKPT macro instruction 95
CLASS parameter 17,26
Class Test subroutine 180
classname, as a SYSOUT subparameter 39
CLIST option 21,27,101,189
CLOSE Statement 128
under MFT and MVT 162
CLOSE REEL Statement 54
COBFC 81,83-85
COBFCLG 83-85
COBFLG 83-85
COBOL
sample program 165-175
COBOL Subroutine library 138
Comments field in job control language 15
Compare subroutine 179
compilation
data sets for 42
example of job control statements 77,78
job step 10
sample 165-171
using the REGION parameter 161
Compiler
calling 191,192
capacity 193,194
data sets 41,42
definition of 10
diagnostic messages
summary of 199-213
internal name 99
machine requirements 161
optimization 189,190
options 27
output 97-102
specifying in EXEC Statement 18
Compiler options
in EXEC statement 18,21,22
completion code 117
in Sort programs 136
COMPUTATIONAL
conversions involving 123-125
efficient use of 119
COMPUTATIONAL-1 data items 124-126
COMPUTATIONAL-2 data items 124-126
COMPUTATIONAL-3
conversions involving 123
COND parameter 26,27,102
in cataloged procedures 86
in EXEC Statement 19
in JOB Statement 16
testing return codes 199
conditional, as a severity level (C) 101,199
Conditions terminating execution
in JOB Statement 16
CONTIG subparameter 33
in BSAM 55
in QISAM 61
Continuation of Job Control Statements, rules 15,16
control section 104,172
length 104
relative location 104
conversion subroutines 177
copy library (see also source program library) 139
COBOL sequence numbers 141
entering source statements 139
IEBUPDTE sequence numbers 141
retrieving source statements 140
updating source statements 139,140
COPY option in job control language 22,27
used in cataloged procedures, example 88
COPY Statement 20,22,78,139-140
error messages 199,201
cross reference list 103,104
used in dumps 113-117
cross reference table 172
CYL 33,34,61
CYLOFL 184

data alignment 123-126
use of PICTURE 122,127
data control block (see also DCB parameter)
fields 181-187
data conversion 123-125
Data Division
programming techniques 119-126
used in Report Writer 129
data formats 123-126
DATA parameter
in DD Statement 28,30
restrictions with UNIT parameter 32
specifying input with 74
data set
adding records to (also see MOD subparameter) 38
blocked 47,48,189

cataloged 39,63,75
cataloged procedures producing 81,82
checkpoint 16
COBOL clauses used to specify 48
compiler 41,42
concatenated 90
definition of 9
deletion of 38
disposition of 38,39
after abnormal termination 117
errors 111
generation 75,76
header label group 159
labels 158-160
linkage editor 45,46
names 31
restrictions 76
organization of 9,10,58-67
(see also the individual access methods)
partitioned 10
retaining 38
retrieving 73,74
scratching 118
under MFT and MVT 118
sharing 38
for Sort programs 132-134
specifying space for 33-35
temporary 31,35,71
data set control block 160
data set label
relationship to DD Statement 156
data set member 10
DATE-COMPILED clause 99
DCB macro instruction 181
DCB parameter (see also data control block) 155,28,49,189
BISAM, used in 64
BSAM, used in 56
description of 31
checkpoint/restart uses of 92
creating data sets with 69,70
error processing with 156,196,198
identifying information in 155
QISAM, used in 61,62,64
QSAM, used in 51-53
retrieving data sets with 73,74
Sort feature uses of 133
DD Statement 9,24-40,69-76
access methods using
BDAM 59
BSAM 57
QISAM 60-63
QSAM 52
checkpoint/restart uses of 91,92
changing libraries with 143
creating libraries with 137
error recovery option 156,157
format of 28,29
relationship to SELECT Statement 156
Sort feature, used in 131-135
ddname
in a cataloged procedure 30
used to allocate space 34
DDNAME parameter 28,182-187
cataloged procedures, used in 89,90
description of 30
error message 213
debug packet 109
debugging language 107-110
(see also TRACE Statement and EXHIBIT Statement)
DECB
error conditions 196,197
error processing with 156
linking with 145
Decimal-point alignment 122
DECK option 22,27,77,102
in cataloged procedures, example 86
Declaratives
USE AFTER STANDARD ERROR 156-158
DEFER parameter 32
DELETE Statement 72,141,142
DELETE subparameter 38
used for cataloged data sets 75
Delimiter job control statement 13,40
DEN subparameter 51
DEVD 182,183
device allocation 99
device class 9
blocking restrictions 32
names 32
diagnostic messages 97,99,101
with ON Statement 107
summary of 199-213
direct-access
data set
beginning address 34
specifying directory for 34
specifying space for 33-35
device
for compiler 161
labels 158-160
Sort feature programs 131,163
storage capacity 55
track capacity 56
volume
characteristics of 35,36
disaster, as a severity level (D) 102,199
DISP parameter 29
data set uses
cataloging 75
creating 69,70
retrieving 73,74
default values of 39
description of 38
in JOBLIB DD Statement 40
in Sort feature 133
DISPLAY Statement 80,106
conversions involving 123-125
relationship to DD Statement 47
DISPLAY subroutine 177
disposition messages 102-105
DMAP option 21,27,99
DSCB (see data set control block)
DSNAME parameter 28
in creating data sets 69,70
description of 30
in QISAM 60,61
in retrieving data sets 69,70
in Sort feature 133
DSORG 182-187
values of
DA 56
IS 61
DUMMY parameter 28,30

dummy records 54
in QISAM 64
dumps
determining location of error 112
errors causing 110-112
requesting 48,106
using SYSABEND DD Statement 110
types of
abnormal termination 110
indicative 110

EBCDIC 51,159,160
ENTER Statement 145
entry name 104
ENTRY Statement 145
EODAD 182,183,185
EROPT subparameter 52,53,110,182,195
error
as a severity level (E) 102,199
conditions
for BDAM 196
for BISAM/QISAM 196-198
for BSAM/QSAM 195
input/output 195-198
messages
return codes 20
severity codes 20
options 182-185
processing 156-158
recovery
DD Statement option 156,157
system 156
table 193
ESD (see external symbol dictionary (ESD))
ESETL macro instruction 65
EXEC Statement 9,18-24
to call cataloged procedures 81
in checkpoint/restart 93
for compiler optimization 189
format of 27
Execution
definition of 10
job step
example of job control language 79,80
output 105,106
sample 175
storage allocation 162
with the REGION parameter 161,162
EXHIBIT Statement 80,107-110
relationship to SYSOUT DD Statement 48
EXHIBIT subroutine 177
EXPDT subparameter 38
external decimal subroutines 178
external floating point subroutines 178
external name
defined 148
external reference
defined 148
external symbol dictionary (ESD) 102

FB
for device types 189
in Sort feature 132
FD 49,100
relationship to DCB 155
with WRITE AFTER ADVANCING 50
file
distinction with data set 48
file-name 48,49
File processing techniques 49-67
(see also BISAM, BDAM, BSAM, QSAM, QISAM)
FILE-LIMIT Clause 53
in BDAM 58
FLAGE option 22,27
FLAGW option 22,27
floating point
error messages 202
floating-point data items (see also COMPUTATIONAL-1 and COMPUTATIONAL-2)
intermediate results 127

generation data set 75,76
GIVING option 131
global table 100
MAP option 21
glossary 98,99,101
requesting through EXEC Statement 21
GO TO Statement
causing errors 111
used in a debug packet 110

IEBUPDTE 82,83,139-142
IEHLIST 118
IEHMOVE 137
IEHPROGM 118
IEQ 101,199
IEQCBL00 18
IEWL 18
IHDFACPT 177
IHDFDISP 177
IHDFSORT 136
INCLUDE linkage editor statement 142,150,151
INCLUDE Statement 22,78,139,140
error messages 199
index
in QISAM 58
master 64
INDEX
in QISAM 61,63
indexed sequential access methods (see QISAM)
indicative dump 110
restriction for MVT 112
information messages 103,104
input/output subroutines 177
input stream
specifying control statements for 30
reading more than one 11
INSERT Statement 141,142
instruction address
causing interrupt 116,117
intermediate results 126-128
internal decimal subroutines 178,179
internal floating point subroutines 178,179
interrupt address, example 116
invalid data
causing abnormal termination 111
INVALID KEY clause 156-158
in file processing techniques 184
invalid key error conditions 196-198

job, definition 9
job control language
  character delimiters 15
  coding 14
  examples of
    compilation 97
    linkage editing 103
  fields of
    comments field 15
    name field 14
    operation field 15
    operand field 15
  notation used in 26
  statement continuation rules 15
  types of
    DD statement 24-40
    Delimiter Statement 40
    EXEC Statement 18-24
    JOB Statement 16-18
job control procedure 13,14
job Control Statements
  format of 14-40
  listing of 14
job library 142
job management 14
job scheduler 14
JOB Statement 9,16-18
  format of 26
job step
  bypassing
    using JOB Statement 16
    using EXEC Statement 19
  definition of 9
  deferring 17
JOBLIB DD Statement 40,79,80,142,153
  restriction with cataloged procedure 82
  restriction with DDNAME parameter 89
jobname 16

KEEP subparameter 38
  used with checkpoint/restart 92
KEY clauses 54
  (see also SYMBOLIC KEY, ACTUAL KEY)
key length 184,187
KEYLEN 184,187
keyword parameters 15

label
  nonstandard 159
  volume 158-160
LABEL parameter 29,37,159
  in creating data sets 70,71
  in retrieving data sets 73,74
  in Sort feature 133
library 10,137-143
  changing 143
  compilation, use of 42,43
  concatenating 40,142
  creating 72
  directory 137
  kinds of
    automatic call 46,138,139
    copy 139
    job 142
    link 137
    private 40
    procedure 138
    source program 139
    system-provided 78,79,137,138
    sort 138
    user-defined 79,139
  linkage editing, use of 46
  members of
    adding 72
    replacing 72
    retrieving 74
  relationship to JOBLIB DD Statement 40,46
  relationship to SYSLIB DD Statement 42,43
  subroutines 46,177-180
    arithmetic 177
    COBOL 138
    conversion 177-179
    input/output 177
    intermediate results 127
LIBRARY linkage editor statement 139,142,151
LIMCT 187
LINE-COUNTER 130
LINECNT option 22,27,97
LINK macro instruction 153,191
linkage
  for error processing handling 156,157
linkage conventions 145,147
linkage editor
  calling compiled programs 192
  capacity of 194
  data sets 45,46
  definition of 10
  input
    additional 148,150
    primary 148-150
  job control statements, example of 78,79
  LIBRARY control statement 139
  messages 105
  options 18,22,27
  output 102-105
  programs 148
  processing
    example of 151
    overlay 152,153
linkage registers 146
LINKLIB 46,79,137
LIST option 22,27,105
literal pool 18,100
literals
  size considerations 194
LOAD macro instruction 191
load module
  definition 10
  length of 117
  output 105,106
  specifications in EXEC Statement 18
LOAD option 22,27,78,102
logical record length
  in file processing techniques (see LRECL)
logical record size
  for SYSIN 189
  for SYSLIB 189
  for SYSPRINT 189
  for SYSPUNCH 189
LRECL 41,182-185

Machine considerations 161-163
MACRF 182-187
macro instructions
  ATTACH 191
  CHKPT 95
  DCB 181
  ESETL 65
  LINK 153,191
  LOAD 191
  SETL 65
magnetic tape
  data sets
    sharing 135
    using DEN and TRTCH subparameters 51
  devices
    compiler optimization using 189
    labels 158,159
    in Sort feature 131,134,163
  volume 36
    removable 36
    reserved 36
map
  memory 98,167
  module 103,104
MAP option 21,22,27,99,100,104
master index 64
master scheduler 14
memory map 98,167
messages
  compiler, summary of 199-213
  format of, defined 199
  identification codes 106
  severity level of 20,101
  summary of 199-213
  types of
    allocation 97,99,103,105
    diagnostic 97,99,101
    disposition 102-105
    error 193
    operator 106
MFT (Multiprogramming with a fixed number of tasks)
  data sets
    intermediate 162
    marking end of 40
    scratching 118
    sharing 38
  description of 10
  JOB Statement parameters 17,18
  with multiple OPEN and CLOSE Statements 162
MOD subparameter 38
  in checkpoint/restart 92
  in compilation 41
  in linkage editing 46
MODE subparameter 51
module map 103,104
MOVE Statement 128
MOVE subroutine 179
MSGCLASS parameter 17,26
MSGLEVEL parameter 16,26
  required in restart 20
MSWA 186
Multiple OPEN and CLOSE Statements
  under MFT and MVT 162
Multiprogramming with a fixed number of tasks (see MFT)
Multiprogramming with a variable number of tasks (see MVT)
MVT (Multiprogramming with a variable number of tasks)
  causing errors 112
  checkpoint/restart with 91-95
  data sets
    intermediate 162
    marking end of 40
    scratching 118
    sharing 38
  description of 10
  EXEC Statement parameters 20,23,24
  input stream in 30
  JOB Statement parameters 17,18
  with multiple OPEN and CLOSE Statements 162
  region code 11
  REGION parameter 17,23,26,27,161,162
MXIG 33

name field in job control language 14,15
names
  cataloged procedure 31
  data set, conventions used in 76
  generation 31
  procedure 193
  qualification of 31,193
  temporary 31
NCP 183,186
NEW subparameter 38
  used with checkpoint/restart 92
NL subparameter 38
NOBASIS (see BASIS option in Job Control Language)
NOCLIST (see CLIST option)
NOCOPY (see COPY option in Job Control Language)
NODECK (see DECK option)
NODMAP (see DMAP option)
NOLOAD option (see also LOAD option)
  in cataloged procedures, example 86
NOMAP (see MAP option)
nontemporary data set 35
NOPMAP (see PMAP option)
NOSEQ (see SEQ option)
NOSOURCE option (see also SOURCE option) 99,189
NOSUPMAP (see SUPMAP option)
NSL subparameter 38
NTM parameter 64,184
numeric picture
  error messages 205

object code
  sample 168
object code listing 98,100
object module
  cataloging 78
  definition of 10
  dumps using 112-117
  error messages 213
  size considerations 194
object module deck 102
object module listing 102
OCCURS clause
  causing errors 111
  efficient use of 119-121

OCCURS DEPENDING ON clause 177
efficient use of 119-121
OLD subparameter 38
in compilation 42
in linkage editing 45
ON Statement 107-110
ONSIZE ERROR option 126
intermediate results 128
OPEN Statement 128
under MFT and MVT 162
Operand field in job control language 15
Operating System Environments 10,11
operation field in job control language 15
operator messages 106
OPTCD subparameter 51,56,61,62,65, 182-184,187
optional services (see OPTCD subparameter)
options
compiler 27
linkage editor 18,22,27
ORGANIZATION clause 49
in file processing techniques 57,59,182-187
Output 97
compiler 97-102
linkage editor 102-105
MAP option 21
requesting 22,106
suppressing 189
Output class
specifying on JOB Statement 17
overflow area (see QISAM)
overlay design 145
overlay structure 152,153
OVFLOW 60,61,63
OVLY option 22,27

page breaks, used in Report Writer 130
parameters
compared to arguments 146
keyword 15
positional 15
PARM option 21,22,27
partitioned data set 10
directory 137
minimum requirement for 33
member 10,137
primary quantity for 33
secondary quantity for 33
PASS subparameter 39
use of 35
PCP (see Primary Control Program)
PDS (see partitioned data set) 10
PGM 18,19,27
PGT (see Program Global Table)
physical records
size restrictions 34
PICTURE Clause
efficient use of 122
error messages 208
storage allocation 127
PMAP option 21,27,88,100
PRESRES, member of SYS1.PROCLIB 35,36
Primary Control Program
description 10
PRIME
in QISAM 61
prime area (see QISAM)
printer
determining line spacing 51
Priority scheduling system
JOB Statement parameters 16-18
relationship to multiprogramming environments 14
sharing data sets 38
SYSOUT parameter for 39
PRIVATE subparameter 37
private volume 36
PROC 18,27
Procedure Division
programming techniques 126-128
Procedure library 138
PROCLIB (see SYS1.PROCLIB)
program
called 145
defined 145
calling 192
defined 145
linkage editing 148
sample 165-175
selective testing of 109
Program Global Table 98,100
sample 167
PRTSP subparameter 51
PRTY parameter 17,26
public volume 36

Q routines 177
QISAM 58-67
considerations when using 64,65
data control block 30,60
data sets
creating 60-62
definition of 9
deleting records in 65
reorganizing 65
DD Statement parameters 30,60
error processing for 157,158,196,198
indexes, description of 59
overflow area, description of 60
prime area, description of 59
QSAM 49-53
data control block 182
data set
definition of 9,50
DD Statement parameters 52
error processing for 156,157,195
sample COBOL coding 165,166
Sort feature uses of 131

RD parameter 16,27,93
in EXEC Statement 20
in JOB Statement 26
READ Statement
in BISAM 66
causing errors 111,112
in QISAM 64
READY TRACE Statement 107
sample 166
RECFM 47,182-185,187
in compilation 189
in Sort feature 132,133
record
addressing 9,54,58
restriction with WRITE Statement 58
blocked 47

dummy 54
duplicate 198
formats 50
RECORD CONTAINS clause
in file processing techniques 182-185
used by Report Writer feature 129
RECORD KEY clause
in BISAM 58,66
in QISAM 58,64,65
record size, logical 189
REDEFINES clause
efficient use of 121
REF parameter 29
REF subparameter 32,37
REGION parameter 17,27
in EXEC Statement 23
in JOB Statement 26
used in compilation 161
used in execution 161,162
register assignment 119
Relative record addressing 54
Relative track addressing 53
relocation list dictionary 102
size considerations 194
Report Group descriptions 129
report picture,
error messages 206
Report Writer 129,130,99,102
size considerations 193
RERUN clause 16,20,91,180
RERUN subroutine 180
RESET TRACE 107
Restart (see also
Checkpoint/Restart) 93-95
automatic 93,94
deferred 94
RD parameter 20,93
RESTART parameter 17,26,94
restrictions 17
RETAIN subparameter 37
RETPD subparameter 38
return code 19,20,102,199
register 146
RETURN Statement 146
REWRITE Statement
in BISAM 66
in QISAM 64
RKP 184
RLD (see Relocation List Dictionary)
RLSE 33,53,160
ROLL parameter 18,27
in EXEC Statement 23,24
in JOB Statement 26
ROUND 33

Save area layout 148
SELECT Statement 48
relationship to DD statement 155,156
SEP parameter 28,31
SEQ option 22,27
sequential data set 9
sequential scheduler 14
SER parameter 29
SER subparameter 37
SETL macro instruction 65
severity levels 20,101
sharing data sets (see SHR subparameter)
SHR subparameter 38
sign
efficient use of 122
SIZE ERROR option 111
SIZE option 21,27,190
SL subparameter 38
SMSW 186
Sort feature 131-136
storage allocation for 163
Sort library 138
Sort subroutine 179
SORTIN 131
SORTIN DD Statement 134
SORTLIB DD Statement 135
SORTOUT 131
SORTOUT DD Statement 134
SORTWORKnn 131,132
SORTWKnn DD Statement 134
source module 77
definition 10
listing of 98,99
SOURCE option 21,27
source program library 139
(see also copy library)
calling 78
uses of SYSLIB 42
SPACE parameter 28,33
in BSAM 53,54,56
in creating data sets 69,70
in MFT and MVT 162
in QISAM 61
in Sort feature 133
in SYSABEND DD statement 106
SPACEn option 22,27,97
special characters
in job control language 27
SPLIT parameter 28,34
in creating data sets 69,70
in QISAM 61
STACK subparameter 51
STOP RUN
under MVT 112,162
Storage allocation
for compiler 41,161,193,194
for execution job step 162
for linkage editing 194
for overlay processing 152,153
for Sort feature 132,134,163
for source program 193,194
storage volume 36
SUBALLOC 28,34
in creating data sets 69,70
subparameters 15
subroutine library 177-180,46,138
subroutines
arithmetic 177,179
conversion 178,179
input/output 177
subscripts
in OCCURS clause 119
SUM Statement 129
SUPMAP option 22,27,100,102,189
SYMBOLIC KEY clause 54,57-59,66,187
SYNAD 182-185
SYSABEND DD Statement 48,80
under MVT 106
SYSCHK DD Statement 94
SYSCP 13,42
SYSDA 13,32,45,84

SYSIN DD Statement 13,30,42,77,80
in cataloged procedures 85,89
in compilation 41
concatenating with SYSLIN 90
logical record size for 189
relationship to ACCEPT Statement 48
under MFT and MVT 162
SYSLIB DD Statement 42,78
in cataloged procedures 88
in compilation 42,43
in linkage editing 45,46
record size for 189
SYSLIN DD Statement 22,42,78,102
in compilation 42
concatenating with SYSLIN 90
in linkage editing 45
record size for 189
SYSLMOD DD Statement 79,80,142
in linkage editing 45,46
SYSOUT Parameter 13,29,39,41,42,75
relationship to DISPLAY Statement 47
under MFT and MVT 162
used in Sort feature 135
SYSPRINT DD Statement 13,75
in compilation 41,42
in linkage editing 45
record size for 189
SYSPUNCH DD Statement 13,22,42,75,77,80
in compilation 42
record size for 189
relationship to DISPLAY Statement 47
SYSSQ 13,32,45,84
system catalog
creating 9
system diagnostic messages 105
SYSUT1 13
in compilation 41,42,161
in linkage editing 45,46
SYSUT2 (see SYSUT1)
SYSUT3 (see SYSUT1)
SYSUT4 (see SYSUT1)
SYS1.COBLIB 138,177
SYS1.LINKLIB 46,138
SYS1.PROCLIB 81,138
adding procedures to 82
SYS1.SORTLIB 138
storage allocation for 163

TALLY
in Report Writer 129
in Sort feature 136
tape (see magnetic tape)
Task Global Table 98,100
sample 167
temporary data set 35
creating 71
temporary names 31
terminal error messages 20
TGT (see Task Global Table)
TIME parameter 23,27
TRACE Statement 80,107-110
relationship to SYSOUT DD Statement 48
sample 166
TRACE subroutine 177
track
addressing 9,53
capacity 55,56
TRACK-AREA Clause
in BISAM 67
private 36
TRANSFORM subroutine 180
TRK 33
TRTCH subparameter 51
TYPRUN parameter 17,26

unblocked records
permissible file techniques 50
UNCATLG subparameter 39,160
UNIT parameter 28
creating data sets with 69,70
description of 31
multi-volume data sets using 54
retrieving data sets with 73,74
Sort programs using 132,133
unit record devices
DD Statements for 74
USAGE Clause 100,111
efficient use of 123-126
USE AFTER ERROR 156-158
in file processing techniques 182-185
USING option 131
Utility programs
IEBUPDTE 82,83,139-142
IEHLIST 118
IEHMOVE 137
IEHPROGM 118
IHDFSORT 136

volume
definition of 9
direct-access 35,36
labels 158-160
magnetic tape 36
permanently resident 35
private 36
public 36
specific and nonspecific 35
storage 36
volume labels, standard 159
VOLUME parameter 29
creating data sets with 69,70
description of 35,36
retrieving data sets with 73,74
Sort programs using 133
volume table of contents (VTOC) 160
VTOC 160

Warning, used as a severity level
(W) 101,199
WRITE Statement
causing errors 111
WRITE AFTER ADVANCING 50
restriction with PRTSP subparameter 51

XREF option 22,27,104
used in cataloged procedures,
example 87

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
[USA Only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]